

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Методические указания
по выполнению лабораторных работ
по дисциплине «Современные технологии программирования»
для студентов направления подготовки
09.03.02 «Информационные системы и технологии»
Профиль «Прикладное программирование в интеллектуальных
информационных системах»

Ставрополь, 2026

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ.....	3
ЛАБОРАТОРНАЯ РАБОТА 1. КЛАССЫ.....	5
ЛАБОРАТОРНАЯ РАБОТА 2. ОБЪЕКТЫ.....	24
ЛАБОРАТОРНАЯ РАБОТА 3. СТРУКТУРА ПРОГРАММ.....	38
ЛАБОРАТОРНАЯ РАБОТА 4. НАСЛЕДОВАНИЕ РЕАЛИЗАЦИИ	49
ЛАБОРАТОРНАЯ РАБОТА 5. ИНТЕРФЕЙСЫ.....	72
ЛАБОРАТОРНАЯ РАБОТА 6. ОБОБЩЕННЫЕ ТИПЫ.....	81
ЛАБОРАТОРНАЯ РАБОТА 7. КОЛЛЕКЦИИ	91
ЛАБОРАТОРНАЯ РАБОТА 8. ГРАФИЧЕСКИЙ ИНТЕРФЕЙС ПОЛЬЗОВАТЕЛЯ. ОСНОВЫ ОБРАБОТКИ СОБЫТИЙ.....	119
ЛАБОРАТОРНАЯ РАБОТА 9. СОБЫТИЯ.....	159
ЛАБОРАТОРНАЯ РАБОТА 10. РАЗРАБОТКА ПРИЛОЖЕНИЙ В РАМКАХ ШАБЛОНА MODEL-VIEW-CONTROLLER	182
ЛАБОРАТОРНАЯ РАБОТА 11. ВЗАИМОДЕЙСТВИЕ С ФАЙЛОВОЙ СИСТЕМОЙ	200
ЛАБОРАТОРНАЯ РАБОТА 12. МНОГОПОТОЧНЫЕ ПРИЛОЖЕНИЯ JAVA.....	219
ЗАКЛЮЧЕНИЕ	236
СПИСОК ЛИТЕРАТУРЫ	237
ПРИЛОЖЕНИЕ 1. ИСХОДНЫЙ КОД КЛАССА «БИБЛИОТЕКА» НА ЯЗЫКЕ C#.....	239

ВВЕДЕНИЕ

Методические указания к лабораторным работам по дисциплине «Современные технологии программирования» направлено на формирование у студентов навыков разработки приложений с использованием базовых концепций объектно-ориентированного программирования. В рамках данного пособия рассматриваются следующие аспекты разработки приложений:

- использование классов для представления объектов предметной области;
- жизненный цикл объектов; особенности объявления, переопределения методов класса;
- особенности применения конструкторов;
- структура программ на языке Java; пакеты и подпакеты;
- механизмы наследования и полиморфного поведения;
- обобщенные классы;
- классы коллекций;
- интерфейсы;
- основы построения графического интерфейса пользователя.

Особое значение при подготовке специалиста в сфере информационных систем и технологий имеет практическая подготовка специалиста: производственные задачи автоматизации связаны с многопоточным программированием, использованием баз данных, применением сложных структур данных.

Популярность объектного подхода обусловлена объективными факторами усложнения программных систем и неуклонным повышением требований к интеллектуальности, производительности, эргономичности, доступности и адаптивности программного обеспечения и средств разработки.

Современные технологии программирования, как самостоятельное направление, на текущем этапе развития информационных технологий

занимает существенную долю всех доступных средств разработки, анализа, проектирования и моделирования.

Особая роль объектно-ориентированных технологий приводит к необходимости детального изучения принципов построения программных компонент информационных систем на базе объектных технологий. При этом Современные технологии программирования (ООП) – это только одно из нескольких самостоятельных направлений изучения и использования теории, в основе которой лежат термины «объект» и «класс». При употреблении термина «Современные технологии программирования» подразумевается вся совокупность всех языков программирования, которые по совокупным признакам можно отнести к данной группе. Такие языки относят к объектно-ориентированным языкам программирования (ООЯП). При этом ООП включает также различные технологии программирования, которые используются на практике при разработке приложений на ООЯП.

Современные технологии объектно-ориентированного программирования интенсивно развиваются – на данный момент программисту недостаточно понимать простейшие принципы ООП (инкапсуляция, полиморфизм, наследование). ООП, как технология, должна реагировать на появление новых требований современного высокотехнологичного мира: параллельный характер процессов в информационных системах; распределенный характер информационных систем; повышение требований к защищенности программного обеспечения: слияние различных технологий разработки приложений и востребованность унифицированного подхода к проектированию и разработке веб-приложений, сервисов, интерфейсов.

ЛАБОРАТОРНАЯ РАБОТА 1. КЛАССЫ

1.1. Цель и содержание

Цель лабораторной работы: изучение принципов определения классов.

Задачи лабораторной работы:

- научиться определять тип класса на языке высокого уровня;
- научиться объявлять поля данных и методы класса;
- научиться решать учебные задачи с использованием статических методов;
- научиться программировать приложения с использованием классов.

1.2. Формируемые компетенции

ОПК-4; ОПК-5; ОПК-6

1.3. Теоретическая часть

1.3.1. Синтаксис определения класса

Ключевым понятием объектно-ориентированного программирования является «класс». В данной лабораторной работе рассматриваются синтаксические правила оформления классов (преимущественно в языке Java), типы классов.

Рассмотрим пример определения класса на языке Java:

```

1
2 public class FlyingObject {
3
4     private String name;        // наименование объекта
5     private float velocity;     // скорость объекта
6
7     public void setName(String pName)
8     {
9         name = pName;
10    }
11
12    public float getVelocity()
13    {
14        return this.velocity;
15    }
16
17    public void IncreaseVelocity(float step)
18    {
19        velocity += step;
20    }
21
22    public void DecreaseVelocity(float step)
23    {
24        if (velocity - step >= 0)
25            velocity -= step;
26    }
27 }
28

```

Рисунок 1.1 – Пример определения класса на языке Java.

Рассмотрим класс `FlyingObject` (летательный объект) подробнее: внутри класса содержатся закрытые поля данных `name` (наименование) и `velocity` (скорость объекта). В классе определены также методы: `setName(String pName)` – устанавливает имя объекта в значение `pName`; `getVelocity()` – возвращает текущую скорость объекта; `IncreaseVelocity(float step)` – увеличение скорости объекта на величину `step`; `DecreaseVelocity(float step)` – уменьшение скорости объекта на величину `step`.

Данное определение класса целиком повторяет синтаксис объявления класса в языке C#. Приведем немного измененное представление класса `flyingObject` на языке C# с использованием свойства класса (заменяет метод `getVelocity` в коде Java):

```

9  class FlyingObject
10 {
11     private String name;           // наименование объекта
12     private float velocity;        // скорость объекта
13
14     0 references
15     public void setName(String pName)
16     {
17         name = pName;
18     }
19
20     0 references
21     public float Velocity
22     {
23         get { return velocity; }
24     }
25
26     0 references
27     public void IncreaseVelocity(float step)
28     {
29         velocity += step;
30     }
31
32     0 references
33     public void DecreaseVelocity(float step)
34     {
35         if (velocity - step >= 0)
36             velocity -= step;
37     }
38 }

```

Рисунок 1.2 – Пример определения класса на языке C#.

В рамках проектирования объектно-ориентированного приложения необходимо маркировать каждое внутренне поле класса (метод или поле данных) модификатором доступа `public`, `private`, `protected`.

В языках Java и C# все типы данных можно разделить на две группы: ссылочные типы и типы по значению. Классы – это пользовательские ссылочные типы.

Разработанный класс `FlyingObject` можно использовать – создавать переменные класса (экземпляры класса) или объекты.

На рис. 1.3 представлен код использования объекта класса `FlyingObject`. Класс – это шаблон, который позволяет задать правила построения экземпляров класса – объектов. Класс не хранит состояния (полям класса не присваиваются значения). Все значения, и состояние в целом, сохраняет только экземпляр класса (объект).

```

public class Starter {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        FlyingObject obj = new FlyingObject();
        obj.setName("Самолет");
        obj.IncreaseVelocity(100.0f);
        obj.DecreaseVelocity(30.0f);
        System.out.println(String.valueOf(obj.getVelocity()));
    }
}

```

Рисунок 1.3 – Простейший пример кода, использующего класс FlyingObject.

Представленный код имеет простую структуру и не требует дополнительных пояснений. Поясните, что будет выведено в консоле. В данном коде перед вызовом методов и установкой значений свойств, определенных в классе, создается экземпляр класса конструкцией:

```
FlyingObject obj = new FlyingObject( );
```

Таким образом, создание объекта класса производится с использованием зарезервированного слова «new». После слова «new» указано не просто имя класса, а специальный метод, который осуществляет создание объекта (фактически возвращает переменную типа FlyingObject).

С созданным объектом можно проводить манипуляции: инициализировать, изменять и считывать его закрытые (private) поля, посредством вызовов его открытых (public) методов.

Функция main (рис. 1.3) сама является методом класса Starter. Инкапсуляция точки входа в программу внутри класса используется и в языке C#. Но в языке Java каждый класс может иметь метод main (это повышает удобство блочного тестирования программ).

Данные классов также могут быть открытыми (public), но это считается плохим стилем программирования. Поля данных следует инкапсулировать внутри класса, организовав интерфейс класса через доступные внешним объектам методы.

Рассмотрим общую схему определения класса:

```
[public][final][abstract] class ИмяКласса
{
    [поля данных и методы класса ]
}
```

Определение класса имеет свои модификаторы доступности. Модификатор `public` определяет внешний (enclosing) класс, доступный из других модулей (в Java – из других пакетов).

1.3.2. Переменные класса

Общий вид определения поля данных класса:

```
[спецификатор] тип ИмяПеременной;
```

Значения спецификаторов определены в таблице 1.1.

Таблица 1.1 – Спецификаторы определения полей данных класса.

Спецификатор	Описание
<code>public</code>	Спецификатор помечает переменную как открытую – доступную на уровне экземпляра класса.
<code>private</code>	Переменная недоступна на уровне экземпляра класса, доступна только внутри класса, то есть видна методам данного класса.
<code>protected</code>	Аналогично спецификатору <code>private</code> , но видимость переменной распространяется на дочерние классы (расширяющие классы в Java).
<code>final</code>	Спецификатор указывает, что значение переменной присваивается в момент объявления переменной и не меняется в процессе работы приложения (аналог <code>const</code> в языке C#).
<code>static</code>	Спецификатор помечает поле как переменную уровня класса, а не экземпляра. Подобные перемене фактически не «принадлежат» экземпляру, а являются частью класса, как типа данных. Это значит, что для всех объектов данного класса <code>static</code> -переменная имеют одинаковое значение.

Подобное значение модификаторов доступности полей класса определено и в C#. В следующих лабораторных работах особенности применения Java-спецификаторов будут уточнены.

1.3.4. Методы класса

Кроме полей данных в классе определяются методы – функции вызываемые на объекте (экземпляре) или на классе (если они помечены как *static*).

Определение метода:

```
[спецификатор] тип_возврата ИмяМетода([тип имя_параметра]);
{
    // тело метода
}
```

На рис. 1.1 приведены примеры объявления методов с различной сигнатурой, на рис. 1.3 продемонстрированы вызовы методов. Следует обратить внимание на синтаксис вызова методов класса (рис. 1.3): метод вызывается как функция, определенная в рамках объектной переменной, то есть общий синтаксис вызова метода класса:

```
имя_объекта.имя_метода(...);
```

Подобный синтаксис используется и в языке C#.

Существует возможность создавать методы, не принадлежащие ни одному объекту данного класса, то есть методы, существующие только в самом классе, а не в его экземплярах. Такие методы объявляются со спецификатором *static*. К таким методам относится, например, метод *main*.

Рассмотрим пример объявления пользовательского статического метода (рис. 1.4). Попытаемся реализовать консольный вывод с помощью «обертывания» стандартного механизма вывода в консоль *System.out.println()* в пользовательский класс консольного вывода. Имена класса и методов возьмем из технологии .NET Framework (класс *Console* в языке C#).

```

2 public class Console {
3
4     public static void WriteLine(String mes)
5     {
6         System.out.println(mes);
7     }
8
9     public static void WriteLine(double dig)
10    {
11        WriteLine(String.valueOf(dig));
12    }
13
14    public static double GetMyFunc(double x)
15    {
16        return Math.PI * x * x;
17    }
18 }

```

Рисунок 1.4 – Определение статического метода.

Обратите внимание, что фактически, объявляется фабрика статических методов: один «узел» – класс Console, через который осуществляется доступ к некоторому функционалу через вызовы статических методов. Методы просты и в пояснении не нуждаются. Отличие, вносимое спецификатором метода static, проявляется при вызове (использовании) метода (рис. 1.5).

```

4     public static void main(String[] args) {
5
6         double x1 = 100, x2 = 200;
7         double y1;
8         Console.WriteLine("Start calculation!");
9         y1 = Console.GetMyFunc(x1);
10        Console.WriteLine(y1);
11        Console.WriteLine(Console.GetMyFunc(x2));
12        Console.WriteLine("Calculation completed!");
13
14    }

```

Рисунок 1.5 – Вызовы статических методов.

Сравните вызовы методов с вызовами на рис. 1.3. Результат работы программы представлен на рис. 1.6.

```

Start calculation!
31415.926535897932
125663.70614359173
Calculation completed!

```

Рисунок 1.6 – Консольный вывод.

Можно сформулировать следующие области применения статических методов:

- методу не требуется доступ к данным о состоянии объекта, так как все необходимые данные передаются явно через параметры метода;
- методу требуется доступ только к статическим полям класса.

1.3.5. Параметры методов

В настоящее время в языках высокого уровня применяются два механизма передачи параметров методам: вызов по ссылке и вызов по значению. В Java всегда используется вызов по значению. Это значит, что в теле метода оказываются копии переменных (такие же значения), которые передавались в качестве фактических параметров методу. Так как в Java существуют две категории объектов: примитивные типы, ссылочные типы, то следует учитывать следующие правила:

- метод не может изменять параметры примитивных типов;
- метод может изменять состояние объекта, передаваемого в качестве параметра;
- метод не может создавать в своих параметрах ссылки на новые объекты.

1.4. Оборудование и материалы

Для выполнения лабораторной работы рекомендуется использовать персональный компьютер со следующими характеристиками: 64-разрядный (x64) процессор с тактовой частотой 1 ГГц и выше, оперативная память – 1 Гб и выше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система Windows 7 и выше, Java Development Kit 8 и выше; IDE Eclipse for Java SE (or Java EE) developers.

1.5. Указания по технике безопасности

Студенты должны следовать общепринятой технике безопасности для пользователей персональных компьютеров. Не следует самостоятельно производить ремонт технических средств, установку и удаление программного обеспечения. В случае обнаружения неисправностей необходимо сообщить об этом администратору компьютерного класса (обслуживающему персоналу лаборатории).

1.6. Методика и порядок выполнения работы

Рассмотрим технологию создания классов с использованием интегрированной среды разработки Eclipse. Учащиеся могут выполнять лабораторную работу в любой среде разработки на языке Java. Допускается также выполнение лабораторных работ с использованием инструментария Microsoft Visual Studio 2013 на языке C#.

В среде IDE Eclipse:

1. Создайте новый проект приложения Java:

1.1 Выберите команду главного меню *File* → *New* → *Project...*

1.2 В открывшемся диалоговом окне (рис. 1.7) выберите «Java Project». Нажмите кнопку «Далее».

1.3 На следующей вкладке укажите имя проекта.

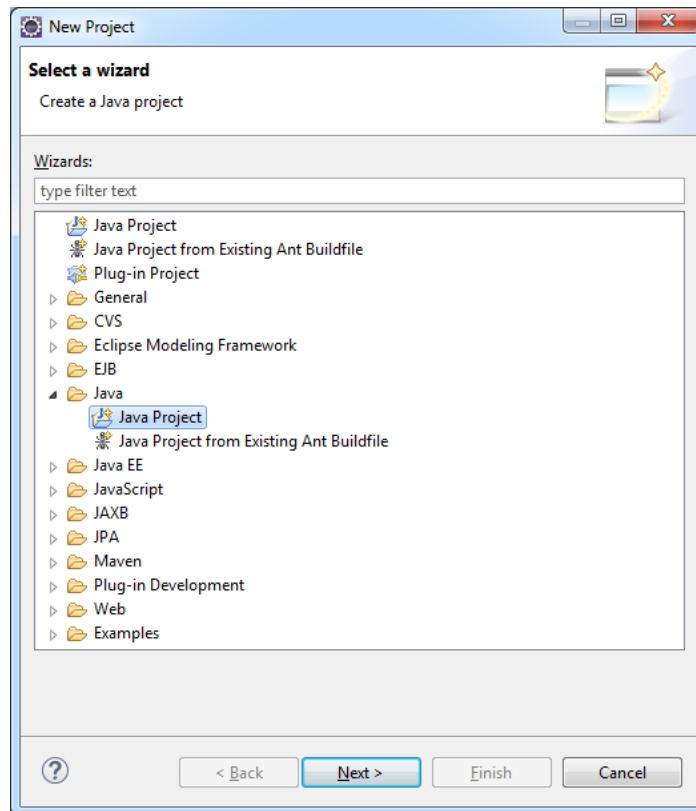


Рисунок 1.7 – Создание нового проекта консольного приложения.

2. После выполнения пункта 1 в среде разработки откроется новый созданный проект (рис. 1.8). Создан пустой проект, не содержащий файлов исходного кода.

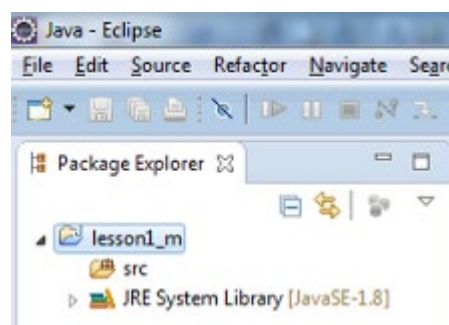


Рисунок 1.8 – Новый проект в окне «Обозреватель пакетов».

3. Создадим класс, содержащий функцию `main`, то есть фактически добавим функцию `main` в наше консольное приложение. Для этого откройте правой кнопкой мыши контекстное меню (рис. 1.9) проекта в окне «Обозреватель пакетов» (Package Explorer) и выберите команду «*New → Class*».

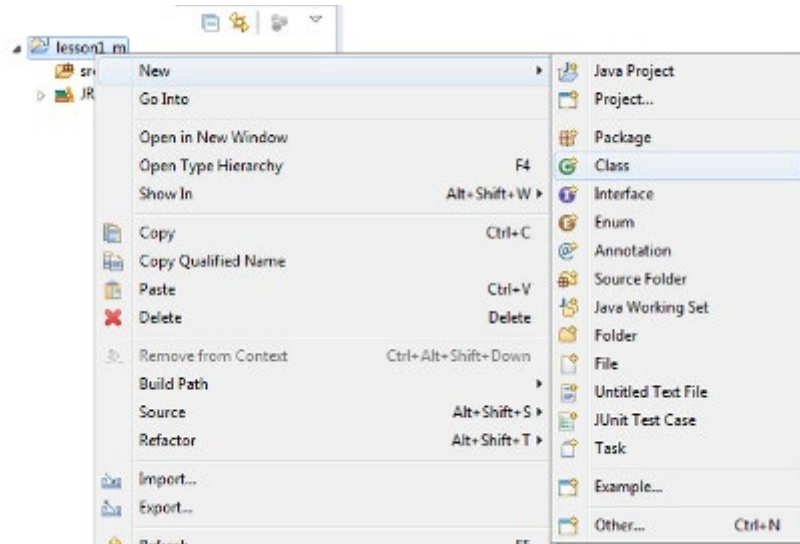


Рисунок 1.9 – Добавление нового элемента в проект.

4. В открывшемся диалоговом окне (рис. 1.10) можно сконфигурировать основные параметры класса: имя класса (поле Name); пакет, в котором будет размещен класс (поле Package); модификаторы доступа к классу; установить флажок, указывающий на необходимость создания функции main внутри нового класса. Нажмите кнопку «Закончить».

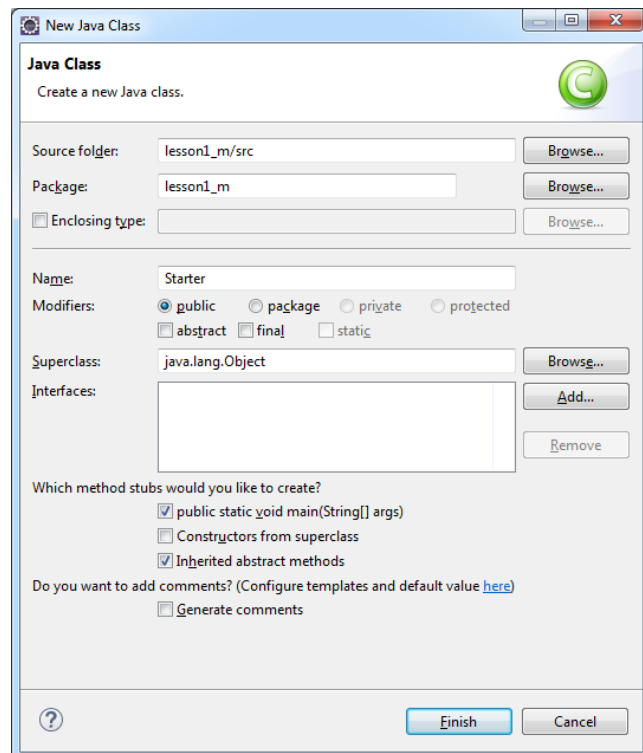


Рисунок 1.10 – Добавление нового элемента в проект.

5. В результате выполнения указанных действий будет создан новый класс с методом `main` и обозреватель проектов примет вид, указанный на рис. 1.11.

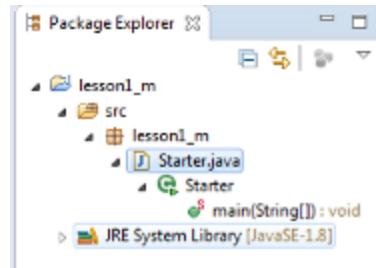


Рисунок 1.11 – Добавление нового элемента в проект.

6. Для закрепления принципов работы с классами выполним решение простой задачи:

Пользователь вводит случайные числа $a_i, i = \overline{1, N}$, программа выводит в консоль последовательность $b_i, i = \overline{1, N}$, где $b_i = \sum_{j=1}^i \frac{1-a}{1+a^2}$. Реализовать решение задачи с использованием методов класса в двух вариантах: с использованием статического метода и с использованием методов уровня объекта.

Для решения задачи модифицируем созданный проект следующим образом:

6.1 Создайте класс `MyTask` для демонстрации решения задачи с использованием статического метода.

6.2 В классе `MyTask` добавьте статический метод `DoTask()` решения задачи. Напишите код решения задачи (рис. 1.12). Рисунок 1.13 демонстрирует код функции `main`. На рис. 1.14 представлен результат выполнения приложения. Из исходного кода видно, что дополнительный класс фактически не нужен, то есть, код из метода `DoTask()` можно перенести в `main`.

```

1 package lesson1_m;
2
3 import java.util.ArrayList;
4 import java.util.Scanner;
5
6 public class MyTask {
7
8     public static void DoTask()
9     {
10         Scanner sc = new Scanner(System.in);
11         System.out.print("Введите количество чисел > ");
12         int N = sc.nextInt();
13
14         double a;
15         ArrayList<Double> aa = new ArrayList();
16
17         // заполняем список числами
18         for(int i = 0; i < N; i++) {
19             System.out.print("Введите число "+ String.valueOf(i+1) + " > ");
20             a = sc.nextDouble();
21             aa.add(a);
22         }
23
24         double b = 0;
25         System.out.println("результатирующая последовательность:");
26         // вывод результирующей последовательности
27         for (int i = 0; i < N; i++) {
28             b += (1-aa.get(i))/(1+aa.get(i)*aa.get(i));
29             System.out.println(b);
30         }
31
32         sc.close();
33     }
34 }

```

Рисунок 1.12 – Статический метод для решения задачи.

```

1 package lesson1_m;
2
3 public class Starter {
4
5     public static void main(String[] args) {
6         MyTask.DoTask();
7     }
8
9 }

```

Рисунок 1.13 – Функция main в случае использования статического метода.

```

Введите количество чисел > 5
Введите число 1 > 0,5
Введите число 2 > 0,7
Введите число 3 > 1
Введите число 4 > 67
Введите число 5 > 100
резльтирующая последовательность:
0.4
0.6013422818791947
0.6013422818791947
0.5866429500306424
0.5767439399316523

```

Рисунок 1.14 – Результат выполнения статического метода.

6.3 Выполним решение задачи без использования статического метода. Для этого создадим класс MyTask2Edit:

```

1 package lesson1_m;
2
3 import java.util.ArrayList;
4 import java.util.Scanner;
5
6 public class MyTask2Edit {
7     private ArrayList<Double> aa = new ArrayList<Double>();
8     private int N;
9
10    public void InputArray()
11    {
12        aa.clear();
13        Scanner sc = new Scanner(System.in);
14        System.out.print("Введите количество чисел > ");
15        N = sc.nextInt();
16
17        double a;
18        for(int i = 0; i < N; i++) {
19            System.out.print("Введите число " + String.valueOf(i+1) + " > ");
20            a = sc.nextDouble();
21            aa.add(a);
22        }
23        sc.close();
24    }
25
26    public void PrintResult()
27    {
28        double b = 0;
29        System.out.println("резльтирующая последовательность:");
30        for (int i = 0; i < N; i++) {
31            b += (1-aa.get(i))/(1+aa.get(i)*aa.get(i));
32            System.out.println(b);
33        }
34    }
35 }

```

Рисунок 1.15 – Код класса MyTask2Edit.

Код функции main представлен на рис. 1.16, результат работы программы – на рис. 1.17.

```

1 package lesson1_m;
2
3 public class Starter {
4     public static void main(String[] args) {
5         // MyTask.DoTask();
6         MyTask2Edit ob = new MyTask2Edit();
7         ob.InputArray();
8         ob.PrintResult();
9     }
10 }

```

Рисунок 1.16 – Использование класса MyTask2Edit.

```

Введите количество чисел > 3
Введите число 1 > 1
Введите число 2 > 0,8
Введите число 3 > ,9
резльтирующая последовательность:
0.0
0.12195121951219509
0.17719983829672548

```

Рисунок 1.17 – Результат работы программы.

7. Изучите код разработанных классов. Какой вариант реализации задачи лучше? Какие существуют недостатки в представленных вариантах решения задачи?

8. Выполните решение задачи в соответствии с индивидуальным вариантом. В каждом варианте необходимо разработать два варианта решения: с использованием исключительно статического метода и с применением методов уровня экземпляра.

Индивидуальное задание.

1. Пользователь вводит случайные числа $a_i, i = \overline{1, N}$, программа выводит последовательность $b_i, i = \overline{1, N}$, где $b_i = \sum_{j=1}^i a_j^2$.

2. Пользователь вводит числа $a_i, i = \overline{1, N}$, программа выводит сумму максимального и минимального числа, а также среднее арифметическое всех чисел.

3. Программа рассчитывает среднее арифметическое чисел $a_i, i = \overline{1, N}$, введенных пользователем.

4. Пользователь вводит случайные числа $a_i, i = \overline{1, N}$, программа выводит последовательность $b_i, i = \overline{1, N}$, где $b_i = \sum_{j=1}^i (a_j + j)$.

5. Пользователь вводит случайные числа $a_i, i = \overline{1, N}$, программа подсчитывает сумму квадратов двух чисел: максимального числа и следующего за ним по убыванию величины.

6. Пользователь вводит случайные числа $a_i, i = \overline{1, N}$, затем программа выводит список $b_i, i = \overline{1, N}$ в котором

$$b_i = \begin{cases} 0, & \text{если } a_i \text{ больше средн. арифметического;} \\ 1, & \text{если } a_i \text{ меньше средн. арифметического;} \\ -1, & \text{в остальных случаях.} \end{cases}$$

7. Пользователь вводит ряд целых случайных чисел $a_i, i = \overline{1, N}$, программа рассчитывает разность наибольшего и наименьшего числа.

8. Пользователь вводит случайные числа $a_i, i = \overline{1, N}$, программа рассчитывает куб суммы всех чисел.

9. Пользователь вводит случайные числа $a_i, i = \overline{1, N}$, программа выводит последовательность $b_i, i = \overline{1, N}$, где $b_i = \sum_{j=1}^i a_j^3$.

10. Пользователь вводит числа $a_i, i = \overline{1, N}$, программа выводит куб суммы чисел с четным номером i .

11. Пользователь вводит случайные числа $a_i, i = \overline{1, N}$, затем программа

выводит список $b_i, i = \overline{1, N}$ в котором $b_i = \begin{cases} 2, \text{ если } a_i \text{ делится на } 2; \\ 3, \text{ если } a_i \text{ делится на } 3; \\ 6, \text{ если } a_i \text{ делится и на } 2, \text{ и на } 3; \\ -1, \text{ в остальных случаях.} \end{cases}$

12. Пользователь вводит случайные числа $a_i, i = \overline{1, N}$, программа рассчитывает сумму минимального элемента и следующего за ним по величине элемента.

13. Пользователь вводит случайные числа $a_i, i = \overline{1, N}$, программа рассчитывает сумму квадратов максимального, минимального элементов и среднего арифметического всех элементов.

14. Пользователь вводит случайные числа $a_i, i = \overline{1, N}$, затем программа

выводит список $b_i, i = \overline{1, N}$ в котором $b_i = \begin{cases} a_i^2, \text{ если } a_i \text{ делится на } 2; \\ a_i^3, \text{ если } a_i \text{ делится на } 5; \\ a_i, \text{ если } a_i \text{ делится и на } 2, \text{ и на } 5; \\ a_i^{-1}, \text{ в остальных случаях.} \end{cases}$

15. Пользователь вводит случайные числа $a_i, i = \overline{1, N}$, программа

рассчитывает куб суммы чисел $b_i, i = \overline{1, N}$, где $b_i = \begin{cases} 1, \text{ если } a_i \text{ делится на } 2; \\ 2, \text{ если } a_i \text{ делится на } 3; \\ 3, \text{ если } a_i \text{ делится на } 5; \\ 0, \text{ в остальных случаях.} \end{cases}$

16. Пользователь вводит случайные числа $a_i, i = \overline{1, N}$, программа выводит последовательность $b_i, i = \overline{1, N}$, где $b_i = \sum_{j=1}^i a_j$.

17. Пользователь вводит случайные числа $a_i, i = \overline{1, N}$, программа выводит последовательность $b_i, i = \overline{1, N}$, где $b_i = \sum_{j=1}^i a_j - a_{N+1-j}$.

18. Пользователь вводит случайные числа $a_i, i = \overline{1, N}$, программа рассчитывает среднее арифметическое максимального и минимального элемента.

19. Пользователь вводит случайные числа $a_i; i = \overline{1, N}; N > 3$, программа выводит три максимальных элемента (максимальный элемент и следующие за ним по убыванию два элемента).

20. Пользователь вводит случайные числа $a_i; i = \overline{1, N}; 0 < a_i < 10$, программа выводит последовательность $b_i, i = \overline{1, N}$, где $b_i = \left(\sum_{j=1}^i a_j - j \right)^2$.

1.7. Содержание отчета и его форма

Отчет по лабораторной работе должен содержать:

1. Номер и название лабораторной работы; задачи лабораторной работы.
3. Ответы на контрольные вопросы.
4. Экранные формы (консольный вывод) и листинг программного кода с комментариями, показывающие порядок выполнения лабораторной работы, и результаты, полученные в ходе её выполнения.

Отчет о выполнении лабораторной работы, подписанный студентом, сдается преподавателю.

1.8. Контрольные вопросы

1. На рис. 1.4. приведено объявление класса. Сколь раз объявлены или вызваны статические методы в этом коде?
2. Для чего используется спецификатор `static` при объявлении полей классов? Можно ли статический метод класса вызвать через объект класса?
3. Поясните термины, используемые в данной лабораторной работе: «пространства имен», «класс», «метод», «функция», «экземпляр класса»,

«объект», «объектная переменная», «...доступен на уровне экземпляра», «...доступен на уровне класса».

4. В каких методах класса нельзя использовать зарезервированное слово `this`?

5. Назовите области применения статических методов класса.

6. Если в Java используется вызов методов по значению, почему объект, передаваемый в качестве параметра методу, может изменять свое состояние?

1.9. Список литературы

Для выполнения лабораторной работы, при подготовке к защите, а также для ответа на контрольные вопросы рекомендуется использовать следующие источники: [1-3].

ЛАБОРАТОРНАЯ РАБОТА 2. ОБЪЕКТЫ

2.1. Цель и содержание

Цель лабораторной работы: научиться проектировать и реализовывать приложения, ориентированные на использование объектов.

Задачи лабораторной работы:

- научиться создавать объекты с использованием конструкторов различной сигнатуры;
- научиться реализовывать этап объектного проектирования;
- научиться решать задачи автоматизации процессов предметной области с помощью приложений, функционирующих в рамках ООП-парадигмы.

2.2. Формируемые компетенции

ОПК-4; ОПК-5; ОПК-6

2.3. Теоретическая часть

2.3.1 Создание объектов. Конструкторы.

В предыдущих работах создание экземпляра типа (объекта) на базе определения типа (класса) уже было продемонстрировано (например, рис. 1.3). Создание объекта происходит с использованием ключевого слова `new`. На рис. 2.1 представлено определение класса, на рис. 2.2 создание объекта и вызов его метода.

```
1 package lesson2_Example;|
2
3 public class Example {
4     public void RepeatAfterMe(String str){
5         System.out.println(str);
6     }
7 }
```

Рисунок 2.1 – Определение класса Example.

```

1 package lesson2_Example;
2
3 public class Starter_Example {
4
5     public static void main(String[] args) {
6         Example ex = new Example();
7         ex.RepeatAfterMe("Say Hello!");
8     }
9 }

```

Рисунок 2.2 – Реализация типа Example.

Пример простой, но необходимо обратить внимание на строку 6 рис. 2.2, в которой производится создание объекта. Переменная *ex* содержит только адрес области памяти, в которой размещен экземпляр типа, построенный с использованием шаблона *Example*. Объект был создан с использованием конструкции:

Имя_Класса имя_переменной
 = new *Имя_Конструктора*([параметры]);

После зарезервированного слова *new* следует имя конструктора класса – специального метода, который не вызывается явно, а используется только при создании объекта совместно с *new*. На рис. 2.1 в определении класса метод с такой сигнатурой отсутствует – в языках Java и C# используется конструктор без параметров, присутствующий в классе по умолчанию. Хотя бы один конструктор должен присутствовать в каждом классе (явный или по умолчанию). Конструкторов в классе может быть несколько (определяется соображениями целесообразности). Несмотря на специальное назначение методов, для них действуют такие же правила перегрузки, как и для остальных методов класса. На рис. 2.3 представлено определение класса с несколькими конструкторами. Конструктор без параметров закомментирован. Конструктор с единственным параметром введен для инициализации закрытой переменной *fiо*. Конструктор с тремя параметрами осуществляет инициализацию трех закрытых полей: поле *fiо* будет проинициализировано с помощью вызова конструктора

этого же класса с другой сигнатурой; остальным полям значения будут присвоены явно.

```

1 package lesson2_Example;
2
3 public class Студент {
4     private String fio;
5     private String группа;
6     private float балл_по_математике;
7     private float балл_по_физике;
8     private float балл_по_программированию;
9     // Конструктор без параметров (не используется)
10    //Студент(){
11    //}
12    // Конструктор с параметром (ФИО студента)
13    Студент(String pFio){
14        fio = pFio;
15    }
16    // Конструктор с параметрами (ФИО, оценки по физик и математике)
17    Студент(String pFio, float bMath, float bPhys)
18    {
19        this(pFio);
20        балл_по_математике = bMath;
21        this.балл_по_физике = bPhys;
22    }
23    // Кнструктор с параметрами (ФИО и все оценки)
24    Студент(String pFio, float bMath,
25            float bPhys, float bProg){
26        this(pFio, bMath, bPhys);
27        балл_по_программированию = bProg;
28    }
29 }
30 }

```

Рисунок 2.3 – Определение класса «Студент».

Следует обратить внимание на использование зарезервированного слова `this`. В одном случае (строка 19 на рис. 2.3) `this` используется для вызова конструктора; в другом (строка 21) – для получения ссылки на текущий объект и, через ссылку, обращение к полю данного объекта. Второй вариант использования `this` применяют программисты если необходимо подчеркнуть принадлежность поля этому классу. На рис. 2.4 показано создание нескольких экземпляров класса «Студент». Обратите внимание, что конструктор по умолчанию становится недоступным после определения хотя бы одного явного конструктора. Если он необходим, то его нужно реализовать явно.

```

1 package lesson2_Example;
2
3 public class Starter_Example {
4
5     public static void main(String[] args) {
6         Студент студент1 = new Студент();
7         Студент студент2 = new Студент("Никитин Н.К.");
8         Студент студент3 = new Студент("Иванова Е.Е.",
9             89.11f, 97.00f);
10        Студент студент4 = new Студент("Климов А.С.",
11            91.30f, 11.70f, 56.50f);
12    }
13 }

```

Рисунок 2.4 – Использование класса «Студент».

Для сравнения, на рис. 2.5 представлено определение класса на языке C#. Создание объектов синтаксически не отличается.

```

9 class Студент
10 {
11     private String fio;
12     private String группа;
13     private float балл_по_математике;
14     private float балл_по_физике;
15     private float балл_по_программированию;
16
17     // Конструктор с параметром (ФИО студента)
18     1 reference
19     public Студент(String pFio)
20     {
21         fio = pFio;
22     }
23     // Конструктор с параметрами (ФИО, оценки по физике и математике)
24     1 reference
25     public Студент(String pFio, float bMath, float bPhys)
26         :this(pFio)
27     {
28         балл_по_математике = bMath;
29         this.балл_по_физике = bPhys;
30     }
31     // Конструктор с параметрами (ФИО и все оценки)
32     0 references
33     public Студент(String pFio, float bMath,
34         float bPhys, float bProg)
35         :this(pFio, bMath, bPhys)
36     {
37         балл_по_программированию = bProg;
38     }
39 }

```

Рисунок 2.5 – Определение класса «Студент» на языке C#.

Сравните определения классов на рис. 2.3 и рис. 2.5. Подумайте, почему в Java перед конструкторами не указан модификатор доступа public.

При программировании классов на языках C# или Java следует понимать отличие объектов Java (C#) от ссылок C++: в C++ такая переменная не может иметь значение null; объектные переменные Java – это указатели на объекты. Все объекты в языке Java располагаются в динамической памяти. Если объект содержит другую объектную переменную, она представляет собой всего лишь указатель на другой объект, расположенный в области памяти под названием «куча» (heap).

2.3.2 Декомпозиция. Проектирование классов.

Современные корпоративные информационные системы предоставляют пользователю большое количество сервисов и манипулируют большим количеством разнообразной информации. В разработке таких сложных систем участвует большое количество программистов-разработчиков. Информационные системы состоят из большого количества подсистем, модулей, которые взаимодействуют между собой через ограниченное количество интерфейсов (методов). Использование в таких системах принципов объектно-ориентированного программирования позволяет сделать приемлемыми сроки проектирования, разработки, модификации. Один из таких принципов – декомпозиция.

Декомпозиция – это определение физических и логических сущностей, а также принципов их взаимодействия на основе анализа предметной области создаваемой системы.

Все системы в ООП-парадигме состоят из классов. При проектировании приложения следует придерживаться следующих принципов; их использование позволяет сделать код более профессиональным и качественным и обеспечить простое сопровождение, поддержку и модификацию:

1. Имя класса должно отображать его роль, назначение.
2. Разрабатывать класс нужно таким образом, чтобы внесение в него изменений происходило относительно просто.
3. Назначение класса должно быть простым.

4. Код конструктора должен только инициализировать состояние объекта, в нем не должно быть длительных операций. Необходимо избегать вызовов других методов в классе (за исключением других конструкторов и методов `final`).

5. Необходимо использовать инкапсуляцию нестатических и неконстантных полей.

6. необходимо избегать слишком длинных методов.

7. Необходимо избегать длинных списков аргументов (возможно следует объединять группы аргументов в новый тип).

8. Если метод используется только другими методами данного класса, то следует объявлять его как `private`.

2.4. Оборудование и материалы

Для выполнения лабораторной работы рекомендуется использовать персональный компьютер со следующими характеристиками: 64-разрядный (x64) процессор с тактовой частотой 1 ГГц и выше, оперативная память – 1 Гб и выше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система Windows 7 и выше, Java Development Kit 8 и выше; IDE Eclipse for Java SE (or Java EE) developers.

2.5. Указания по технике безопасности

Студенты должны следовать общепринятой технике безопасности для пользователей персональных компьютеров. Не следует самостоятельно производить ремонт технических средств, установку и удаление программного обеспечения. В случае обнаружения неисправностей необходимо сообщить об этом администратору компьютерного класса (обслуживающему персоналу лаборатории).

2.6. Методика и порядок выполнения работы

Для закрепления теоретического материала и выполнения задач лабораторной работы рассмотрим пример решения задачи.

Условие. Разработать приложение для автоматизации рабочего места сотрудника библиотеки. Задача является учебной и необходимо выделить ряд объектов предметной области; определить действия, которые они реализуют; установить интерфейсы их взаимодействия.

Решение. Представим предметную область совокупностью следующих классов: «Книга» (рис. 2.6), «Читатель» (рис. 2.8), «Библиотека» (рис. 2.9). Затем реализовать перечень необходимых операций (методов) для изменения состояния объектов.

```

1 package lesson2;
2
3 public class Книга {
4     String название;
5     String автор;
6     String код;
7     Жанр жанр;
8     Состояние состояние;
9 }

```

Рисунок 2.6 – Определение класса «Книга».

Дополнительно определим два перечня: «Состояние» и «Жанр» (рис. 2.7). Наравне с классами в формировании структуры программы принимают участие и другие типы: перечисления, структуры, интерфейсы, события.

```

1 package lesson2;
2
3 public enum Состояние {
4     НаРуках,
5     Доступна
6 }

```

а)

```

1 package lesson2;
2
3 public enum Жанр {
4     Детектив,
5     Триллер,
6     Образование,
7     Информатика,
8     Физика,
9     Психология
10 }

```

б)

Рисунок 2.7 – Вспомогательные типы-перечисления: а – описывает состояние

объекта; б – описывает жанровую принадлежность книги.

Обоснованное использование перечней также является хорошим признаком качества программного кода. В данном случае программист достигает консолидации всей информации о жанрах книг и всех возможных состояниях каждой книги всего в двух структурах данных.

```

1 package lesson2;
2
3 public class Читатель {
4     String фио;
5     String билет;
6     public Читатель(String фиоНовый,
7                     String билетНовый) {
8         фио = фиоНовый;
9         билет = билетНовый;
10    }
11 }

```

Рисунок 2.8 – Определение класса «Читатель».

В классе «Библиотека» инкапсулированы методы, выполняющие основные функции автоматизации поиска и учета книг.

```

1 package lesson2;
2
3 import java.util.Hashtable;
4
5
6
7
8
9
10 public class Библиотека {
11
12     Hashtable<Книга, Vector<Читатель>> книги =
13         new Hashtable<Книга, Vector<Читатель>>();
14
15     private Библиотека() { }
16
17     public static Библиотека Инициализация(){
18         Библиотека ges = new Библиотека();
19         Книга новый = new Книга();
20         новый.автор = "Коротков А.Е.";
21         новый.жанр = Жанр.Образование;
22         новый.код = "001";

```

```

23     новый.название = "Программирование C#";
24     новый.состояние = Состояние.Доступна;
25     res.книги.put(
26         новый,
27         new Vector<>());
28     новый = new Книга();
29     новый.автор = "Тодеев А.А.";
30     новый.жанр = Жанр.Образование;
31     новый.код = "002";
32     новый.название = "Java. Учебник";
33     новый.состояние = Состояние.Доступна;
34     res.книги.put(
35         новый,
36         new Vector<>());
37
38     return res;
39 }

41 public void ВыдачаКниги(String кодКниги, Читатель читатель) {
42     // Ищем книгу с переданным кодом
43     Optional<Entry<Книга, Vector<Читатель>>> find =
44         книги.entrySet().stream()
45             .filter(e->e.getKey().код.equals(кодКниги)).findFirst();
46
47     // Если книга нашлась
48     if(find.isPresent()){
49         // проверяем доступность
50         if(find.get().getKey().состояние == Состояние.Доступна){
51             find.get().getKey().состояние = Состояние.НаРуках;
52             find.get().getValue().add(читатель);
53             System.out.println("Выдача книги зафиксирована.");
54         }
55         else
56             System.out.println("Книга на руках!!!");
57     }
58
59     else
60         System.out.println("Книги с таким кодом нет!!!");
61 }
62
63 public void ВозвратКниги(Читатель читатель) {
64     System.out.println("Функция не реализована!!!");
65 }
66
67 public void Каталог(){
68     книги.entrySet().stream().forEach(e->{
69         Книга tek = e.getKey();
70         System.out.println(tek.автор + "\n" +
71             tek.название + "\n" + tek.код + "\n");
72     });
73 }
74 }

```

Рисунок 2.9 – Определение класса «Библиотека».

Класс «Библиотека» содержит несколько методов, назначение которых не нуждается в пояснении. Но исходные коды методов Каталог() и ВыдачаКниги(...) содержат интересный пример использования лямбда-

выражений, которые в данном случае заменяют громоздкие конструкции циклов. Для сравнения в приложении 1 представлен исходный код класса «Библиотека», реализованного на языке C# и описывающего аналогичные методы.

На рис. 2.10 представлен код метода `main()`. Следует учитывать, что при практическом построении корпоративных приложений в рамках ООП-парадигмы такой большой объем кода в функции `main` является недопустимым. В данной функции должны только создаваться объекты (небольшое количество) вызываться их методы, запускающие задачи на выполнение. Постарайтесь исправить данный недостаток при реализации индивидуального задания.

```

1 package lesson2;
2
3 import java.util.Scanner;;
4
5 public class Starter {
6
7     public static void main(String[] args) {
8         Scanner sc = new Scanner(System.in);
9         Библиотека мояБиблиотека = Библиотека.Инициализация();
10        System.out.println("Библиотека готова к работе!");
11        System.out.println("Идентификация читателя:");
12        System.out.print("ФИО > ");
13        String фиоЧ = sc.nextLine();
14        System.out.print("Читательский билет > ");
15        String билетЧ = sc.next();
16        Читатель клиент = new Читатель(фиоЧ,билетЧ);
17        int операция = 0;
18        do{
19            System.out.print("Уважаемый " + клиент.фио +
20                ", Выберите действие:\n1 - Вывод каталога\n" +
21                "2 - Запрос книги\n3 - Сдача книги\n" +
22                "0 - выход\n Ваш выбор>");
23            операция = sc.nextInt();
24            switch(операция){
25                case 1: мояБиблиотека.Каталог(); break;
26                case 2:
27                    // какую книгу брать?
28                    System.out.print("Код книги ?> ");
29                    String код = sc.next();
30                    мояБиблиотека.ВыдачаКниги(код, клиент);
31                    break;
32                case 3:
33                    мояБиблиотека.ВозвратКниги(клиент);
34                    break;
35                default:break;
36            }
37        } while (операция != 0);
38
39        sc.close();
40        System.out.println("Библиотека. Работа завершена!");
41    }
42 }

```

Рисунок 2.10 – Основная программа.

На рис. 2.11 показан фрагмент консольного вывода приложения.

```

Библиотека готова к работе!
Идентификация читателя:
ФИО > Новиков И.Е.
Читательский билет > 999-7
Уважаемый Новиков И.Е., Выберите действие:
1 - Вывод каталога
2 - Запрос книги
3 - Сдача книги
0 - выход
Ваш выбор>1
Коротков А.Е.
Программирование С#
001

Тодеев А.А.
Java. Учебник
002

Уважаемый Новиков И.Е., Выберите действие:
1 - Вывод каталога
2 - Запрос книги
3 - Сдача книги
0 - выход
Ваш выбор>0
Библиотека. Работа завершена!

```

Рисунок 2.11 – Консольный вывод программы.

Внимательно изучите код полученной программы, Затем выполните индивидуальное задание.

Индивидуальное задание.

В каждом приложении необходимо выделить классы, необходимые для описания предметной области; разработать структуру каждого класса (необходимые методы и поля). После этого реализуйте приложение с использованием разработанных типов данных. Всего необходимо реализовать не менее 3–4 классов в рамках автоматизации задачи.

Вариант	Описание предметной области
1	Учет поступления продуктов питания на склад
2	Учет взаиморасчетов с клиентами
3	Система поддержания товарно-материальных запасов
4	Система управления взаимоотношениями с клиентами
5	Учет сетевого и компьютерного оборудования
6	Электронный журнал учебного заведения
7	Учет поступления налоговых платежей от контрагентов
8	Система анализа сетевых запросов

9	Система управления роботом-манипулятором
10	Система планирования работ в коллективе исполнителя
11	Учет товаров в разрезе товарных групп
12	Система учета платежей за услуги провайдера
13	Система анализа банковских счетов клиентов
14	Пропускная система на предприятии
15	Учет текущей аттестации студентов
16	Система учета фактов правонарушений на дорогах
17	Билетная система кинотеатра
18	Система заказов интернет-магазина
19	Электронная очередь частной клиники
20	Учет автомобильных комплектующих
21	Система анализа кредитоспособности клиентов банка
22	Учет поступления комплектующих в сборочный цех
23	Анализ эффективности менеджеров
24	Учет финансовых операций предприятия
25	Система поддержания запасов стройматериалов

2.7. Содержание отчета и его форма

Отчет по лабораторной работе должен содержать:

1. Номер и название лабораторной работы; задачи лабораторной работы.
3. Ответы на контрольные вопросы.
4. Экранные формы (консольный вывод) и листинг программного кода с комментариями, показывающие порядок выполнения лабораторной работы, и результаты, полученные в ходе её выполнения.

Отчет о выполнении лабораторной работы, подписанный студентом, сдается преподавателю.

2.8. Контрольные вопросы

1. В каких случаях при создании объекта не производится вызов конструктора?
2. Сравните синтаксис определения и вызова конструктора класса на языках C# и Java? Существуют ли синтаксические различия?

3. Что такое конструктор по умолчанию? Когда конструктор по умолчанию становится недоступным?
4. Какие типы данных кроме классов существуют в Java?
5. Сколько конструкторов может иметь класс? Что такое «сигнатура метода»? Чем отличаются конструкторы одного и того же класса?
6. Поясните принципы выделения отдельных сущностей предметной области в классы. Опишите рекомендации, которыми нужно при этом руководствоваться.

2.9. Список литературы

Для выполнения лабораторной работы, при подготовке к защите, а также для ответа на контрольные вопросы рекомендуется использовать следующие источники: [1], [7].

ЛАБОРАТОРНАЯ РАБОТА 3. СТРУКТУРА ПРОГРАММ

3.1. Цель и содержание

Цель лабораторной работы: изучить принципы структурной декомпозиции приложений.

Задачи лабораторной работы:

- научиться создавать и импортировать пакеты;
- изучить принцип структуризации стандартных библиотек Java и .NET Framework;
- научиться формировать структуру программных модулей при разработке сложного приложения.

3.2. Формируемые компетенции

ОПК-4; ОПК-5; ОПК-6.

3.3. Теоретическая часть

3.3.1 Основы структурной организации исходного кода.

В предыдущих лабораторных работах создавались классы и демонстрировались методы использования возможностей стандартной библиотеки Java SE. Для эффективной разработки и поддержки ООП-приложений необходимо принимать дополнительные меры по структурированию кода. Необходимо понимать, что классы следует определять в отдельных файлах (для java – это файлы с расширением *.java; при использовании C# – *.cs). Распределение классов по отдельным файлам не единственная возможность для программиста для оптимизации исходных кодов приложения.

В стандартную библиотеку Java API входят сотни классов. Каждый программист в ходе работы добавляет к ним десятки своих классов. Множество классов растет и становится неуправляемым. Уже давно принято отдельные классы, решающие какую-то одну определенную задачу, объединять в библиотеки классов (объединение по функциональному признаку). Но библиотеки классов, кроме стандартной библиотеки, не являются частью языка.

Разработчики Java включили в язык дополнительную конструкцию – пакеты (packages). Все классы Java распределяются по пакетам. Кроме классов пакеты могут содержать интерфейсы и вложенные подпакеты (subpackages). Образуется древовидная структура пакетов и подпакетов.

Эта структура в точности отображается на структуру файловой системы. Все файлы с расширением class (содержащие байт-коды), образующие один пакет, хранятся в одном каталоге файловой системы. Подпакеты образуют подкаталоги этого каталога.

Каждый пакет создает одно пространство имен (namespace). Это означает, что все имена классов, интерфейсов и подпакетов в пакете должны быть уникальны. Имена в разных пакетах могут совпадать, но это будут разные программные единицы. Таким образом, ни один класс, интерфейс или подпакет не может оказаться сразу в двух пакетах. Если надо в одном месте программы использовать два класса с одинаковыми именами из разных пакетов, то имя класса уточняется именем пакета: <имя_пакета>.<Имя_класса>. Такое уточненное имя называется полным именем класса (fully qualified name).

Все эти правила, опять-таки, совпадают с правилами хранения файлов и подкаталогов в каталогах, только в файловых системах для разделения имен каталогов в пути к файлу обычно используется наклонная черта или двоеточие, а не точка.

Еще одно отличие от файловой системы – подпакет не является частью пакета, и классы, находящиеся в нем, не относятся к пакету, а только к подпакету. Поэтому, для того чтобы создать подпакет, не надо предварительно создавать пакет. С другой стороны, включение в программу пакета не означает

включение его подпакетов. Пакетами пользуются еще и для того, чтобы добавить к уже имеющимся правам доступа к членам класса `private`, `protected`. и `public` (уже рассматривались в предыдущих лабораторных). В Java существует еще один, «пакетный» уровень доступа.

Если член класса не отмечен ни одним из модификаторов `private`, `protected`, `public`, то по умолчанию к нему осуществляется пакетный доступ (`default access`), т. е. к такому члену может обратиться любой метод любого класса из того же пакета. Пакеты ограничивают и доступ к классу целиком – если класс не помечен модификатором `public`, то все его члены, даже открытые, `public`, не будут видны из других пакетов.

Следует обратить внимание на то, что члены с пакетным доступом не видны в подпакетах данного пакета.

3.3.2 Пакеты и подпакеты

Чтобы создать пакет, необходимо в первой строке `java`-файла с исходным кодом записать строку

`package имя_пакета;`

например:

```
package lesson2;
```

Тем самым создается пакет с указанным именем `lesson2` и все классы, записанные в этом файле, попадут в пакет `lesson2`. Повторяя эту строку в начале каждого исходного файла, включаем в пакет новые классы.

Имя подпакета уточняется именем пакета. Чтобы создать подпакет с именем, например, `subpack`, следует в первой строке исходного файла написать:

`package имя_пакета. имя_подпакета;`

и все классы этого файла и всех файлов с такой же первой строкой попадут в данный подпакет. Можно создать и подпакет подпакета произвольной вложенности.

Поскольку строка «`package имя_пакета;`» только одна и это обязательно первая строка файла, каждый класс попадает только в один пакет или подпакет.

Соглашение «Code Conventions» рекомендует записывать имена пакетов строчными буквами. Тогда они не будут совпадать с именами классов, которые, по соглашению, начинаются с прописной буквы. Кроме того, соглашение советует использовать в качестве имени пакета или подпакета доменное имя своего сайта, записанное в обратном порядке, например:

```
com.sun.developer
```

Это обеспечит уникальность имени пакета во всем Интернете.

Вы можете не указывать пакет при использовании интегрированной среды разработки. Компилятор всегда создает для таких классов безымянный пакет (unnamed package), которому соответствует текущий каталог (current working directory) файловой системы.

Безымянный пакет служит обычно хранилищем небольших пробных или промежуточных классов. Большие проекты лучше хранить в пакетах. Более того, некоторые программные продукты Java вообще не работают с безымянным пакетом. Поэтому в технологии Java рекомендуется все классы помещать в пакеты.

Библиотека классов Java SE 7 API хранится в пакетах java, javax, org. Пакет java содержит только подпакеты applet, awt, beans, dyn, io, lang, math, net, nio, rmi, security, sql, text, util и ни одного класса. Эти пакеты имеют свои подпакеты, например пакет создания графического пользовательского интерфейса и графики java.awt содержит классы, интерфейсы и подпакеты color, datatransfer, dnd, event, font, geom, im, image, print. Необходимо понимать, что количество и состав пакетов Java SE API меняется с каждой новой версией.

3.3.3 Права доступа к членам класса.

В предыдущих лабораторных работах при написании кода спецификаторы доступа указывались как перед определением классов, так и перед полями и методами, принадлежащими классам. Для усвоения принципов распределения классов по пакетам рассмотрим пример:

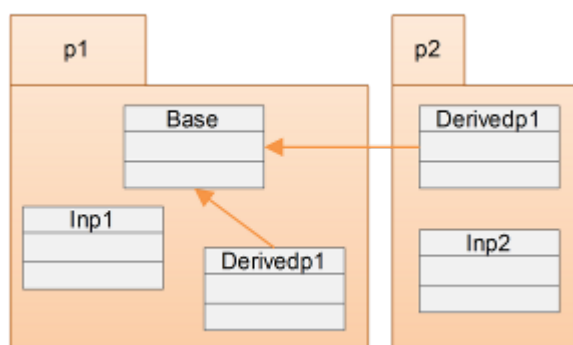


Рисунок 3.1 – Распределение классов по пакетам.

В файле Base.java описаны три класса: Inp1, Base и класс Derivedp1, расширяющий класс Base. Эти классы размещены в пакете p1.

В файле Inp2.java описаны два класса: Inp2 и класс Derivedp2, расширяющий класс Base. Эти классы находятся в другом пакете p2. Класс Base должен быть помечен при своем описании в пакете p1 модификатором public, иначе из пакета p2 не будет видно ни одного его члена.

В пакете p2, доступ ограничен в большей степени. Из независимого класса можно обратиться только к открытым, public, полям класса другого пакета. Из подкласса можно обратиться еще и к защищенным, protected, полям, но только унаследованным непосредственно, а не через экземпляр суперкласса.

Все указанное относится не только к полям, но и к методам.

В таблице 3.1 суммируется информация о доступности полей и методов класса в зависимости от контекста и модификатора доступности.

Таблица 3.1 – Особенности доступа к полям класса.

	Класс	Пакет	Пакет и подклассы	Все классы
private	+			
«package»	+	+		
protected	+	+	*	
public	+	+	+	+

3.3.4 Импорт пакетов и классов

В коде на языке Java присутствует оператор `import`, имеющий отношение к структурной организации исходного кода.

Компилятор «видит» типы, определенные только в текущем пакете и стандартном пакете `java.lang`. С целью использования всех возможностей библиотеки, а также для сокращения имен типов (чтобы не писать полное имя типа) необходимо указывать компилятору какие типы и пакеты будут использоваться в программе. Для этого используется оператор `import`. Указывается ключевое слово `import` и через пробел полное имя класса, завершенное точкой с запятой. Сколько классов надо указать, столько операторов `import` и пишется. Можно использовать конструкцию:

```
import имя_пакета.*;
```

Существует разновидность оператора импорта для поиска статических полей и методов. Например:

```
import static java.lang.Math.*;
```

После подобного импорта можно использовать все статические метода класса `Math` без указания имени класса. Очевидно, что оператор `import` по своему значению близок к конструкции `using` языка `C#`, но не соответствует `include` в языке `C++`, так как не подключает никакие файлы.

Можно сформулировать следующие правила построения исходных файлов Java:

- в первой строке файла может быть необязательный оператор `package`;
- в следующих строках могут быть необязательные операторы `import`;
- далее идут описания классов и интерфейсов;
- среди классов файла может быть только один открытый `public`-класс.
- имя файла должно совпадать с именем открытого класса, если такой существует.

3.4. Оборудование и материалы

Для выполнения лабораторной работы рекомендуется использовать персональный компьютер со следующими характеристиками: 64-разрядный (x64) процессор с тактовой частотой 1 ГГц и выше, оперативная память – 1 Гб и выше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система Windows 7 и выше, Java Development Kit 8 и выше; IDE Eclipse for Java SE (or Java EE) developers.

3.5. Указания по технике безопасности

Студенты должны следовать общепринятой технике безопасности для пользователей персональных компьютеров. Не следует самостоятельно производить ремонт технических средств, установку и удаление программного обеспечения. В случае обнаружения неисправностей необходимо сообщить об этом администратору компьютерного класса (обслуживающему персоналу лаборатории).

3.6. Методика и порядок выполнения работы

1. Спроектируйте структуру приложения для решения задачи в соответствии с вариантом: продумайте структуру файлов, пакетов; необходимые классы.
2. Реализуйте проект приложения в соответствии с разработанной структурой.
3. Проанализируйте структурную организацию программ на языке C#. В чем сходство и в чем различия с технологией Java?

Индивидуальное задание.

Объявите требуемые переменные, значения переменным пользователь должен присваивать в процессе выполнения программы (считываются с

консоли), выведите на экран с использованием форматной строки значения переменных и результат вычисления выражения во всех возможных форматах:

Вариант	Задание
1	Приложение осуществляет сложение обыкновенных дробей, то есть чисел $\frac{1}{3}$, $\frac{7}{12}$ и т.д. Требуется реализовать только сложение. Исходные слагаемые пользователь вводит с клавиатуры.
2	Дана исходная матрица размером $M \times N$. Вывести исходную матрицу. Вывести среднее арифметическое для каждой строки и результирующую матрицу, в которой все элементы заменены по следующим правилам: - если элемент меньше чем среднее арифметическое строки, то выводится 0; - если элемент больше чем среднее арифметическое строки, то выводится 1; - если не выполняются предыдущие правила, то элемент не заменяется.
3	Дана исходная матрица размером $M \times N$. Вывести исходную матрицу. Вывести минимальный и максимальный элементы в матрице (и их индексы) и результирующую матрицу, которая является транспонированной по отношению к исходной.
4	Реализуйте программу для выполнения операций над комплексными числами: сложение, умножение, вычисление модуля комплексного числа. Исходные комплексные числа пользователь вводит с клавиатуры.
5	Приложение осуществляет умножение обыкновенных дробей, то есть чисел $\frac{1}{3}$, $\frac{7}{12}$ и т.д. Требуется реализовать только умножение. Исходные множители пользователь вводит с клавиатуры.
6	Дана исходная матрица размером $M \times N$. Вывести исходную матрицу. Вывести максимальный элемент матрицы и результирующую матрицу, в которой все элементы, большие среднего арифметического, заменены на 0.
7	Реализуйте приложение, осуществляющее сокращение обыкновенной дроби. Дробь пользователь вводит с клавиатуры.
8	Дана исходная матрица размером $M \times N$. Вывести исходную матрицу. Вывести минимальный элемент матрицы и результирующую матрицу, в которой все

	элементы, большие среднего арифметического в данной строке, заменены на минимальный элемент матрицы.
9	Приложение осуществляет деление обыкновенных дробей, то есть чисел $\frac{1}{3}$, $\frac{7}{12}$ и т.д. Требуется реализовать только деление. Исходные дроби пользователь вводит с клавиатуры.
10	Дана исходная матрица размером $M \times N$. Вывести исходную матрицу. Вывести среднее арифметическое элементов матрицы и результирующую матрицу, в которой все элементы больше среднего арифметического матрицы, заменены на -1.
11	Дана исходная матрица размером $M \times N$. Вывести исходную матрицу. Вывести средние арифметические для каждой строки и результирующую матрицу, в которой все четные элементы заменены на среднее арифметическое в данной строке.
12	Дана исходная матрица размером $M \times N$. Вывести исходную матрицу. Вывести средние арифметические для каждого столбца и результирующую матрицу, в которой все элементы, заменяются по правилу: четные элементы заменяются на 1, нечетные – на 0.
13	Дана исходная матрица размером $M \times N$. Вывести исходную матрицу. Вывести минимальный элемент для каждой строки и результирующую матрицу, в которой все элементы, которые делятся на 3, заменены на минимальный элемент во всей матрице.
14	Дана исходная матрица размером $M \times N$. Вывести исходную матрицу. Вывести максимальный элемент для каждого столбца и результирующую матрицу, которая является транспонированной по отношению к исходной.
15	Дана исходная матрица размером $M \times N$. Вывести исходную матрицу. Вывести среднее арифметическое для каждой строки и результирующую матрицу, в которой все элементы заменены по следующим правилам: - если элемент делится на 2, то вывести 2; - если элемент делится на 3, то вывести 3; - если не выполняются предыдущие правила, то элемент не заменяется.
16	Дана исходная матрица размером $M \times N$. Вывести исходную матрицу. Вывести среднее арифметическое для каждой строки и результирующую матрицу, в которой все элементы заменены по следующим правилам: - если элемент меньше чем среднее арифметическое строки,

	то выводится 0; - если элемент больше чем среднее арифметическое строки, то выводится 1; - если не выполняются предыдущие правила, то элемент не заменяется.
17	Дана исходная матрица размером $M \times N$. Вывести исходную матрицу. Вывести минимальный и максимальный элементы в матрице (и их индексы) и результирующую матрицу, которая является транспонированной по отношению к исходной.
18	Дана исходная матрица размером $M \times N$. Вывести исходную матрицу. Вывести максимальный элемент матрицы и результирующую матрицу, в которой все элементы, большие максимального, заменены на 0.
19	Дана исходная матрица размером $M \times N$. Вывести исходную матрицу. Вывести минимальный элемент матрицы и результирующую матрицу, в которой все элементы, большие среднего арифметического в данной строке, заменены на минимальный элемент матрицы.
20	Дана исходная матрица размером $M \times N$. Вывести исходную матрицу. Вывести среднее арифметическое элементов матрицы и результирующую матрицу, в которой все элементы, меньшие среднего арифметического, заменены на 0, а большие среднего арифметического – на 1.
21	Дана исходная матрица размером $M \times N$. Вывести исходную матрицу. Вывести средние арифметические для каждой строки и результирующую матрицу, в которой все четные элементы заменены на среднее арифметическое в данной строке.
22	Дана исходная матрица размером $M \times N$. Вывести исходную матрицу. Вывести среднее арифметическое для каждого столбца и результирующую матрицу, в которой все элементы заменены по следующим правилам: - если элемент делится на 2, то вывести 2; - если элемент делится на 5, то вывести 5; - если не выполняются предыдущие правила, то вывести 0.

3.7. Содержание отчета и его форма

Отчет по лабораторной работе должен содержать:

1. Номер и название лабораторной работы; задачи лабораторной работы.
3. Ответы на контрольные вопросы.

4. Экранные формы (консольный вывод) и листинг программного кода с комментариями, показывающие порядок выполнения лабораторной работы, и результаты, полученные в ходе её выполнения.

Отчет о выполнении лабораторной работы, подписанный студентом, сдается преподавателю.

3.8. Контрольные вопросы

1. Что такое пакет?
2. Какой оператор необходимо использовать для импортирования класса?
3. Какие правила рекомендуется выполнять при именовании классов и пакетов?
4. Назовите правила, которые необходимо выполнять при определении классов и распределении их по файлам и пакетам?
5. Укажите различные варианты оператора `import`.
6. Укажите сходство механизмов структурной организации программ в технологиях Java и .NET Framework.
7. Укажите отличия механизмов структурной организации программ в технологиях Java и .NET Framework.

3.9. Список литературы

Для выполнения лабораторной работы, при подготовке к защите, а также для ответа на контрольные вопросы рекомендуется использовать следующие источники: [4], [7], [8].

ЛАБОРАТОРНАЯ РАБОТА 4. НАСЛЕДОВАНИЕ РЕАЛИЗАЦИИ

4.1. Цель и содержание

Цель лабораторной работы: изучить принципы механизма наследования реализации при разработке объектно-ориентированных приложений.

Задачи лабораторной работы:

- изучить синтаксические принципы реализации наследования;
- изучить принципы реализации наследования при решении практических задач.

4.2. Формируемые компетенции

ОПК-4; ОПК-5; ОПК-6.

4.3. Теоретическая часть

4.3.1 Механизм наследования реализации.

Наследование – фундаментальное понятие объектно-ориентированного программирования. Идея, лежащая в основе наследования, заключается в том, что новые классы можно создавать из уже существующих. При наследовании реализации методы, и поля существующего класса используются повторно (наследуются) вновь создаваемым классом, причем для адаптации нового класса к новым ситуациям в него добавляют дополнительные поля и методы. Этот прием играет в языке Java важную роль.

4.3.2 Проектирование приложения с применением механизма наследования

Для демонстрации принципов наследования рассмотрим пример. В рамках проекта необходимо спроектировать классы для сущностей предметной

области: геоинформационная система (ГИС). Выделим следующие сущности: «Подвижный объект», «Неподвижный объект», «Человек», «Транспорт», «Здание», «Дорога». Очевидно, что для описания реально функционирующей ГИС, предметную область нужно детализировать. При проектировании данного учебного примера будем придерживаться определенных допущений. Превосходство объектно-ориентированного подхода в том, что на данном этапе проектирования разработчик может описывать сущности (и их свойства) предметной области в терминах естественного языка:

- для сущности «Подвижный объект» необходимо реализовать хранение данных о перемещении объекта;
- сущность «Неподвижный объект» содержит информацию о размерах, геометрии, координатах объекта;
- сущность «Человек» инкапсулирует информацию: фамилия, имя, отчество, выполняемое задание;
- сущность «Транспорт» инкапсулирует информацию: номер, тип, модель;
- сущность «Здание» предполагает, что все здания имеют форму прямоугольника;
- дороги в ГИС представляются отрезками; сущность «Дорога» инкапсулирует начальную и конечную координату представляемого пути.

Для ГИС характерно, что все представляемые на карте объекты должны иметь определенные координаты – то есть свойство наличия координат является общим для всех сущностей.

Следует также обратить внимание, что сущности «Дорога» и «Здание» обладает определенной семантической связью с сущностью «Неподвижный объект»: две случаи являются частными случаями третьей. Подобные связи можно установить и для прочих выявленных типов объектов проектируемой ГИС.

От словесного описания сущностей следует перейти к определению классов Java. Это удобно реализовать в виде графического представления – диаграммы классов. На рис. 4.1 представлена диаграмма классов.

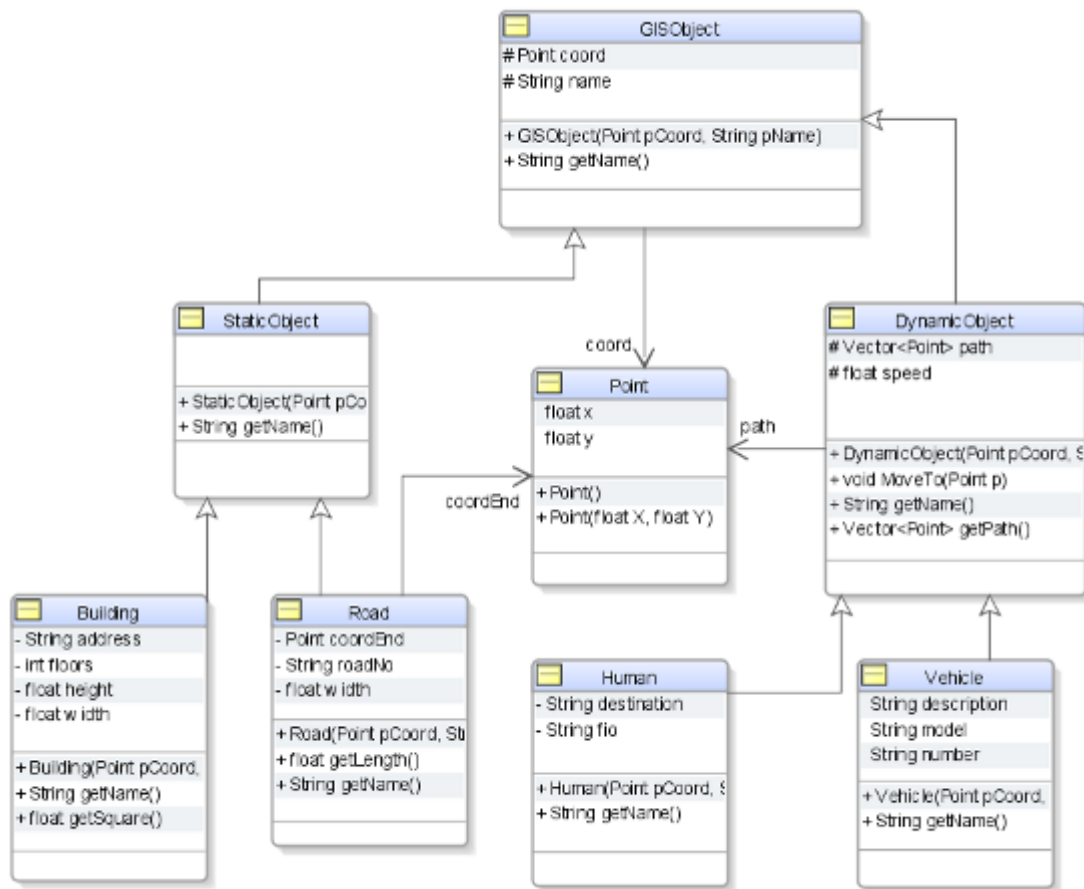


Рисунок 4.1 – Диаграмма классов приложения ГИС.

Кроме перечисленных сущностей на диаграмме отображен класс Point, созданный для представления двумерной координаты в ГИС.

На диаграмме введен «главный» класс GISObject (любой объект, присутствующий в ГИС). Этот класс инкапсулирует информацию, которая необходима для любого объекта, присутствующего в нашей системе: координата и наименование. Эти поля реализованы как protected. В классе присутствует также открытый метод getName(), который возвращает имя объекта.

Поля класса GISObject понадобятся в каждом классе, но на диаграмме эти поля не повторяются – вместо этого между классами установлены связи. Графически связь представлена как $\longrightarrow$. Класс, к которому направлена стрелка,

является базовым (класс-родитель, суперкласс); от которого – производным (класс-наследник, подкласс). Подобная связь указывает, что класс `DynamicObject` наследует реализацию `GISObject`. Под реализацией класса понимаются поля данных и сигнатуры и тела методов, то есть класс-наследник обладает такой же реализацией (функциональностью). Но, одновременно с реализацией, класс-наследник расширяет базовый класс, то есть добавляет что-то свое, специфическое – добавляет поля и методы, отсутствующие в базовом классе. Теперь трансформируем диаграмму в код Java. Рассмотрим трансформацию на примере классов `DynamicObject` и `GISObject`. Диграмма была создана в IDE `JDeveloper 12c`, исходный код был сгенерирован автоматически.

```

3  public class GISObject {
4      protected Point coord;
5      protected String name;
6
7      public GISObject(Point pCoord, String pName) {
8          coord = pCoord;
9          name = pName;
10     }
11
12     public String getName() {
13         return this.name;
14     }
15 }

```

Рисунок 4.2 – Класс `GISObject`.

```

4 public class DynamicObject extends GISObject {
5     protected Vector<Point> path;
6     protected float speed;
7
8     public DynamicObject(Point pCoord, String pName) {
9         super(pCoord, pName);
10    }
11
12    @Override
13    public String getName() {
14        return "Мобильный объект: "
15            + this.name;
16    }
17
18    public Vector<Point> getPath() {
19        return path;
20    }
21
22    public void MoveTo(Point p) {
23        path.add(p);
24    }
25 }

```

Рисунок 4.3 – Класс DynamicObject.

В технологии Java для механизма наследования реализации наиболее подходит термин «расширяет». Синтаксически наследование определяется с использованием зарезервированного слова `extends` (рис. 4.3). В реализации класса `DynamicObject` присутствует метод `getName()`, сигнатура которого совпадает с методом класса `GISObject`. Это демонстрация механизма переопределения методов. Метод один для обоих классов, но реализация его особенная для каждого класса.

Существуют поля (например, поле `name` класса `GISObject`), определение которых отсутствуют в производных классах, но они видимы в классах-наследниках (на рис. 4.3 в строке 11 происходит обращение к полю базового класса). Это указывает на тот факт, что реализация базового класса была унаследована в производном классе.

На рис. 4.1 показан не только механизм наследования на примере двух классов (родитель – потомок), а спроектированы классы ГИС в виде иерархии. Совокупность всех классов, производных от одного суперкласса, называется

иерархией наследования (например, рис. 4.1). Путь от конкретного класса к его потомкам в иерархии называется цепочкой наследования.

Базовый класс иерархии – GISObject. Одни поля этого класса (name, coord) используются всеми классами иерархии; другие (метод getName()) переопределяются, если это необходимо.

Дополнительно, необходимо обратить внимание на спецификаторы доступа полей name и coord класса GISObject. Они маркированы protected. Это гарантирует, что поля будут закрыты для доступа к ним через объектную переменную типа GISObject, но они будут видимы в классах-потомках (в контекстах их методов).

4.3.3 Зарезервированные слова

При реализации наследования программисту доступны зарезервированные слова, которые указывают, какой именно метод необходимо вызывать: метод базового класса или метод производного. Если при вызове метода используется вызов метода с использованием синтаксиса:

```
myMeth();
```

то этот вызов, фактически, заменяется на:

```
this.myMeth();
```

Но если метод myMeth() переопределен и необходимо явно указать, что нужно вызвать метод базового класса, то программист должен указать это явно:

```
super.myMeth();
```

Зарезервированное слово super отсылает компилятор к полям базового класса.

4.3.4 Реализация полиморфизма

Экономия при наборе кода, предотвращение дублирования блоков кода – не единственные преимущества наследования. Наследование, как механизм, открывает путь к полиморфному поведению объектов. Полиморфизм, наряду с наследованием, является фундаментальным принципом ООП.

Существует простое правило, позволяющее определить, стоит ли в конкретной ситуации применять наследование или нет. Если между объектами существует отношение «является» («is-a»), то каждый объект подкласса является объектом суперкласса.

Например, каждый объект типа Vehicle («Транспорт») является объектом типа DynamicObject («Подвижный объект»). Следовательно, имеет смысл сделать класс Vehicle подклассом класса DynamicObject. Естественно, обратное утверждение неверно – не каждый подвижный объект в ГИС является автотранспортом.

Другой способ – принцип подстановки. Этот принцип гласит, что объект подкласса можно использовать вместо любого объекта суперкласса.

Например, объект подкласса можно присвоить переменной суперкласса:

```

3 public class Starter {
4     public static void main(String[] args) {
5         DynamicObject e = new DynamicObject(new Point(), "Объект 0");
6         e = new Human(new Point(), "Новак Е.И.", "?");
7         e = new Vehicle(new Point(), "Hyundai Sol",
8                         "y445KE", "?");
9         System.out.println(e.getName());
10    }
11 }

```

Рисунок 4.4 – Полиморфный объект.

В языке Java объектные переменные являются полиморфными. Переменная типа DynamicObject может ссылаться как на объект DynamicObject, так и на объект любого подкласса класса DynamicObject (например, Vehicle, Human).

Рассмотрим проявление полиморфизма на примере. Для манипулирования объектами в ГИС создается единый массив объектов.

На рис. 4.3 представлено определение класса DynamicObject, на рис. 4.5 и 4.6 представлены определения классов Vehicle и Human соответственно.

```

1 public class Vehicle extends DynamicObject {
2     String model;
3     String number;
4     String description;
5
6     public Vehicle(Point pCoord, String pModel,
7         String pNumber, String pDescr) {
8         super(pCoord, pNumber);
9         model = pModel;
10        description = pDescr;
11    }
12
13    @Override
14    public String getName() {
15        return "Транспорт" +
16            "\n Модель: " + model +
17            "\n Номер: " + number +
18            "\n Информация: " + description;
19    }
20 }

```

Рисунок 4.5 – Определение класс Vehicle.

```

1 public class Human extends DynamicObject {
2     private String fio;
3     private String destination;
4
5     public Human(Point pCoord, String pFIO, String pDest) {
6         super(pCoord, pFIO);
7         fio = pFIO;
8         destination = pDest;
9     }
10
11    @Override
12    public String getName() {
13        return "ФИО: " + fio +
14            "\n Задание: " + destination;
15    }
16 }

```

Рисунок 4.6 – Определение класс Human.

Модуль, который будет использовать данные классы должен работать со всеми подвижными объектами единообразными методами, то есть не нужно писать два разных алгоритма, чтобы вывести описание сущностей «Человек» и «Транспорт». Поэтому необходимо хранить объекты классов Human и Vehicle в одном списке (массиве), работая с каждым элементом единообразно, вызывая методы базового класса DynamicObject (рис. 4.7).

```

6 public class Starter {
7     public static void main(String[] args) {
8
9         Vector<DynamicObject> allDynamics =
10             new Vector<DynamicObject>();
11         allDynamics.add(new Vehicle(
12             new Point(), "Hyundai Sol", "У445KE", "?"));
13         allDynamics.add(new Human(
14             new Point(), "Новак Е.И.", "в центр"));
15         allDynamics.add(new Vehicle(
16             new Point(), "Kia Rio", "У111KE", "новый"));
17         allDynamics.add(new Human(
18             new Point(), "Николаев Е.И.", "на юго-запад"));
19         allDynamics.add(new Vehicle(
20             new Point(), "Лада", "К433ТЕ", "белый"));
21
22         // Единообразная работа со всеми элементами
23         for(int i = 0; i < allDynamics.size(); i++)
24             System.out.println(allDynamics.get(i).getName());
25     }
26 }

```

Рисунок 4.7 – Работа со списком DynamicObject.

На рис. 4.8 показан фрагмент вывода программы, демонстрирующий, что метод getName(), вызываемый на каждом объекте коллекции, отображает различную информацию. Данный метод не только выводит разные строки – он обращается к различным полям внутри объектов-элементов коллекции.

```

D:\Server\Oracle\Middleware\Orac
Транспорт
  Модель: Hyundai Sol
  Номер: null
  Информация: ?
ФИО: Новак Е.И.
  Задание: в центр
Транспорт
  Модель: Kia Rio
  Номер: null
  Информация: новый
ФИО: Николаев Е.И.
  Задание: на юго-запад
Транспорт
  Модель: Лада
  Номер: null
  Информация: белый

```

Рисунок 4.8 – Вывод приложения ГИС.

4.3.5 Предотвращение наследования. Абстрактные классы.

Иногда наследование является нежелательным. Классы, которые нельзя расширить, называются финальными. Для того чтобы отметить этот факт, в определении класса используется ключевое слово `final`. Например:

```

1 public final class Human extends DynamicObject {
2     private String fio;
3     private String destination;
4 }

```

При таком определении класса `Human`. Теперь класс `Human` не может являться суперклассом ни для какого другого класса; на данном классе прерывается цепочка наследования.

Спецификатором `final` может быть помечен отдельный метод класса. В этом случае запрещено переопределять этот метод в подклассах.

Например, класс `String` библиотеки Java определен как `final`. Создавать подклассы этого класса запрещено. Поэтому если у вас есть переменная типа `String`, то вы можете быть уверены, что она ссылается именно на строку, и ни на какой другой объект.

При разработке приложений программист определяет собственные классы, не являющиеся подклассами какого-либо иного класса. Например класс `GISObject` является корнем иерархии (рис. 4.1). В Java такие классы неявным образом расширяют класс `Object`.

Если суперкласс явно не указан, им считается класс `Object`. Поскольку в языке Java каждый класс расширяет `Object`, программисту необходимо знать, какими возможностями обладает сам класс `Object`.

4.3.6 Динамическое связывание

Важно понимать, что происходит при вызове метода, принадлежащего некоторому объекту. При этом выполняются следующие действия:

1. Компилятор проверяет объявленный тип объекта и имя метода. Предположим, происходит вызов метода `ob.MyMeth(args)`, причем неявный параметр `ob` объявлен как экземпляр класса `MyClass`. Заметим, что может

существовать несколько методов с именем `MyMeth`, имеющих разные типы параметров (например, `MyMeth(int)` и метод `MyMeth(String)`). Компилятор нумерует все методы с именем `MyMeth` в классе `MyClass` и все общедоступные методы с именем `MyMeth` в суперклассах класса `MyClass`. Теперь компилятор знает всех возможных «кандидатов» при вызове метода.

2. Затем компилятор определяет типы параметров, указанных при вызове метода. Если среди всех методов с именем `MyMeth` есть только один метод, типы параметров которого совпадают с указанными, происходит его вызов. Этот процесс называется разрешением перегрузки. Например, при вызове `ob.MyMeth("Привет")` компилятор выберет метод `MyMeth(String)`, а не метод `MyMeth(int)`. Если компилятор не находит ни одного метода с подходящим набором параметров или в результате преобразования типов возникает несколько методов, соответствующих данному вызову, выдается сообщение об ошибке. Теперь компилятор знает имя и типы параметров метода, подлежащего вызову.

3. Если метод является приватным (`private`), статическим (`static`), финальным (`final`) или конструктором, компилятор точно знает, как его вызвать. Такой процесс называется статическим связыванием. В противном случае метод, подлежащий вызову, определяется по фактическому типу неявного параметра и во время выполнения программы используется динамическое связывание. В нашем примере компилятор сгенерировал бы вызов метода `MyMeth(String)` с помощью динамического связывания.

4. Если при выполнении программы для вызова метода используется динамическое связывание, виртуальная машина должна вызвать версию метода, соответствующую фактическому типу объекта, на который ссылается переменная `ob`. Предположим, что объект имеет фактический тип `SuperClass`, являющийся суперклассом класса `MyClass`. Если в классе `SuperClass` определен метод `MyMeth(String)`, то вызывается именно он. Если нет, то поиск метода `MyMeth(String)`, подлежащего вызову, выполняется в суперклассе и т.д.

На поиск вызываемого метода уходит слишком много времени, поэтому виртуальная машина заранее создает для каждого класса таблицу методов, в которой перечисляются сигнатуры всех методов и фактические методы, подлежащие вызову. При вызове метода виртуальная машина просто просматривает таблицу методов. В нашем примере виртуальная машина проверяет таблицу методов класса `SuperClass` и обнаруживает метод `MyMeth(String)`, подлежащий вызову. Такими методами могут быть `SuperClass.MyMeth(String)` или `SomeClass.MyMeth(String)`, если `SomeClass` – некоторый суперкласс класса `SuperClass`.

Описанный поэтапный механизм называется динамическим связыванием. Динамическое связывание обладает одной важной особенностью: оно позволяет модифицировать программы без перекомпиляции их кодов. Это делает программы динамически расширяемыми.

4.3.7 Класс `Object`. Метод `equals()`.

Класс `Object` является предком всех классов – каждый класс в языке Java расширяет класс `Object`. Указывать явно расширение класса `Object` с использованием зарезервированного слова `extends` не нужно. Если суперкласс явно не указан, им считается класс `Object`. В языке Java каждый класс расширяет `Object`, поэтому каждый класс обладает некоторой базовой функциональностью, которая предоставлена суперклассом `Object`. Рассмотрим основные возможности:

1. Переменную типа `Object` можно использовать в качестве ссылки на объект любого типа:

```
Object ob = new Human(new Point(100, 100), "Николаев Е.И.", "Юг");
```

При таком использовании объектов суперкласса, переменная `ob` будет использована только для хранения значений произвольного типа (вызывать методы класса `Human` нельзя). Для получения доступа к полям класса `Human` необходимо выполнить процедуру преобразования типа:

```
Human obH = (Human)ob;  
obH.MoveTo(new Point(0.0f, 0.0f));
```

В Java не являются объектами, производными от Object, переменные простых типов. Наследование от Object распространяется на массивы (как на переменные-массивы, так и на элементы массивов).

2. Класс Object предоставляет метод equals(). Данный метод проверяет, эквивалентны ли два объекта. Поскольку метод equals() реализован в классе Object, он определяет лишь, ссылаются ли переменные на один и тот же объект. В качестве проверки по умолчанию эти действия вполне оправданы: всякий объект эквивалентен самому себе. Для некоторых классов большего и не требуется. Однако в ряде случаев эквивалентными должны считаться объекты одного типа, имеющие одинаковые состояния. Например, на рис. 4.9 представлен класс Order (Заказ). В классе присутствуют поля, отражающие свойства: номер заказа, дата, товар, количество, цена, покупатель; в структуре класса предусмотрены методы: конструктор, получение общей стоимости заказа, представление заказа.

```

5 public class Order {
6     private int no; // номер заказа
7     private Date dt; // дата заказа
8     private String stuff; // товар
9     private float price; // цена
10    private int count; // количество
11    private String customer; // покупатель
12
13    public Order(int pNo, String pStuff, float pPrice,
14                int pCount, String pCustomer){
15        no = pNo; stuff = pStuff; price = pPrice;
16        count = pCount; customer = pCustomer;
17        dt = new Date();
18    }
19
20    public float getTotalPrice(){
21        return price * count;
22    }
23
24    public String getOrderName(){
25        return "Заказ №" + String.valueOf(no) +
26               " от " + String.valueOf(dt) +
27               "(" + getTotalPrice() + ")";
28    }
29 }

```

Рисунок 4.9 – Класс Order.

На рис. 4.10 продемонстрировано использование объектов данного класса.

```

56 public static void main(String[] args) throws InterruptedException{
6     // Создание первого заказа
7     Order o1 = new Order(1, "Переходник HDMI-DVI",
8         780.00f, 12, "Николаев Е.И.");
9     // Создание заказа идентичного первому
10    Order o1_copy = new Order(1, "Переходник HDMI-DVI",
11        780.00f, 12, "Николаев Е.И.");
12    // Задержка
13    Thread.sleep(5000);
14    // Создание второго заказа
15    Order o2 = new Order(2, "Монитор Samsung L110",
16        7600.00f, 2, "Ирисова А.Ю.");
17    // Вывод информации
18    System.out.println(o1.getOrderName());
19    System.out.println(o1_copy.getOrderName());
20    System.out.println(o2.getOrderName());
21
22    System.out.println(o1.equals(o1_copy));
23    System.out.println(o1.equals(o2));
24 }

```

Рисунок 4.10 – Использование возможностей объектов типа Order.

В результате выполнения данного кода получим результат, представленный на рис. 4.11.

```

Заказ №1 от Thu Feb 12 15:24:17 MSK 2015(9360.0)
Заказ №1 от Thu Feb 12 15:24:17 MSK 2015(9360.0)
Заказ №2 от Thu Feb 12 15:24:22 MSK 2015(15200.0)
false
false

```

Рисунок 4.11 – Результат сравнения объектов Order.

Каждый вызов метода equals() вернул false, так как анализировались ссылки, хранящиеся в переменных o1, o1_copy, o2. Все эти переменные ссылаются на различные области памяти.

Необходимо доработать класс Order для реализации корректного сравнения объектов данного типа. Добавим переопределение метода equals():

```

30 @Override
31 public boolean equals(Object anotherOb){
32     if(!(anotherOb instanceof Order))
33         return false;
34     Order ob = (Order)anotherOb;
35     if((this.no == ob.no) && this.dt.equals(ob.dt) &&
36         this.stuff.equals(ob.stuff) && (this.price == ob.price) &&
37         (this.count == ob.count) && this.customer.equals(ob.customer))
38         return true;
39
40     return false;
41 }

```

Рисунок 4.12 – Переопределение метода equals() в подклассе Order суперкласса

Object.

Вывод программы представлен на рис. 4.13.

```
Заказ №1 от Thu Feb 12 15:43:43 MSK 2015(9360.0)
Заказ №1 от Thu Feb 12 15:43:43 MSK 2015(9360.0)
Заказ №2 от Thu Feb 12 15:43:48 MSK 2015(15200.0)
true
false
```

Рисунок 4.13 – Результат работы программы с использованием
переопределенного метода equals().

Таким образом, программист может самостоятельно определять бизнес-логику приложения и программировать правила сравнения объектов.

Спецификация языка Java требует, чтобы метод equals() обладал следующими характеристиками:

1. Рефлексивность. Для любой ненулевой ссылки ob вызов ob.equals(ob) должен возвращать значение true.
2. Симметричность. Для любых ссылок ob1 и ob2 вызов ob1.equals(ob2) должен возвращать значение true тогда и только тогда, когда ob2.equals(ob1) возвращает true.
3. Транзитивность. Для любых ссылок ob1, ob2 и ob3, если вызовы ob1.equals (ob2) и ob2.equals(ob3) возвращают значение true, вызов ob1.equals(ob3) возвращает значение true.
4. Согласованность. Если объекты, на которые ссылались переменные ob1 и ob2 не изменяются, то повторный вызов ob1.equals (ob2) должен возвращать то же значение.
5. Для любой null-ссылки ob1 вызов ob1.equals(null) должен возвращать значение false.

При реализации метода сравнения программисту важно придерживаться следующих стадий сравнения объектов в методе equals():

1. Предположим, что явный параметр называется anotherOb. После приведения типа – ob (рис. 4.12).

2. Проверить, идентичны ли ссылки `this` и `anotherOb`.

```
if(this == anotherOb)
    return true;
```

Это выражение используется для оптимизации проверки. Гораздо быстрее проверить идентичность ссылок, чем сравнивать поля объектов.

3. Выяснить, является ли ссылка `anotherOb` нулевой (`null`). Если да, вернуть значение `false`. Эту проверку нужно делать обязательно.

```
if(anotherOb == null)
    return false;
```

4. Сравнить классы `this` и `anotherOb`. Если семантика проверки может измениться в подклассе, использовать метод `getClass()`.

```
if(getClass() != anotherOb.getClass())
    return false;
```

Если принцип проверки остается справедливым для всех подклассов, использовать операцию `instanceof`.

```
if(!(anotherOb instanceof Order))
    return false;
```

5. Преобразовать объект `anotherOb` в переменную требуемого класса.

```
Order ob = (Order)anotherOb;
```

6. Далее следует реализовать логику сравнения с использованием требований предметной области. Для полей простых типов используется операция `==`, для объектных полей – метод `equals()`.

```
if((this.no == ob.no) && this.dt.equals(ob.dt) &&
    this.stuff.equals(ob.stuff) && (this.price == ob.price) &&
    (this.count == ob.count) && this.customer.equals(ob.customer))
    return true;
```

4.3.8 Метод `toString()` класса `Object`.

Еще одним важным методом класса `Object` является `toString()`, возвращающий значение объекта в виде строки. Например, для объекта класса `Order`, определенного на рис. 4.9, вызов метода `toString()` возвратит следующую строку: «`lesson4.Order@70dea4e`». То есть, если метод суперкласса `Object` не переопределен в подклассе, то он возвращает строку формата:

<имя_класса>@<адрес_объекта>

Создадим переопределение метода toString() для класса Order:

```
public String getOrderName(){
    return "Заказ №" + String.valueOf(no)
        + " от " + String.valueOf(dt) +
        "(" + getTotalPrice() + ")";
}

public boolean equals(Object anotherOb){

@Override
public String toString(){
    return this.getOrderName();
}
```

Рисунок 4.14 – Переопределение метода toString() класса Order.

Таким образом, метод toString() суперкласса, переопределенный в Order, выполняет роль пользовательского метода getOrderName(). Метод getOrderName() может быть удален.

При разработке программ рекомендуется всегда переопределять метод toString() для пользовательских классов: это не только является хорошим стилем программирования, но и позволяет персонифицировать поведение пользовательских классов.

4.4. Оборудование и материалы

Для выполнения лабораторной работы рекомендуется использовать персональный компьютер со следующими характеристиками: 64-разрядный (x64) процессор с тактовой частотой 1 ГГц и выше, оперативная память – 1 Гб и выше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система Windows 7 и выше, Java Development Kit 8 и выше; IDE Eclipse for Java SE (or Java EE) developers.

4.5. Указания по технике безопасности

Студенты должны следовать общепринятой технике безопасности для пользователей персональных компьютеров. Не следует самостоятельно производить ремонт технических средств, установку и удаление программного обеспечения. В случае обнаружения неисправностей необходимо сообщить об этом администратору компьютерного класса (обслуживающему персоналу лаборатории).

4.6. Методика и порядок выполнения работы

1. Спроектируйте иерархию классов приложения в соответствии с предметной областью, описанной в индивидуальном задании. Приведенную иерархию классов в задании необходимо дополнить и расширить тремя дополнительными классами.
2. Реализуйте все спроектированные классы в приложении.
3. Запустите приложение, протестируйте его и исправьте ошибки.
4. Продемонстрируйте полиморфное поведение объектов.

Индивидуальное задание.

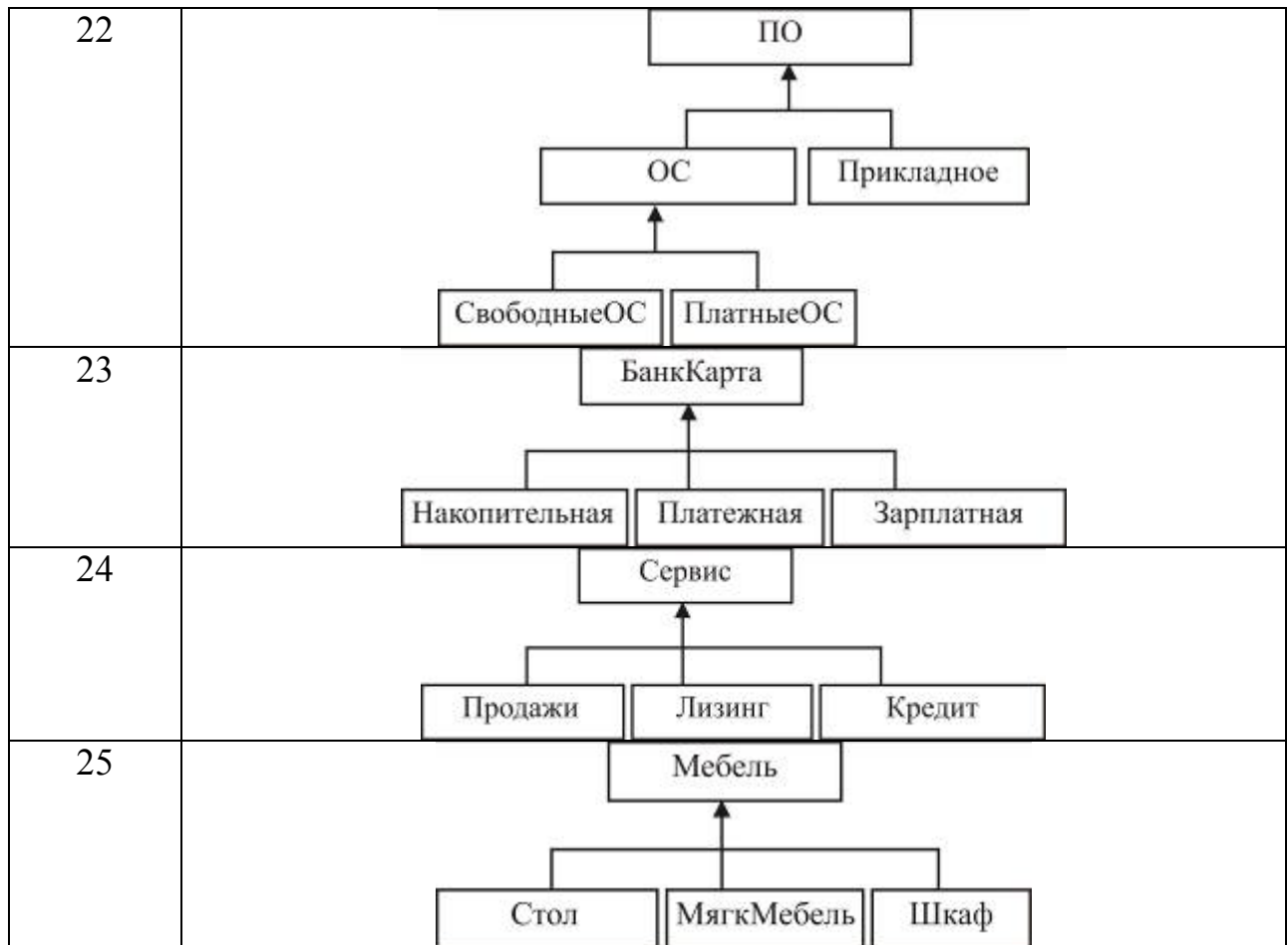
Перед выполнением задания требуется самостоятельно определить закономерность изменения членов последовательности, чтобы применить цикл, условный оператор или, если потребуется, оператор выбора.

Вариант	Предметная область (иерархия сущностей)
1	 <pre> graph BT ГрузовойАвто --> Автомобиль ЛегковойАвто --> Автомобиль Автомобиль --> ТехСредство Вертолёт --> ТехСредство </pre>
2	 <pre> graph BT Сервер --> КомпТехника Ноутбук --> КомпТехника ПК --> КомпТехника </pre>

3	<pre> graph BT Геометрия --> Четырехугольник Геометрия --> Треугольник Четырехугольник --> Квадрат Четырехугольник --> Ромб </pre>
4	<pre> graph BT Издание --> Книга Издание --> Журнал Издание --> ЭлРесурс </pre>
5	<pre> graph BT Мультимедиа --> Изображение Мультимедиа --> Видео Изображение --> Растровое Изображение --> Векторное </pre>
6	<pre> graph BT Вооружение --> Стрелковое Вооружение --> Танк Вооружение --> Сомолет </pre>
7	<pre> graph BT Машины --> Авиасредство Машины --> Грузовик Авиасредство --> Вертолет Авиасредство --> Истребитель </pre>
8	<pre> graph BT Растение --> Трава Растение --> Дерево Растение --> Кустарник </pre>

9	<pre> graph BT ПО[ПО] --> ОС[ОС] ПО --> Прикладное[Прикладное] ОС --> СвободныеОС[СвободныеОС] ОС --> ПлатныеОС[ПлатныеОС] </pre>
10	<pre> graph BT ЭВМ[ЭВМ] --> Мобильный[Мобильный] ЭВМ --> Стационарный[Стационарный] Мобильный --> Планшет[Планшет] Мобильный --> Ноутбук[Ноутбук] </pre>
11	<pre> graph BT Электроника[Электроника] --> СетТелефон[СетТелефон] Электроника --> Фотоаппарат[Фотоаппарат] Электроника --> Планшет[Планшет] </pre>
12	<pre> graph BT ТехСредство[ТехСредство] --> Автомобиль[Автомобиль] ТехСредство --> Вертолёт[Вертолёт] Автомобиль --> ГрузовойАвто[ГрузовойАвто] Автомобиль --> ЛегковойАвто[ЛегковойАвто] </pre>
13	<pre> graph BT БытТехника[БытТехника] --> Телевизор[Телевизор] БытТехника --> Утюг[Утюг] БытТехника --> Микроволновка[Микроволновка] </pre>
14	<pre> graph BT ЭВМ[ЭВМ] --> Мобильный[Мобильный] ЭВМ --> Стационарный[Стационарный] Мобильный --> Планшет[Планшет] Мобильный --> Ноутбук[Ноутбук] </pre>

15	<pre> graph BT Ручка --> КанцТовары Органайзер --> КанцТовары Скоросшиватель --> КанцТовары </pre>
16	<pre> graph BT Вертолет --> Авиасредство Истребитель --> Авиасредство Авиасредство --> Грузовик Грузовик --> Машины </pre>
17	<pre> graph BT Квартира --> Недвижимость Дом --> Недвижимость Гараж --> Недвижимость </pre>
18	<pre> graph BT Трактор --> Тяжелое Комбайн --> Тяжелое Тяжелое --> Среднее Среднее --> Машиностроение </pre>
19	<pre> graph BT Фотон --> ЭлемЧастица Электрон --> ЭлемЧастица Протон --> ЭлемЧастица </pre>
20	<pre> graph BT ПрямоПараллелепипед --> Многоугольник Треугольник --> Многоугольник Многоугольник --> Эллипс Эллипс --> Геометрия </pre>
21	<pre> graph BT Программист --> Специалист Врач --> Специалист Летчик --> Специалист </pre>



4.7. Содержание отчета и его форма

Отчет по лабораторной работе должен содержать:

1. Номер и название лабораторной работы; задачи лабораторной работы.
3. Ответы на контрольные вопросы.
4. Диаграмма классов, экранные формы (консольный вывод) и листинг программного кода с комментариями, показывающие порядок выполнения лабораторной работы, и результаты, полученные в ходе её выполнения.

Отчет о выполнении лабораторной работы, подписанный студентом, сдается преподавателю.

4.8. Контрольные вопросы

1. Что такое наследование реализации?

2. Опишите графическое представление наследования на диаграммах классов. Опишите синтаксические конструкции для реализации наследования в приложениях Java.

3. Опишите сходства и различия в механизмах наследования в Java и C#.

4. Что такое полиморфизм? Опишите механизм полиморфного поведения объектов Java.

5. Поясните смысл терминов: «суперкласс», «подкласс», «наследование реализации», «класс-наследник», «иерархическая цепочка», «динамическое связывание».

6. Какие возможности предоставляет класс `Object` библиотеки Java. Опишите назначение его основных методов. Существует ли класс с подобным функционалом в C#?

7. На рис. 4.12 метод `equals()` класса `Order` помечен как `@Override`. Поясните, какая цель подобного аннотирования?

4.9. Список литературы

Для выполнения лабораторной работы, при подготовке к защите, а также для ответа на контрольные вопросы рекомендуется использовать следующие источники: [2-4].

ЛАБОРАТОРНАЯ РАБОТА 5. ИНТЕРФЕЙСЫ

5.1. Цель и содержание

Цель лабораторной работы: изучить принципы проектирования приложений с использованием интерфейсных типов.

Задачи лабораторной работы:

- научиться объявлять и использовать интерфейсы;
- научиться проектировать приложения с использованием интерфейсов;
- научиться реализовывать полиморфное поведение в программах с использованием интерфейсов.

5.2. Формируемые компетенции

ОПК-4; ОПК-5; ОПК-6.

5.3. Теоретическая часть

Объектно-ориентированное программирование как теория, а вместе с ней и языки программирования, непрерывно развиваются и позволяют строить все более сложные программные архитектуры. Усложнение программных комплексов, необходимость взаимодействия со сторонними системами, повсеместное распространение коллективной разработки – все эти факторы способствуют появлению новых технологий в области разработки. Одним из нововведений в объектно-ориентированных языках стали типы-интерфейсы (или просто интерфейсы).

Интерфейсы – это способ указывать, что должны делать классы, не описывая, как они это должны делать. Классы могут реализовывать один или несколько интерфейсов. Если возникает необходимость использовать интерфейс, применяются объекты этих классов.

Интерфейс в языке Java не является классом. Он представляет собой множество требований, предъявляемых к классу, который должен соответствовать интерфейсу. Интерфейс можно понимать как контракт (описание того, что необходимо сделать). Контракт, сам по себе, ничего не делает, не реализует никаких методов, не хранит в себе поля объектов. Реализацией контракта занимается класс. Если класс реализует интерфейс, то он должен реализовать все методы интерфейса. Один интерфейс могут реализовывать несколько различных классов, при этом каждый из них делает это по-своему.

Рассмотрим пример. На рис. 4.1 представлена иерархия классов. Допустим, что проект ГИС развивался и возникла необходимость рисовать все эти объекты на карте. Это общий функционал для всех объектов предметной области, то есть метод `Draw()`, который рисует объекты на карте, должен быть членом класса `GISObject`, а каждый подкласс в иерархии должен переопределить этот метод – реализовать полиморфное поведение. В данной цепочке рассуждений есть недостатки:

1. При рисовании объектов понадобятся новые данные: цвет, толщина линий, геометрические примитивы и т.п. Все эти данные нужно будет включать в `GISObject`.

2. Не все объекты требуют рисования. Например, объекты классов `DynamicObject` и `StaticObject` не представляют интереса для пользователя ГИС, они только инкапсулируют общий функционал для своих подклассов. Значит, эти объекты не должны рисоваться на карте. Что делать с методом `Draw()`? Оставить пустым? Возникает альтернативная классификация типов: не «подвижные – статические», а «рисуемые – невидимые».

3. Рисование на карте – это только одна область, которая может возникнуть в процессе разработки. В системе ГИС может потребоваться: добавить к выборочным типам контактную информацию; реализовать механизм сохранения; реализовать сетевое взаимодействие; добавить механизм интерактивного взаимодействия с пользователем и т.п. Новый функционал

повлечет перестроение иерархии классов и(или) создаст перегруженность суперкласса иерархии.

Для разрешения проблемы предназначены интерфейсы. Объявление интерфейсного типа показано на рис. 5.1

```
3 public interface IDrawable {  
4     public void Draw();  
5     public String Note();  
6 }
```

Рисунок 5.1 – Пример объявления интерфейса.

Все методы интерфейса автоматически считаются общедоступными, поэтому, объявляя метод в интерфейсе, указывать модификатор `public` не обязательно. Этот конкретный интерфейс имеет два метода (естественно, программист может определить любое количество). Важен тот факт, что интерфейсы не могут реализовываться в виде объектов. В интерфейсах нет ни полей, ни тела методов; тела методов содержатся в классах, реализующих соответствующие интерфейсы. Таким образом, интерфейс можно воспринимать как абстрактный класс, лишенный каких-либо полей экземпляра. Однако между понятиями «абстрактный класс» и «интерфейс» есть существенные различия.

Теперь предположим, что объекты класса `Human` должны рисоваться на карте, то есть данный класс должен реализовывать интерфейс `IDrawable`.

Для того чтобы класс реализовал интерфейс, нужно выполнить два действия:

1. Объявить, что класс реализует интерфейс.
2. Определить в классе все методы, указанные в интерфейсе.

Реализация классом интерфейса показана на рис. 5.2 с использованием диаграммы классов.

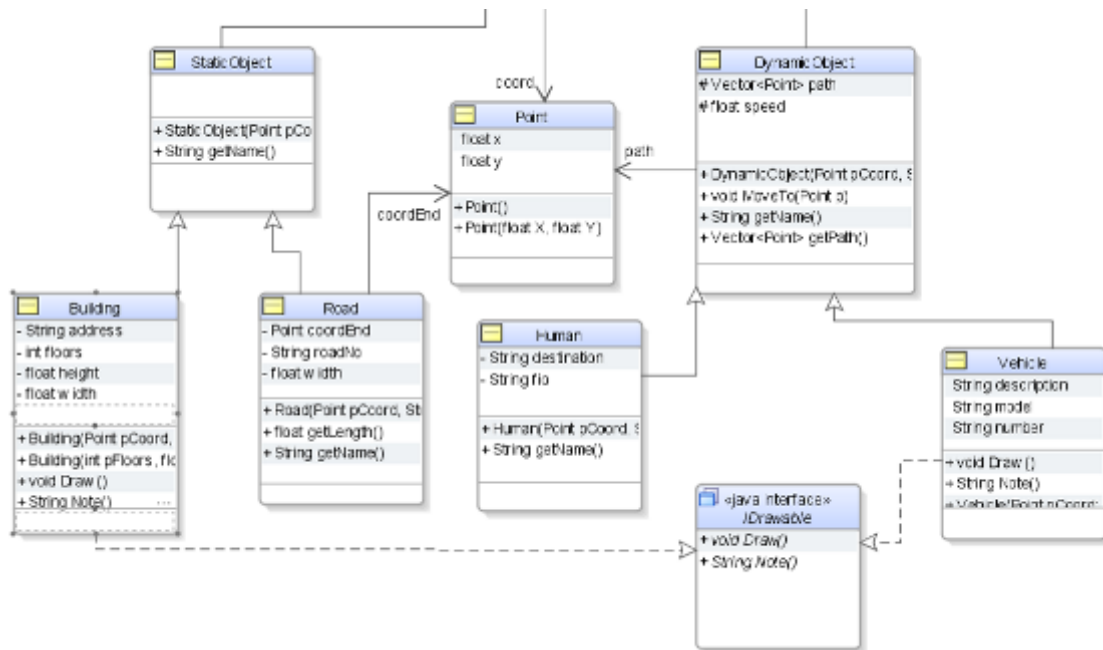


Рисунок 5.2 – Реализация интерфейса IDrawable на диаграмме.

В данном фрагменте интерфейс IDrawable реализуют классы Vehicle и Building. Это классы находятся в различных иерархических цепочках, но именно для них добавляется функционал рисования.

Синтаксически, реализация интерфейса оформляется так: указывается зарезервированное слово `implements` за которым следуют имена реализуемых интерфейсов. Необходимо также добавить методы, определенные в интерфейсе, в класс Vehicle (рис. 5.3).

```

public class Vehicle extends DynamicObject implements IDrawable {
    4    String model;
    5    String number;
    6    String description;
    7
    8    public void Draw() {
    9        String str =
    10            "    ----- \n" +
    11            "  _/          \\ _____ \n" +
    12            " |                      | \n" +
    13            " --- (@) ----- (@) -- \n";
    14        System.out.println(str);
    15    }
    16
    17    public String Note() {
    18        return "Объект: транспорт" +
    19            model + " " + number;
    20    }
    21
    22
  
```

Рисунок 5.3 – Реализация интерфейса IDrawable классом Vehicle.

На рис. 5.4 представлено определение класса Building.

```

3 public class Building extends StaticObject implements IDrawable {
4     private float width;
5     private float height;
6     private int floors;
7     private String address;
8
9     public void Draw() {
10         String str =
11             " /-/-/-/-/-/\\n" +
12             " | ____ _ |\\n" +
13             " ||_|| | ||\\n" +
14             " ||_|| | ||\\n";
15         System.out.println(str);
16     }
17
18     public String Note() {
19         return "Объект: дом" +
20             "Этажей: " + floors + "Площадь: " + getSquare();
21     }
22
23     public Building(Point pCoord, String pName) {
24         super(pCoord, pName);
25     }
26
27     public Building(int pFloors, float pW, float pH) {
28         this(new Point(), "Строение");
29         floors = pFloors;
30         width = pW; height = pH;
31     }
32
33     public float getSquare() {
34         return width*height;
35     }
36
37     public String getName() {
38         return "Строение" +
39             "\\n Этажность: " + String.valueOf(floors)
40             +"\\n Адрес: " + address;
41     }
42 }

```

Рисунок 5.4 – Определение класса Building.

Оба класса реализуют интерфейс IDrawable, но каждый вкладывает свой смысл в реализацию.

На рис. 5.5 продемонстрировано манипулирование объектами разработанной иерархии классов при использовании интерфейса.

```

6 public class Starter {
7     public static void main(String[] args) {
8
9         Vector<IDrawable> allDynamics =
10             new Vector<IDrawable>();
11         allDynamics.add(new Vehicle(
12             new Point(), "Hyundai Sol", "Y445KE", "?"));
13         allDynamics.add(new Building(5, 20.0f, 10.0f));
14         allDynamics.add(new Vehicle(
15             new Point(), "Kia Rio", "Y111KE", "новый"));
16         allDynamics.add(new Building(2, 5.0f, 7.0f));
17         allDynamics.add(new Vehicle(
18             new Point(), "Лада", "K433TE", "белый"));
19
20         // Единообразная работа со всеми элементами
21         for(int i = 0; i < allDynamics.size(); i++) {
22             System.out.println(allDynamics.get(i).Note());
23             allDynamics.get(i).Draw();
24         }
25     }
26 }

```

Рисунок 5.5 – Работа с коллекцией объектов через реализуемый интерфейс.

В данном примере добавление в коллекцию осуществляется в соответствии с правилом: добавить можно только объект того класса, который реализует интерфейс IDrawable.

На рис. 5.6 показан вывод программы.

```

D:\Server\Oracle\Middleware\Oracle_Ho
Объект: транспортHyundai Sol null
-----
  ____/          \____
 |                  |
 |---(8)-----{8}---

Объект: домЭтажей: 5Площадь: 200.0
/-/-/-/-/-/\
 |  ____  ____  |
 | |__|__| |__| |
 | |__|__| |__| |

Объект: транспортKia Rio null
-----
  ____/          \____
 |                  |
 |---(8)-----{8}---

Объект: домЭтажей: 2Площадь: 35.0
/-/-/-/-/-/\
 |  ____  ____  |
 | |__|__| |__| |
 | |__|__| |__| |

Объект: транспортЛада null
-----
  ____/          \____
 |                  |
 |---(8)-----{8}---

Process exited with exit code 0.

```

Рисунок 5.6 – Вывод программы.

Синтаксически наследование реализации (используется `extends`) и наследование интерфейсов (`implements`) отличается, но отличие существует также в идеи механизма: каждый класс может иметь только один суперкласс, а интерфейсов может реализовывать — множество. На рис. 5.5 продемонстрировано полиморфное поведение объектов коллекции, достигнутое использованием интерфейсов.

Следует понимать, что интерфейсные типы также могут образовывать иерархию.

5.4. Оборудование и материалы

Для выполнения лабораторной работы рекомендуется использовать персональный компьютер со следующими характеристиками: 64-разрядный (x64) процессор с тактовой частотой 1 ГГц и выше, оперативная память — 1 Гб и

выше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система Windows 7 и выше, Java Development Kit 8 и выше; IDE Eclipse for Java SE (or Java EE) developers.

5.5. Указания по технике безопасности

Студенты должны следовать общепринятой технике безопасности для пользователей персональных компьютеров. Не следует самостоятельно производить ремонт технических средств, установку и удаление программного обеспечения. В случае обнаружения неисправностей необходимо сообщить об этом администратору компьютерного класса (обслуживающему персоналу лаборатории).

5.6. Методика и порядок выполнения работы

Необходимо модифицировать приложение, разработанное в рамках лабораторной работы №4 следующим образом:

1. Проанализировать, какой функционал требуется дополнительно реализовать в приложении.
2. Спроектировать новые функциональные возможности с использованием инструментария диаграмм. Новый функционал добавить на базе интерфейсных типов с минимальными изменениями уже разработанной иерархии классов.
3. Модифицировать код приложения в соответствии с разработанным проектом.
4. Продемонстрируйте в программе полиморфное поведение объектов на основе использования интерфейсных типов.

5.7. Содержание отчета и его форма

Отчет по лабораторной работе должен содержать:

1. Номер и название лабораторной работы; задачи лабораторной работы.
3. Ответы на контрольные вопросы.
4. Диаграмма классов, экранные формы (консольный вывод) и листинг программного кода с комментариями, показывающие порядок выполнения лабораторной работы, и результаты, полученные в ходе её выполнения.

Отчет о выполнении лабораторной работы, подписанный студентом, сдается преподавателю.

5.8. Контрольные вопросы

1. Что такое интерфейс?
2. Опишите каким образом представляется наследование интерфейса на диаграмме классов и в коде приложения.
3. Опишите механизм реализации полиморфизма на основе использования интерфейсов.
4. Опишите синтаксис наследования интерфейсов в языке C#. Что общего в наследовании интерфейсов в C# и Java?
5. Чем интерфейс отличается от абстрактного класса?

5.9. Список литературы

Для выполнения лабораторной работы, при подготовке к защите, а также для ответа на контрольные вопросы рекомендуется использовать следующие источники: [3], [7].

ЛАБОРАТОРНАЯ РАБОТА 6. ОБОБЩЕННЫЕ ТИПЫ

6.1. Цель и содержание

Цель лабораторной работы: изучить принципы работы с обобщенными типами Java.

Задачи лабораторной работы:

- изучить теоретические принципы реализации обобщений;
- научиться определять обобщенные классы;
- научиться реализовывать приложения с использованием обобщенных типов;
- научиться применять ограничения при реализации обобщенных типов.

6.2. Формируемые компетенции

ОПК-4; ОПК-5; ОПК-6.

6.3. Теоретическая часть

6.3.1 Механизм обобщенного программирования.

Обобщения (generics) представляют собой наиболее существенное изменение в языке программирования Java со времен версии 1.0. Появление обобщений в Java 5.0 стало результатом первых требований к спецификации Java (Java Specification Requests – JSR 14), которые были сформулированы в 1999 г. Группа экспертов потратила около пяти лет на разработку спецификаций и тестовые реализации. Обобщения понадобились потому, что они позволяют писать более безопасный код, который легче читается, чем код, перегруженный переменными Object и приведениями типов. Механизм обобщения максимально используется классами коллекций.

Обобщенное программирование означает написание кода, который может быть многократно использован с объектами многих различных типов. Например, неэффективно программировать отдельные классы для коллекционирования объектов `String` и `Int`. И делать это нет необходимости – единственный `ArrayList` или `Vector` может собирать объекты любого класса. Это лишь один пример обобщенного программирования, который использовался в предыдущих лабораторных работах.

Реализовать обобщение можно средствами наследования реализации, но подобная стратегия способствует «разбуханию» кода и требует множественных конвертаций и приведений типов (подумайте почему).

6.3.2 Реализация механизма обобщения.

Такие обобщенные классы, как `ArrayList`, использовать легко. И большинство программистов Java просто применяют типы вроде `ArrayList<String>` – как если бы они были частью языка, подобно массивам `String[]`.

Но реализовать обобщенный класс не так просто. Чаще всего данные классы будут использоваться как библиотечные. Программисты, которые будут использовать обобщенный код, попытаются подставлять всевозможные классы вместо ваших параметров типа. Они ожидают, что все будет работать без досадных ограничений и запутанных сообщений об ошибках. задача разработчика как обобщенного программиста – предвидеть все возможные будущие применения обобщенного класса.

Прикладным программистам, вероятно, не придется писать много обобщенного кода. Разработчики библиотеки Java уже выполнили самую тяжелую работу и предусмотрели параметры типа для всех классов коллекций. В качестве эмпирического правила можно сказать так: от применения параметров типа выигрывает только тот код, в котором традиционно присутствует много приведений от очень общих типов (таких как `Object` или интерфейс `Comparable`).

Рассмотрение обобщений начнем с примера:

1. Требуется реализовать класс, который инкапсулирует поля и методы и описывает сущность «Пара элементов различного типа». Назовем класс MyPair.
2. Реализовать класс, описывающий сущность «Нетривиальный список», который отличается от коллекций Java ограниченным набором операций. Данный класс назовем MyCollection.

На рис. 6.1 представлен пример определения типа MyPair.

```

3 public class MyPair<T, U> {
4     private T first;
5     private U second;
6
7     public MyPair(T pF, U pS){
8         first = pF;
9         second = pS;
10    }
11
12    public T getFirst() {
13        return first;
14    }
15
16    public U getSecond() {
17        return second;
18    }
19 }

```

Рисунок 6.1 – Обобщенный класс для представления пары элементов.

Здесь T и U – параметры типа. В приведенном классе присутствуют поля first типа T и second типа U, но таких типов в данном контексте не существует. Такое объявление становится возможным, так как при объявлении класса используется MyPair<T, U> вместо простого MyPair. Указание переменных типа T и U в угловых скобках после имени класса говорит об обобщенном характере определяемого класса. Вместо T и U при объявлении переменной типа MyPair должны быть подставлены конкретные (не обобщенные) типы, например:

```

5 public class Starter {
6
7     public static void main(String[] args) {
8         MyPair<String, Integer> p1
9             = new MyPair<String, Integer>("Nikolaev E.I.", 26);
10
11         MyPair<String, Vector<Float>> p2
12             = new MyPair<String, Vector<Float>>(<
13             "Nikolaev E.I.",
14             new Vector<Float>());
15     }
16 }

```

Рисунок 6.2 – Использование обобщенного класса.

При объявлении переменной `p1` обобщенный класс конкретизируется путем присвоения переменным типа конкретных типов: `T = String`, `U = Integer`. Создается объект обобщенного типа, который содержит конкретные данные.

При создании объекта `p2` используется конкретизация: `T = String`, `U = Vector<Float>`. То есть переменные типов могут представлять как угодно сложные типы.

Из примера видно, что обобщенные типы выступают своего рода фабрикой классов.

На рис. 6.3 приводится определение класса `MyCollection`.

```

4 public class MyCollection<T> {
5     private Vector<T> vec = new Vector<T>();
6
7     public MyCollection(T item){
8         vec.add(item);
9     }
10
11     public void Add(T item){
12         vec.add(item);
13     }
14
15     public void Remove(T item){
16         vec.remove(item);
17     }
18 }

```

Рисунок 6.3 – Объявление класса `MyCollection`.

Очевидно, что при объявлении обобщенного класса можно использовать любое количество переменных типов.

При использовании обобщенных типов возникает вопрос о возможных значениях переменных типов. Иногда класс или метод нуждается в наложении ограничений на переменные типов. Операция ограничения имеет следующий синтаксис:

```
public class MyCollection<T extends Comparable> {
```

Подобное определение указывает на то, что в качестве типа T может выступать любой тип, реализующий интерфейс Comparable.

Возможны такие варианты ограничения:

```
public class MyCollection<T extends Object & Comparable & Serializable> {
```

Ограничивающие типы разделяются знаком &, потому что запятые используются для разделения переменных типа.

Как и в наследовании Java, вы можете иметь столько интерфейсных подтипов, сколько хотите, но только один из ограничивающих типов может быть классом. Если вы используете для ограничения класс, он должен быть первым в списке ограничений.

Существуют ограничения, которые накладываются на код, использующий обобщенные типы:

1. Параметры типа не могут принимать примитивные типы. Нельзя подставить примитивные тип вместо параметра типа. То есть не бывает Pair<double>, а только Pair<Double>. Причина в «подчистке» типов (компилятор работает только с конкретными типами). После подчистки класс Pair имеет поля типа Object, и их нельзя использовать их для хранения значений double.

2. Исследование типов времени выполнения работает только с «сырыми» типами. Это означает, что код

```
if(p1 instanceof MyPair)
```

выполняет только проверку принадлежности переменной p1 типу MyPair без учета конкретных значений параметра типа. Нельзя проверить:

```
if(p1 instanceof MyPair<Float, String>)
```

3. Массивы параметризованных типов не разрешены.

4. Нельзя создавать экземпляры переменных типов. Например нельзя в методах обобщенного класса использовать: `T ob = new T()`, где `T` – параметр обобщенного типа.

5. Переменные типа в статическом контексте обобщенных классов не разрешены.

6.3.3 Обобщенные методы.

Кроме обобщенных классов, программист может определять обобщенные методы, которые принадлежат классу, который не является обобщенным. На рис. 6.4 представлено определение класса, содержащего метод `WriteLine`, который оборачивает вывод в консоль, выполняемый средствами библиотеки Java:

```

3 public class Console {
4     public static <T> void WriteLine(T mes){
5         System.out.println(mes);
6     }
7
8     public static <T> void Write(T mes){
9         System.out.print(mes);
10    }
11 }

```

Рисунок 6.4 – Обобщенный метод.

Обратите внимание, что переменная типа вставляется после модификаторов (`public static` в данном случае) и перед типом возврата. Использование созданных методов и консольный вывод программы показан на рис. 6.5.

<pre> public static void main(String[] args) { Console.WriteLine("Hello, world!"); Console.Write(55.0f); } </pre>	<pre> Hello, world! 55.0 </pre>
---	----------------------------------

а)

б)

Рисунок 6.5 – Использование методов: а – код метода `main`; б – результат работы программы.

При вызове обобщенного метода в качестве параметров передаются действительные типы.

6.3.4 Ограничения переменных типов.

В предыдущих подразделах рассматривались обобщенные классы и методы, использующие переменные типа; при этом никаких сведений о типе, который представлен переменной типа программист не указывает. На практике может возникнуть потребность ограничить возможные варианты использования обобщения. Например: переменный тип может быть только классом, производным от `Object` (подтипом `Object`); переменный тип может быть только типом, реализующим определенный интерфейс и т.д. Может возникнуть необходимость наложить несколько ограничений на переменные типа.

На рис. 6.6 представлен пример подобных ограничений.

```

5 public class Console {
6     public static <T extends Object> void WriteLine(T mes){
7         System.out.println(mes);
8     }
9
10    public static <T extends Object & Comparable> void Write(T mes){
11        System.out.print(mes);
12    }
13 }

```

Рисунок 6.6 – Ограничение переменной типа.

Из примера видно, что ограничение указывается в угловых скобках после ключевого слова `extends`. Через знак `&` перечисляются интерфейсы, которые должен реализовывать переменный тип. Супертип может быть только один, он указывается первым, сразу после `extends`.

6.4. Оборудование и материалы

Для выполнения лабораторной работы рекомендуется использовать персональный компьютер со следующими характеристиками: 64-разрядный (x64) процессор с тактовой частотой 1 ГГц и выше, оперативная память – 1 Гб и

выше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система Windows 7 и выше, Java Development Kit 8 и выше; IDE Eclipse for Java SE (or Java EE) developers.

6.5. Указания по технике безопасности

Студенты должны следовать общепринятой технике безопасности для пользователей персональных компьютеров. Не следует самостоятельно производить ремонт технических средств, установку и удаление программного обеспечения. В случае обнаружения неисправностей необходимо сообщить об этом администратору компьютерного класса (обслуживающему персоналу лаборатории).

6.6. Методика и порядок выполнения работы

1. Реализуйте библиотечный класс на основе технологии обобщения в соответствии с вариантом индивидуального задания. При реализации библиотеки потребуется определить дополнительные классы для представления сущностей предметной области. Необходимо реализовать обобщение таким образом, чтобы не допустить передачу недопустимого типа, то есть требуется корректно определить ограничение переменной типа.

2. В функции main реализуйте логику приложения для демонстрации возможностей библиотеки, особенно возможности использования различных типов вместо переменных типа.

Индивидуальное задание.

Вариант	Библиотека для реализации
1, 11	Библиотека для манипулирования геометрическими объектами.
2, 12	Библиотека для манипулирования любыми числовыми типами.
3, 13	Библиотека для манипулирования должностными ставками (менеджер, директор, ...).

4, 14	Библиотека для манипулирования объектами, представляющими печатные издания.
5, 15	Библиотека для манипулирования объектами, представляющими автомобили организации.
6, 16	Библиотека для манипулирования объектами, представляющими объекты недвижимости.
7, 17	Библиотека для манипулирования объектами, представляющими банковские услуги.
8, 18	Библиотека для манипулирования объектами, представляющими офисную технику.
9, 19	Библиотека для манипулирования объектами, представляющими основные объекты информационной системы торгового предприятия: склады, товары, товарные группы, ...
10, 20	Библиотека для манипулирования объектами, представляющими основные объекты информационной системы управления предприятием: сотрудники, оргтехника, задания, проекты.

6.7. Содержание отчета и его форма

Отчет по лабораторной работе должен содержать:

1. Номер и название лабораторной работы; задачи лабораторной работы.
3. Ответы на контрольные вопросы.
4. Диаграмма классов, экранные формы (консольный вывод) и листинг программного кода с комментариями, показывающие порядок выполнения лабораторной работы, и результаты, полученные в ходе её выполнения.

Отчет о выполнении лабораторной работы, подписанный студентом, сдается преподавателю.

6.8. Контрольные вопросы

1. Что такое обобщенный тип? Что такое переменная типа?
2. Для чего используется механизм обобщения? Поясните синтаксис при реализации данного механизма. Приведите примеры.
3. Приведите примеры использования механизма обобщения при реализации типов стандартной библиотеки Java.

4. Что такое ограничения на переменные типа? Для чего используется данный механизм?

5. Какие зарезервированные слова используются для реализации механизма обобщения?

6.9. Список литературы

Для выполнения лабораторной работы, при подготовке к защите, а также для ответа на контрольные вопросы рекомендуется использовать следующие источники: [3], [7].

ЛАБОРАТОРНАЯ РАБОТА 7. КОЛЛЕКЦИИ

1. Цель и содержание

Цель лабораторной работы: изучить основные возможности, предоставляемые Java Collection Framework.

Задачи лабораторной работы:

- изучить теоретические принципы построения коллекций.
- изучить различные типы коллекций;
- научиться разрабатывать приложения и алгоритмы на основе применения коллекций.

7.2. Формируемые компетенции

ОПК-4; ОПК-5; ОПК-6.

7.3. Теоретическая часть

7.3.1 Иерархия классов коллекций.

Для Java была разработана целая иерархия коллекций, получившая название Java Collections Framework (JCF). На рис. 7.1 представлена иерархия классов. Курсивом записаны имена интерфейсов.

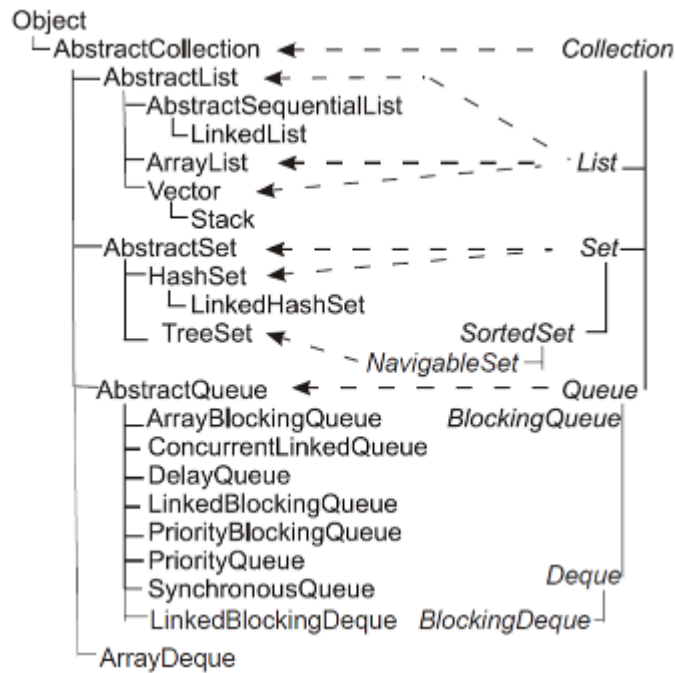


Рисунок 7.1 – Иерархия классов и интерфейсов-коллекций.

Пунктирные линии связывают интерфейсы и классы, реализующие эти интерфейсы.

Интерфейс Map не входит в эту иерархию – по мнению разработчиков Java Collections Framework, отображения типа Map не являются коллекциями. Они показаны на рис. 7.2.

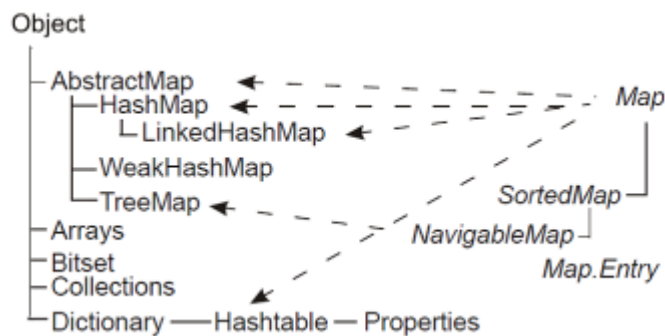


Рисунок 7.2 – Иерархия классов и интерфейсов-отображений.

Все интерфейсы, входящие в Java Collections Framework, – настраиваемые (объясните, что это значит?), их можно использовать как шаблоны классов, хранящих ссылки на элементы одного и того же типа.

В целях классификации и упорядочивания материала все типы данных библиотеки можно разделить по следующим группам:

- интерфейсы коллекций;

- готовые классы коллекций;
- абстрактные классы коллекций;
- классы представления отображений;
- классы представления списков;
- классы представления множеств;
- вспомогательные классы для работы с коллекциями.

Перед исследованием многообразия инструментов JCF рассмотрим простые коллекции и их функциональные возможности.

Можно выделить следующие коллекции JCF, используемые наиболее часто:

Таблица 7.1 – Коллекции из JCF.

Коллекция	Описание
ArrayList	Индексированная последовательность, динамически растягивающаяся и сжимающаяся
LinkedList	Упорядоченная последовательность, позволяющая эффективную вставку и удаление в любой позиции
ArrayDeque	Упорядоченная последовательность, реализованная в виде циклического массива
HashSet	Неупорядоченная коллекция, исключая дубликаты
TreeSet	Сортированный набор
EnumSet	Множество значений перечислимого типа
LinkedHashSet	Набор, запоминающий порядок вставки элементов
PriorityQueue	Коллекция, позволяющая эффективно удалять наименьший элемент
HashMap	Структура данных, сохраняющая ассоциации «ключ-значение»
TreeMap	Карта с сортированными ключами
EnumMap	Карта, в которой ключи относятся к перечислимому типу
LinkedHashMap	Карта, запоминающая порядок, в котором добавлялись элементы
WeakHashMap	Карта со значениями, которые могут удаляться сборщиком мусора, если таковые нигде не используются
IdentityHashMap	Карта с ключами, сравниваемыми с помощью ==, а не equals()

Можно использовать классификацию коллекций по их архитектурным и функциональным характеристикам, которые, в свою очередь, определяются реализуемыми интерфейсами:

Таблица 7.1 – Коллекции из JCF.

Группа коллекций	Описание
Массивы	
Связные списки	В Java все списки двунаправленные. Данные структуры сокращают время добавления и удаления элемента.
Списки-массивы	Классы ArrayList и Vector, реализующие интерфейс List. Используют возможность прямого доступа к элементам с использованием <code>get()</code> и <code>set()</code> , а не только с использованием итератора.
Хеш-наборы	Если не требуется информация о порядке элементов, то используется структура данных, которая позволит находить элементы намного быстрее. Ее недостаток в том, что эта структура не обеспечивает никакого контроля за порядком появления элементов. Эта структура данных организует элементы в порядке, удобном для ее собственных целей. Эта известная структура данных для быстрого нахождения объектов называется хеш-таблицей. Хеш-таблица вычисляет целое число, называемое хеш-кодом, для каждого объекта. Хеш-код – это целое число, которое некоторым образом производится на основе полей экземпляра объекта, причем предполагается, что объекты с разными данными порождают разные коды.
Древовидные наборы	Класс TreeSet (древовидный набор) подобен хеш-набору, но с одним дополнительным усовершенствованием. Он представляет собой сортированную коллекцию. Элементы вставляются в любом порядке. Когда происходит итерация по элементам, значения оказываются автоматически представленными в отсортированном порядке. Современная реализация использует «красно-белое дерево» (red-black tree).
Однонаправленные и двунаправленные очереди	Двусторонняя очередь (deque) позволяет эффективно добавлять и удалять элементы на обоих концах. Добавление элементов в середину очереди не поддерживается. В Java SE 6 появился интерфейс Deque. Он реализован классами ArrayDeque и LinkedList – оба представляют двустороннюю очередь, которая может расти по мере необходимости. Существуют ограниченные, однонаправленные и двунаправленные

	очереди.
Очереди с приоритетами	Типичное использование – планирование задач. Очереди с приоритетами извлекают элементы в отсортированном порядке после того, как они были вставлены в произвольном порядке.
Карты	Карта хранит пары «ключ/значение». Вы можете найти значение, если предоставите ключ. Библиотека Java предлагает две реализации карт общего назначения: HashMap и TreeMap. Оба класса реализуют интерфейс Map. Функции хеширования или сравнения применяются только к ключам. Значения, ассоциированные с ключами, не хешируются и не сравниваются.

7.3.2 Класс Vector.

Рассмотрим простой и интуитивно понятный класс Vector. В классе Vector из пакета java.util хранятся элементы типа Object, а значит, ссылки любого типа. Количество элементов может быть произвольным и не определяться заранее. Индексация элементов начинается с 0. Класс Vector аналогичен классу List библиотеки .NET Framework.

Кроме количества элементов, называемого размером (size) вектора, есть еще размер буфера – емкость (capacity) вектора. Обычно емкость совпадает с размером вектора, но можно ее увеличить методом ensureCapacity(int minCapacity) или сравнить с размером вектора методом trimToSize().

В Java 2 класс Vector переработан так, чтобы включить его в иерархию классов-коллекций. Для этого добавлено много новых методов, реализующих методы соответствующих интерфейсов-коллекций. Сейчас многие действия можно совершать старыми и новыми методами. Рекомендуется использовать новые методы, поскольку старые могут быть исключены из следующих версий Java.

Конструкторы класса Vector:

1) Vector() – создает пустой объект нулевой длины с емкостью в 10 элементов;

2) Vector(int capacity) – создает пустой объект указанной емкости capacity;

3) `Vector(int capacity, int increment)` – формирует пустой объект указанной емкости `capacity` и задает число `increment`, на которое увеличивается емкость при необходимости;

4) `Vector(Collection c)` – вектор создается по указанной коллекции.

В векторе разрешено хранить объекты разных типов, поскольку на самом деле в нем хранятся не значения объектов, а ссылки на них. Класс `Vector` настраиваемый, и если предполагается заполнять вектор ссылками одного и того же типа, то этот тип можно задать при создании вектора в шаблоне (generic) коренного типа, в угловых скобках:

```
Vector<String> v = new Vector<String>();
```

или сокращенная запись:

```
Vector<String> v = new Vector<>();
```

Методы класса `Vector` приводятся в таблице 7.1.

Таблица 7.1 – Методы класса `Vector`.

Метод	Описание
<code>add(Object element)</code>	Позволяет добавить элемент в конец вектора. То же делает старый метод <code>addElement(Object element)</code> .
<code>add(int index, Object element)</code>	Вставляет элемент в указанное место <code>index</code> . Элемент, находившийся на этом месте, и все последующие элементы сдвигаются, их индексы увеличиваются на единицу (старый метод <code>insertElementAt(Object element, int index)</code>).
<code>addAll(Collection coll)</code>	Добавляет в конец вектора все элементы коллекции <code>coll</code> .
<code>addAll(int index, Collection coll)</code>	Вставляет в позицию <code>index</code> все элементы коллекции <code>coll</code> .
<code>set(int index, Object element)</code>	Заменяет элемент, стоявший в векторе в позиции <code>index</code> , на элемент <code>element</code> . Старый метод <code>setElementAt(Object element, int index)</code> .
<code>size()</code>	Возвращает размер коллекции.
<code>capacity()</code>	Возвращает емкость коллекции.
<code>isEmpty()</code>	Возвращает <code>true</code> , если в векторе нет ни одного элемента.
<code>firstElement()</code>	Возвращает первый элемент.
<code>lastElement()</code>	Возвращает последний элемент

Метод	Описание
get(int index)	Возвращает элемент с индексом index. Старый метод – elementAt(int index).
toArray() или toArray(Object[] a)	Возвращает все элементы вектора в виде массива типа Object[]. Во втором методе заполняется массив a, если в нем достаточно места.
contains(Object element)	Возвращает true, если элемент element находится в векторе.
containsAll(Collection c)	Возвращает true, если вектор содержит все элементы указанной коллекции
indexOf(Object element)	Возвращает индекс первого появления элемента element в векторе.
indexOf(Object element, int begin)	Ведет поиск, начиная с индекса begin включительно.
lastIndexOf(Object element)	Возвращает индекс последнего появления элемента в векторе.
lastIndexOf(Object element, int start)	Ведет поиск от индекса start включительно к началу вектора.
remove(Object element)	Удаляет из вектора первое вхождение указанного элемента element. Старый метод removeElement(Object element).
remove(int index)	Удаляет элемент из позиции index и возвращает его в качестве своего результата типа Object. Старый метод removeElementAt(int index).
removeRange(int begin, int end)	Удаляет элементы от позиции begin включительно до позиции end исключительно.
removeAll(Collection coll)	Удаляет из данного вектора все элементы коллекции coll.
setSize(int newSize)	Удаляет последние элементы урезав вектор. Например удалить все элементы можно вызовом setSize(0).
retainAll(Collection coll)	Удаляет все элементы, кроме входящих в указанную коллекцию coll.
clear()	Удаляет все элементы вектора. Старый метод removeAllElements().

Класс Vector является примером того, как можно объекты класса Object, а значит, любые объекты, объединить в коллекцию. Этот тип коллекции упорядочивает и даже нумерует элементы. В векторе есть первый элемент, есть последний элемент. К каждому элементу обращаются непосредственно по

индексу. При добавлении и удалении элементов оставшиеся элементы автоматически перенумеровываются.

7.3.3 Класс Stack.

Класс Stack из пакета java.util объединяет элементы в стек и расширяет класс Vector. Стек (stack) реализует порядок работы с элементами по принципу LIFO (Last In – First Out).

Перед работой создается пустой стек конструктором Stack().

Дополнительно к методам класса Vector класс Stack содержит пять методов, позволяющих работать с коллекцией как со стеком (таблица 7.2).

Таблица 7.2 – Методы класса Stack.

Метод	Описание
push(Object item)	Помещает элемент item в стек.
pop()	Извлекает верхний элемент из стека.
peek()	Читает верхний элемент, не извлекая его из стека.
empty()	Проверяет, не пуст ли стек.
search(Object item)	Находит позицию элемента item в стеке. Если элемент не найден, возвращается -1.

7.3.4 Класс Hashtable.

Класс Hashtable расширяет абстрактный класс Dictionary. В объектах этого класса хранятся пары «ключ – значение». Их можно представить себе как таблицу из двух столбцов. Такие таблицы часто называют словарями или ассоциативными массивами.

Каждый объект класса Hashtable кроме размера (size) – количества пар, имеет еще две характеристики: емкость (capacity) – размер буфера, и показатель загруженности (load factor) – процент заполненности буфера, по достижении которого увеличивается емкость таблицы.

Конструкторы класса Hashtable:

1) Hashtable() – создает пустой объект с начальной емкостью в 101 элемент и показателем загруженности 0,75;

2) `Hashtable(int capacity)` – формирует пустой объект с начальной емкостью `capacity` и показателем загруженности `0,75`;

3) `Hashtable(int capacity, float loadFactor)` – создает пустой объект с начальной емкостью `capacity` и показателем загруженности `loadFactor`;

4) `Hashtable(Map f)` – создает объект класса `Hashtable`, содержащий все элементы отображения `f`, с емкостью, равной удвоенному числу элементов отображения `f`, но не менее `11`, и показателем загруженности `0,75`.

Если предполагается хранить в таблице пары определенных типов, то эти типы можно указать заранее при создании таблицы. Для этого в угловых скобках в шаблоне (generics) коренного типа записываются конкретные типы:

```
Hashtable<Integer, String> h = new Hashtable<Integer, String>();
```

Можно использовать сокращенную форму записи:

```
Hashtable<Integer, String> h = new Hashtable<>();
```

После такого определения компилятор будет следить за типами заносимых в таблицу элементов. Извлечение элементов из таблицы не потребует явного приведения типов.

Методы класса `Hashtable` представлены в таблице 7.3.

Таблица 7.3 – Методы класса `Hashtable`.

Метод	Описание
<code>Object put(Object key, Object value)</code>	Добавляет пару «ключ – значение», если ключа <code>key</code> не было в таблице, и меняет значение <code>value</code> ключа <code>key</code> , если он уже есть в таблице. Возвращает старое значение ключа или <code>null</code> , если его не было. Если хотя бы один аргумент равен <code>null</code> , возникает исключительная ситуация.
<code>void putAll(Map f)</code>	добавляет все элементы отображения <code>f</code> .
<code>get(Object key)</code>	Возвращает значение элемента с ключом <code>key</code> в виде объекта класса <code>Object</code> или того класса, с которым создана таблица.
<code>containsKey(Object key)</code>	Возвращает <code>true</code> , если в таблице есть ключ <code>key</code> .
<code>containsValue(Object value)</code>	Возвращает <code>true</code> , если в таблице есть ключи со значением <code>value</code> .
<code>isEmpty()</code>	Возвращает <code>true</code> , если в таблице нет элементов.

Метод	Описание
values()	Представляет все значения value таблицы в виде объекта типа Collection.
keySet()	Предоставляет все ключи key таблицы в виде объекта типа интерфейса Set. Все изменения в этом объекте типа Set корректируют таблицу, и наоборот.
entrySet()	Представляет все пары «ключ – значение» таблицы в виде объекта типа интерфейса Set. Все модификации в этом объекте типа Set изменяют таблицу, и наоборот.
toString()	Возвращает строку, содержащую все пары таблицы.
remove(Object key)	Удаляет пару с ключом key, возвращая значение этого ключа, если оно есть, и null, если пара с ключом key не найдена.
clear()	Удаляет все элементы, очищая таблицу.

7.3.5 Класс Properties.

Класс Properties расширяет класс Hashtable таким образом, что в нем хранятся пары ссылок не на произвольный тип, а на строки – пары типа String. Он предназначен в основном для работы с парами «свойства системы – их значения», записанными в файлах свойств.

В классе Properties два конструктора:

- 1) Properties() – создает пустой объект;
- 2) Properties(Properties default) – создает объект с заданными парами свойств default.

Методы класса Properties представлены в таблице 7.4.

Таблица 7.4 – Методы класса Properties.

Метод	Описание
setProperty(String key, String value)	Добавляет новую пару, если ключа key нет, и меняет значение, если ключ key есть.
load(InputStream in)	Загружает свойства из входного потока in.
list(PrintStream out), list(PrintWriter out)	Выводят свойства в выходной поток out.
store(OutputStream out, String header)	Выводит свойства в выходной поток out с заголовком header.

На рис. 7.3 представлен пример листинга программы, использующей класс Properties. Это простая программа, позволяющая вывести все системные свойства Java.

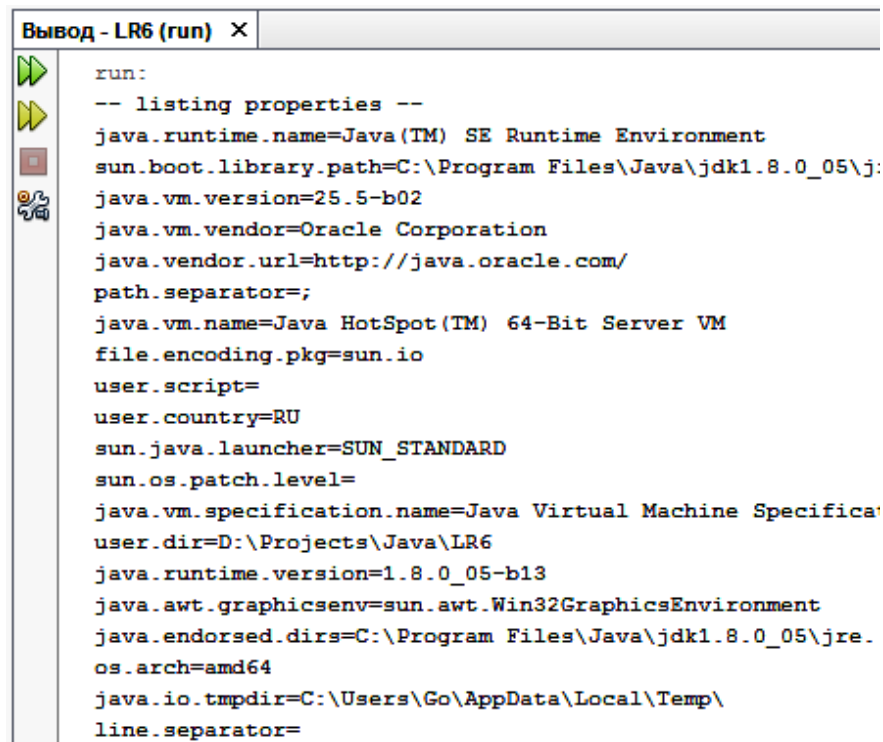
```

1  package lr6;
2
3  public class LR6 {
4      public static void main(String[] args) {
5          System.getProperties().list(System.out);
6      }
7  }

```

Рисунок 7.3 – Пример использования класса Properties.

На рис. 7.4 представлен результат работы программы (фрагмент вывода в консоль). Создайте подобную программу и изучите свойства Java в рамках лабораторной работы.



```

Вывод - LR6 (run) X
run:
-- listing properties --
java.runtime.name=Java(TM) SE Runtime Environment
sun.boot.library.path=C:\Program Files\Java\jdk1.8.0_05\j
java.vm.version=25.5-b02
java.vm.vendor=Oracle Corporation
java.vendor.url=http://java.oracle.com/
path.separator=;
java.vm.name=Java HotSpot(TM) 64-Bit Server VM
file.encoding.pkg=sun.io
user.script=
user.country=RU
sun.java.launcher=SUN_STANDARD
sun.os.patch.level=
java.vm.specification.name=Java Virtual Machine Specifica
user.dir=D:\Projects\Java\LR6
java.runtime.version=1.8.0_05-b13
java.awt.graphicsenv=sun.awt.Win32GraphicsEnvironment
java.endorsed.dirs=C:\Program Files\Java\jdk1.8.0_05\jre.
os.arch=amd64
java.io.tmpdir=C:\Users\Go\AppData\Local\Temp\
line.separator=

```

Рисунок 7.4 – Вывод программы с использованием класса Properties.

7.3.6 Интерфейс Collection.

Интерфейс `Collection` из пакета `java.util` описывает общие свойства коллекций `List`, `Set` и `Queue`. Он содержит методы добавления и удаления элементов, проверки и преобразования элементов (таблица 7.5).

Таблица 7.5 – Методы интерфейса `Collection`.

Метод	Описание
<code>boolean add(Object obj)</code>	Добавляет элемент <code>obj</code> в конец коллекции; возвращает <code>false</code> , если такой элемент в коллекции уже есть, а коллекция не допускает повторяющиеся элементы; возвращает <code>true</code> , если добавление прошло успешно.
<code>boolean addAll(Collection coll)</code>	Добавляет все элементы коллекции <code>coll</code> в конец данной коллекции.
<code>void clear()</code>	Удаляет все элементы коллекции.
<code>boolean contains(Object obj)</code>	Проверяет наличие элемента <code>obj</code> в коллекции.
<code>boolean containsAll(Collection coll)</code>	Проверяет наличие всех элементов коллекции <code>coll</code> в данной коллекции.
<code>boolean isEmpty()</code>	Проверяет, пуста ли коллекция.
<code>Iterator iterator()</code>	Возвращает итератор данной коллекции.
<code>boolean remove(Object obj)</code>	Удаляет указанный элемент из коллекции; возвращает <code>false</code> , если элемент не найден, <code>true</code> , если удаление прошло успешно.
<code>boolean removeAll(Collection coll)</code>	Удаляет элементы указанной коллекции, лежащие в данной коллекции.
<code>boolean retainAll(Collection coll)</code>	Удаляет все элементы данной коллекции, кроме элементов коллекции <code>coll</code> .
<code>int size()</code>	Возвращает количество элементов в коллекции.
<code>Object[] toArray()</code>	Возвращает все элементы коллекции в виде массива.
<code>Object[] toArray(Object[] a)</code>	Записывает все элементы коллекции в массив <code>a</code> , если в нем достаточно места.

7.3.7 Интерфейс `List`.

Интерфейс `List` из пакета `java.util`, расширяющий интерфейс `Collection`, описывает методы работы с упорядоченными коллекциями. Иногда их называют последовательностями (*sequence*). Элементы такой коллекции пронумерованы, начиная от нуля, к ним можно обратиться по индексу. В

отличие от коллекции Set элементы коллекции List могут повторяться. Класс Vector – одна из реализаций интерфейса List.

Интерфейс List добавляет к методам интерфейса Collection методы, использующие индекс index элемента (таблица 7.6).

Таблица 7.6 – Методы интерфейса List.

Метод	Описание
<code>void add(int index, Object obj)</code>	Вставляет элемент <code>obj</code> в позицию <code>index</code> ; старые элементы, начиная с позиции <code>index</code> , сдвигаются, их индексы увеличиваются на единицу.
<code>boolean addAll(int index, Collection coll)</code>	Вставляет все элементы коллекции <code>coll</code> .
<code>Object get(int index)</code>	Возвращает элемент, находящийся в позиции <code>index</code> .
<code>int indexOf(Object obj)</code>	Возвращает индекс первого появления элемента <code>obj</code> в коллекции.
<code>int lastIndexOf(Object obj)</code>	Возвращает индекс последнего появления элемента <code>obj</code> в коллекции.
<code>ListIterator listIterator()</code>	Возвращает итератор коллекции.
<code>ListIterator listIterator(int index)</code>	Возвращает итератор конца коллекции от позиции <code>index</code> .
<code>Object set(int index, Object obj)</code>	Заменяет элемент, находящийся в позиции <code>index</code> , элементом <code>obj</code> .
<code>List subList(int from, int to)</code>	Возвращает часть коллекции от позиции <code>from</code> включительно до позиции <code>to</code> исключительно.

7.3.8 Интерфейс Set.

Интерфейс Set из пакета `java.util`, расширяющий интерфейс Collection, описывает неупорядоченную коллекцию, не содержащую повторяющихся элементов. Это соответствует математическому понятию множества (set). Такие коллекции удобны для проверки наличия или отсутствия у элемента свойства, определяющего множество. Новые методы в интерфейс Set не добавлены, просто метод `add()` не станет добавлять еще одну копию элемента, если такой элемент уже есть в множестве. Этот интерфейс расширен интерфейсом SortedSet.

7.3.9 Интерфейс SortedSet.

Интерфейс SortedSet из пакета java.util, расширяющий интерфейс Set, описывает упорядоченное множество, отсортированное по естественному порядку возрастания его элементов или по порядку, заданному какой-либо реализацией интерфейса Comparator.

Дополнительные методы интерфейса SortedSet представлены в таблице 7.7.

Таблица 7.7 – Методы интерфейса SortedSet.

Метод	Описание
Comparator comparator()	Возвращает способ упорядочения коллекции.
Object first()	Возвращает первый, меньший элемент коллекции.
SortedSet headSet(Object toElement)	Возвращает начальные, меньшие элементы до элемента toElement исключительно.
Object last()	Возвращает последний, больший элемент коллекции.
SortedSet subset (Object fromElement, Object toElement)	Возвращает подмножество коллекции от элемента fromElement включительно до элемента toElement исключительно.
SortedSet tailSet(Object fromElement)	Возвращает последние, большие элементы коллекции от элемента fromElement включительно.

7.3.10 Интерфейс NavigableSet.

Интерфейс NavigableSet из пакета java.util, расширяющий интерфейс SortedSet, описывает отсортированное множество, в котором можно организовать бинарный поиск.

Чтобы осуществить это, в интерфейсе описаны методы, позволяющие для каждого данного элемента множества найти ближайший больший и ближайший меньший элементы того же множества (таблица 7.8).

Таблица 7.8 – Методы интерфейса NavigableSet.

Метод	Описание
Object lower(Object elem)	Возвращает ссылку на наибольший элемент множества, меньший данного элемента elem.
Object floor(Object elem)	Возвращает ссылку на наибольший элемент множества, меньший или равный данному элементу elem.
Object higher(Object elem)	Возвращает ссылку на наименьший элемент множества, больший данного элемента elem.
Object ceiling(Object elem)	Возвращает ссылку на наименьший элемент множества, больший или равный данному элементу elem.
NavigableSet subSet(Object fromElement, boolean fromInclusive, Object toElement, boolean toInclusive)	Возвращает подмножество коллекции от элемента fromElement включительно, если fromInclusive == true, или исключительно, если fromInclusive == false, до элемента toElement включительно или исключительно в зависимости от истинности последнего параметра toInclusive.
NavigableSet headSet(Object toElement, boolean inclusive)	Возвращает начальные, меньшие элементы до элемента toElement включительно или исключительно в зависимости от истинности параметра inclusive.
NavigableSet tailSet(Object fromElement, boolean inclusive)	Возвращает последние, большие элементы коллекции от элемента fromElement включительно или исключительно в зависимости от истинности параметра inclusive.
Object pollFirst()	Возвращает ссылку на наименьший элемент множества и удаляет его.
Object pollLast()	Возвращает ссылку на наибольший элемент множества и удаляет его.

7.3.11 Интерфейс Queue.

Интерфейс Queue из пакета java.util, расширяющий интерфейс Collection, описывает методы работы с очередями, то есть структурами данных, организованными по принципу FIFO (First In – First Out, «первым пришел – первым ушел»). Интерфейс Queue добавляет к методам интерфейса Collection методы, характерные для очередей (таблица 7.9).

Таблица 7.9 – Методы интерфейса Queue.

Метод	Описание
Object element()	Возвращает первый элемент очереди, не удаляя его из очереди.
Object peek()	Возвращает первый элемент очереди, не удаляя его.
Object remove()	Возвращает первый элемент очереди и удаляет его из очереди.
Object poll()	Возвращает первый элемент очереди и удаляет его из очереди.
boolean offer(Object obj)	Вставляет элемент в конец очереди и возвращает true, если вставка удалась.

В библиотеке Java существуют также следующие интерфейсы, предназначенные для организации специфических структур данных:

1. Интерфейс `BlockingQueue`. Содержится в пакете `java.util.concurrent` и расширяет интерфейс `Queue`. Данный интерфейс описывает очередь, с которой работают одновременно несколько подпроцессов, вставляющих и удаляющих элементы.

2. Интерфейс `Deque`. Интерфейс `Deque` (double ended queue) из пакета `java.util`, расширяющий интерфейс `Queue`, описывает методы работы с разновидностью очередей, называемой деком, у которого элементы вставляются и удаляются с обоих концов.

3. Интерфейс `BlockingDeque`. Интерфейс `BlockingQueue` из пакета `java.util.concurrent`, расширяющий интерфейсы `Queue` и `Deque`, описывает дек, с которым работают одновременно несколько подпроцессов, вставляющих и удаляющих элементы. Их работа организуется таким образом, чтобы подпроцесс, пытающийся забрать элемент из пустого дека, ждал, когда другой подпроцесс занесет в него хотя бы один элемент.

Методы данных интерфейсов необходимо рассмотреть самостоятельно с использованием документации Java.

7.3.12 Интерфейс Map.

Интерфейс Map из пакета java.util описывает своеобразную коллекцию, состоящую не из элементов, а из пар «ключ – значение». У каждого ключа может быть только одно значение, что соответствует математическому понятию однозначной функции, или отображения (map).

Такую коллекцию часто называют еще словарем (dictionary) или ассоциативным массивом (associative array).

Следует понимать, что любой программист, работавший с массивами, уже знаком с простейшим примером словаря. Массив – это простейший словарь, представляющий пары «ключ – значение» в виде «индекс массива – элемент массива».

Класс Hashtable – одна из реализаций интерфейса Map.

Методы интерфейса Map представлены в таблице 7.10.

Таблица 7.10 – Методы интерфейса Map.

Метод	Описание
boolean containsKey(Object key)	Проверяет наличие ключа key.
boolean containsValue(Object value)	Проверяет наличие значения value.
Set entrySet()	Представляет коллекцию в виде множества с элементами в виде пар из данного отображения, с которыми можно работать методами вложенного интерфейса Map.Entry.
Object get(Object key)	Возвращает значение, отвечающее ключу key.
Set keySet()	Представляет ключи коллекции в виде множества.
Object put(Object key, Object value)	Добавляет пару «ключ – значение», если такой пары не было, и заменяет значение ключа key, если такой ключ уже есть в коллекции.
void putAll(Map m)	Добавляет к коллекции все пары из отображения m.
Collection values()	Представляет все значения в виде коллекции.

В интерфейс Map вложен интерфейс Map.Entry, содержащий методы работы с отдельной парой отображения.

Этот интерфейс описывает методы работы с парами, полученными методом `entrySet()`:

- методы `getKey()` и `getValue()` позволяют получить ключ и значение пары;
- метод `setValue(Object value)` меняет значение в данной паре.

В библиотеке Java существуют также интерфейсы:

1. Интерфейс `SortedMap`. Расширяет интерфейс `Map`, описывает упорядоченную по ключам коллекцию `Map`. Сортировка производится либо в естественном порядке возрастания ключей, либо в порядке, описываемом в интерфейсе `Comparator`.

2. Интерфейс `NavigableMap`. Расширяет интерфейс `SortedMap`, описывает отсортированное по ключам отображение, в котором можно организовать бинарный поиск.

Методы данных интерфейсов необходимо изучить самостоятельно с использованием документации Java. Необходимо также рассмотреть интерфейсы `Iterator` и `ListIterator`, которые имеют важное значение для построения алгоритмических конструкций последовательного обхода элементов коллекций.

Таким образом, очевидно, что библиотека Java обладает обширной коллекцией интерфейсов. Каждый интерфейс предназначен для организации коллекции с определенными свойствами. При этом, следует понимать, что для организации коллекции программисту доступны несколько различных вариантов разработки.

7.3.13 Абстрактные классы коллекций.

Программист может создать свои коллекции, реализовав соответствующие интерфейсы. Это трудоемкая процедура, поскольку в интерфейсах много методов. Чтобы упростить данную задачу, в Java Collections Framework введены частичные реализации интерфейсов – абстрактные классы-коллекции.

Эти классы лежат в пакете `java.util`. Абстрактный класс `AbstractCollection` реализует интерфейс `Collection`, но оставляет нереализованными методы `iterator()`, `size()`.

Абстрактный класс `AbstractList` реализует интерфейс `List`, но оставляет нереализованным метод `get(int)` и унаследованный метод `size()`. Этот класс позволяет реализовать коллекцию с прямым доступом к элементам, подобно массиву.

Абстрактный класс `AbstractSequentialList` реализует интерфейс `List`, но оставляет нереализованным метод `listIterator(int index)` и унаследованный метод `size()`. Данный класс позволяет реализовать коллекции с последовательным доступом к элементам с помощью итератора `ListIterator`.

Абстрактный класс `AbstractSet` реализует интерфейс `Set`, но оставляет нереализованными методы, унаследованные от `AbstractCollection`.

Абстрактный класс `AbstractQueue` реализует интерфейс `Queue`, но оставляет нереализованными методы, унаследованные от `AbstractCollection`.

Абстрактный класс `AbstractMap` реализует интерфейс `Map`, но оставляет нереализованным метод `entrySet()`.

Наконец, в составе Java API есть полностью реализованные классы-коллекции. Кроме классов `Vector`, `Stack`, `Hashtable` и `Properties`, это классы `ArrayList`, `LinkedList`, `HashSet`, `TreeSet`, `HashMap`, `TreeMap`, `WeakHashMap` и много других классов.

Для работы с указанными классами разработаны интерфейсы `Iterator`, `ListIterator`, `Comparator` и классы `Arrays` и `Collections`.

7.3.14 Итератор.

После внедрения в практику программирования механизма коллекций, был выработан механизм обхода элементов коллекций. В Java Collections Framework введен интерфейс `Iterator`, описывающий способ обхода всех элементов коллекции. В каждой коллекции есть метод `iterator()`, возвращающий реализацию интерфейса `Iterator` для указанной коллекции. Получив эту

реализацию, можно обходить коллекцию в некотором порядке, определенном данным итератором, с помощью методов, описанных в интерфейсе `Iterator` и реализованных в этом итераторе.

В интерфейсе `Iterator` представлены всего три метода:

- логический метод `hasNext()` возвращает `true`, если обход еще не завершен;
- метод `next()` делает текущим следующий элемент коллекции и возвращает его в виде объекта класса `Object`;
- метод `remove()` удаляет текущий элемент коллекции.

Фактически, итератор – это указатель на элемент коллекции. При создании итератора указатель устанавливается перед первым элементом, метод `next()` перемещает указатель на первый элемент. Следующее применение метода `next()` перемещает указатель на второй элемент коллекции и т.д.

Метод `remove()` позволяет при просмотре коллекции удалять из нее ненужные элементы, сохраняя при этом порядок следования элементов. В то время как метод `remove()` самой коллекции, удалив элемент, перестроит оставшиеся элементы коллекции и итератор может неправильно просмотреть оставшуюся часть коллекции.

Рассмотрим пример использования интерфейса `Iterator` (рис. 7.5).

```

9  public static void main(String[] args) {
10     Collection<String> col = new Vector<String>();
11     Iterator<String> iter = col.iterator();
12     while(iter.hasNext()){
13         System.out.println(iter.next());
14     }

```

Рисунок 7.5 – Использование класса `Iterator`.

Начиная с Java SE 5.0 появилось синтаксис, который позволяет сократить код, представленный рис. 7.5.

```

9  public static void main(String[] args) {
10     Collection<String> col = new Vector<String>();
11     for(String item: col){
12         System.out.println(item);
13     }
14 }

```

Рисунок 7.6 – Косвенное использование класса Iterator.

Компилятор просто транслирует цикл «for each» в цикл с итератором.

Цикл «for each» работает с любым объектом, реализующим интерфейс Iterable, который имеет единственный метод:

```

public interface Iterable<E>{
    Iterator<E> iterator();
}

```

Интерфейс Collection расширяет интерфейс Iterable. Поэтому вы можете использовать цикл «for each» с любой коллекцией из стандартной библиотеки.

Порядок, в котором перебираются элементы, зависит от типа коллекции. Если проходится коллекция ArrayList, то итератор начинает с индекса 0 и увеличивает его значение на каждом шаге. Однако если итератор работает по элементам HashSet, то они будут возвращаться в совершенно случайном порядке. Но за время итерации переберутся все элементы коллекции, хотя нельзя сделать никаких предположений относительно их порядка. Обычно это не является проблемой, потому что порядок не имеет значения при таких вычислениях, как, например, подсчет суммы или количества совпадений.

7.3.15 Варианты создания классов-коллекций.

Возможными путями создания и использования классов-коллекций являются:

1. Применение готовых классов, имеющихся в библиотеке Java. Эти классы уже определены, определены их методы. Каждый такой класс реализует один или несколько интерфейсов. К таким классам относятся: Hashtable, Stack,

Vector, Properties, ArrayList, ArrayDeque, LinkedList, HashSet, TreeSet, HashMap, TreeMap, WeakHashMap.

2. Непосредственная реализация интерфейса. Программист может самостоятельно реализовать интерфейс (или несколько) для организации своей коллекции со специфичными свойствами.

3. В Java Collection Framework существует также возможность реализации своего типа коллекции через расширение абстрактного класса-коллекции. Смысл подобных абстрактных классов – облегчить труд программиста. Как известно, при реализации интерфейса, программист должен реализовать все его методы. Это достаточно трудоемкий процесс. Абстрактные классы коллекций уже реализовали часть методов, то есть программисту остается лишь уточнить часть требуемых методов. К абстрактным классам коллекций относят: `AbstractCollection`, `AbstractList`, `AbstractSequentialList`, `AbstractSet`, `AbstractQueue`, `AbstractMap`.

7.4. Оборудование и материалы

Для выполнения лабораторной работы рекомендуется использовать персональный компьютер со следующими характеристиками: 64-разрядный (x64) процессор с тактовой частотой 1 ГГц и выше, оперативная память – 1 Гб и выше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система Windows 7 и выше, Java Development Kit 8 и выше; IDE Eclipse for Java SE (or Java EE) developers.

7.5. Указания по технике безопасности

Студенты должны следовать общепринятой технике безопасности для пользователей персональных компьютеров. Не следует самостоятельно производить ремонт технических средств, установку и удаление программного

обеспечения. В случае обнаружения неисправностей необходимо сообщить об этом администратору компьютерного класса (обслуживающему персоналу лаборатории).

7.6. Методика и порядок выполнения работы

1. Рассмотрим пример решения задачи с использованием класса коллекции.

Задание. Требуется реализовать хранение логинов и паролей пользователей в структуре данных, а также поиск информации о пользователях по введенному логину и паролю.

Решение. Для решения задачи будем использовать коллекцию Hashtable. После формирования коллекции пользователю предоставляется возможность получить доступ к полной информации о пользователе с логином и паролем, введенным оператором. На рис. 7.7 представлен код класса UserInfo. Данный класс является хранилищем информации о пользователе. Данных о логине и пароле в данном классе нет.

```
1 package lesson7_Task;
2
3 public class UserInfo {
4
5     public UserInfo(String pFio,
6                     String pMail,
7                     String pCity){
8         fio = pFio;
9         mail = pMail;
10        city = pCity;
11    }
12
13    public String fio;
14    public String mail;
15    public String city;
16 }
```

Рисунок 7.7 – Реализация класса UserInfo.

На рис. 7.8 представлен листинг класса для представления пары «логин – пароль».

```

1 package lesson7_Task;
2
3 public class AuthorizeInfo {
4     private String login;
5     private String pwd;
6
7     public AuthorizeInfo(String pLogin, String pPwd){
8         login = pLogin;
9         pwd = pPwd;
10    }
11
12    @Override
13    public boolean equals(Object ob){
14        if(!(ob instanceof AuthorizeInfo))
15            return false;
16        AuthorizeInfo a = (AuthorizeInfo)ob;
17        if((this.login.equals(a.login)) && (this.pwd.equals(a.pwd)))
18            return true;
19        return false;
20    }
21
22    @Override
23    public int hashCode(){
24        int res = 1;
25        res = res * 31 + login.length();
26        res = res * 31 + pwd.length();
27        return res;
28    }
29 }

```

Рисунок 7.8 – Реализация класса AuthorizeInfo.

Для установления соответствия между объектами класса AuthorizeInfo и объектами класса UserInfo реализуем класс UserStorage (рис. 7.9).

```

6 public class UserStorage{
7
8     public UserStorage(){
9         data = new Hashtable<AuthorizeInfo, UserInfo>();
10        data.put(new AuthorizeInfo("1", "11"),
11                new UserInfo("Novak E.A.", "novak@serv.com", "Stavropol"));
12        data.put(new AuthorizeInfo("2", "22"),
13                new UserInfo("Storzev E.P.", "storzev@serv.com", "Moskow"));
14        data.put(new AuthorizeInfo("3", "33"),
15                new UserInfo("Gumin T.S.", "gumin@serv.com", "New York"));
16        data.put(new AuthorizeInfo("4", "44"),
17                new UserInfo("Shildin F.A.", "shildin@serv.com", "Paris"));
18        data.put(new AuthorizeInfo("5", "55"),
19                new UserInfo("Skripin T.T.", "skripin@serv.com", "Krasnodar"));
20        data.put(new AuthorizeInfo("6", "66"),
21                new UserInfo("Gunin F.F.", "gunin@serv.com", "Georgievsk"));
22    }
23
24    private Hashtable<AuthorizeInfo, UserInfo> data;
25
26    public UserInfo getUser(AuthorizeInfo ob){
27        if(data.containsKey(ob))
28            return data.get(ob);
29        else return null;
30    }
31 }

```

Рисунок 7.9 – Реализация класса UserStorage.

После реализации классов `AuthorizeInfo`, `UserStorage` и `UserInfo` можно реализовать логику метода `main()`. На рис. 7.10 представлена реализация данного метода.

```

5 public class LogoExample {
6
7     public static void main(String[] args) {
8         UserStorage task = new UserStorage();
9         Scanner sc = new Scanner(System.in);
10        System.out.print("Введите логин > ");
11        String login = sc.next();
12        System.out.print("Введите пароль > ");
13        String pwd = sc.next();
14        UserInfo user = task.getUser(new AuthorizeInfo(login, pwd));
15        if(user != null)
16        {
17            System.out.println(user.fio);
18            System.out.println(user.city);
19            System.out.println(user.mail);
20        }
21        else
22            System.out.println("нет такого пользователя!");
23    }
24 }
25 }

```

Рисунок 7.10 – Метод `main()`.

На рис. 7.11 показан результат работы программы (консольный вывод).

```

Введите логин > 3
Введите пароль > 33

Gumin T.S.
New York
gumin@serv.com

```

Рисунок 7.11 – Консольный вывод программы.

2. Перед выполнением индивидуального задания ответьте на следующий вопрос: для чего в классе `AuthorizeInfo` реализованы методы `equals()` и `hashCode()`?

3. Выполните индивидуальное задание. В каждом задании необходимо самостоятельно спроектировать классы предметной области, определить коллекцию, которая наиболее удовлетворяет целям проектирования.

Индивидуальное задание.

№ п/п	Задание
1	Реализовать поиск, добавление и удаление информации о товарах предприятия; товары сгруппированы по товарным группам.

2	Реализовать поиск, добавление и удаление информации о сотрудниках предприятия; сотрудники учитываются в разрезе возрастных групп.
3	Реализовать поиск, добавление и удаление информации о книжных изданиях; книги сгруппированы по жанру.
4	Реализовать поиск, добавление и удаление информации о рейсах поездов; рейсы группируются по дальности следования.
5	Реализовать поиск, добавление и удаление информации о расписании автобусов.
6	Реализовать поиск, добавление и удаление информации о расчетах за электроэнергию.
7	Реализовать поиск, добавление и удаление информации о платежах пользователей интернет-магазина; у каждого пользователя может быть несколько платежей.
8	Реализовать поиск, добавление и удаление информации о сотрудниках, каждый из которых может быть материально ответственным за определенное количество объектов учета.
9	Реализовать поиск, добавление и удаление информации о строительных бригадах; каждая бригада может работать на нескольких объектах.
10	Реализовать поиск, добавление и удаление информации о транспортных средствах предприятия.
11	Реализовать поиск, добавление и удаление информации о кредитных услугах банка.
12	Реализовать поиск, добавление и удаление информации о штатной структуре предприятия; в каждом отделе может быть несколько сотрудников.
13	Реализовать поиск, добавление и удаление информации о товарах; каждый товар входит в определенную товарную категорию.
14	Реализовать поиск, добавление и удаление информации о студентах и группах.
15	Реализовать поиск, добавление и удаление информации о курсовых работах; курсовые работы ведутся в разрезе дисциплин.
16	Реализовать поиск, добавление и удаление информации о преподавателях; все преподаватели работают на кафедрах.
17	Реализовать поиск, добавление и удаление информации о канцелярских товарах.
18	Реализовать поиск, добавление и удаление информации об объектах недвижимости.
19	Реализовать поиск, добавление и удаление информации о строительных материалах; каталог материалов организован по категориям.
20	Реализовать поиск, добавление и удаление информации о фильмотеке; все фильмы сгруппированы по жанрам.
21	Реализовать поиск, добавление и удаление информации о товарах магазина; все товары разбиты по поставщикам.
22	Реализовать поиск, добавление и удаление информации о клиентах

	банка; все клиенты сгруппированы по группам оказываемых услуг.
23	Реализовать поиск, добавление и удаление информации о задолженниках банка; клиенты банка группируются по размеру долга.
24	Реализовать поиск, добавление и удаление информации о программных продуктах; все программные продукты сгруппированы по группам.
25	Реализовать поиск, добавление и удаление информации о товарах предприятия; все товары сгруппированы по ценовым диапазонам.

7.7. Содержание отчета и его форма

Отчет по лабораторной работе должен содержать:

1. Номер и название лабораторной работы; задачи лабораторной работы.
3. Ответы на контрольные вопросы.
4. Диаграмма классов, экранные формы (консольный вывод) и листинг программного кода с комментариями, показывающие порядок выполнения лабораторной работы, и результаты, полученные в ходе её выполнения.

Отчет о выполнении лабораторной работы, подписанный студентом, сдается преподавателю.

7.8. Контрольные вопросы

1. Перечислите классы-коллекции и опишите их характеристики.
2. Какие классы относятся к классам-утилитам?
3. Опишите назначение и методы класса Hashtable.
4. Почему при рассмотрении классов-коллекций обязательно изучают интерфейсы коллекций?
5. Какие классы и методы используются для взаимодействия с внешней средой (операционной системой) приложения?
6. Перечислите и дайте характеристику основным подходам, применяемым для создания и манипулирования коллекциями в Java.
7. Чем отличается ArrayList от LinkedList?

8. Предложите эффективный алгоритм удаления нескольких рядом стоящих элементов из середины списка ArrayList.
9. Как устроена HashMap?
10. В чем отличия TreeSet и HashSet?
11. Для чего используется переопределение hashCode() и equals()? Как это переопределение влияет на работу с коллекциями?

7.9. Список литературы

Для выполнения лабораторной работы, при подготовке к защите, а также для ответа на контрольные вопросы рекомендуется использовать следующие источники: [6-8].

ЛАБОРАТОРНАЯ РАБОТА 8. ГРАФИЧЕСКИЙ ИНТЕРФЕЙС ПОЛЬЗОВАТЕЛЯ. ОСНОВЫ ОБРАБОТКИ СОБЫТИЙ

8.1. Цель и содержание

Цель лабораторной работы: научиться разрабатывать приложения с графическим интерфейсом на языке Java; научиться программировать с использованием технологии WPF.

Задачи лабораторной работы:

- изучить возможности библиотек AWT и Swing;
- изучить архитектуру оконного приложения WPF и Swing;
- научиться проектировать интерфейс приложения с использованием простых элементов управления Swing и WPF;
- научиться использовать контейнеры компоновки в рамках технологий Swing и WPF;
- изучить сходства и различия технологий построения приложений с графическим пользовательским интерфейсом в Java и .NET Framework;
- научиться разрабатывать графические приложения для решения учебных задач.

8.2. Формируемые компетенции

ОПК-4; ОПК-5; ОПК-6.

8.3. Теоретическая часть

8.3.1 Основные технологии и библиотеки Java SE.

Объектно-ориентированное программирование как технология претерпевала развитие параллельно с развитием технологий взаимодействия компьютеров с пользователями. Появление графического интерфейса

пользователя (ГИП) существенно усложнило разработку прикладного ПО и вывело на первый план именно ООП-парадигму, которая наиболее полно учитывает потребности разработчиков графических приложений.

В большинстве высокоуровневых языков разработки приложений с ГИП реализованы библиотеки для построения интерфейса пользователя. В языке C# используются технологии Windows Forms и Windows Presentation Foundation. В Java существуют библиотеки AWT и Swing.

Стандартная поставка Java SE включает в себя библиотеку классов, обеспечивающих создание графического интерфейса пользователя (ГИП) (Graphical User Interface, GUI). Эта графическая библиотека получила название JFC (Java Foundation Classes). В библиотеке JFC можно выделить шесть основных частей:

1) AWT (Abstract Window Toolkit) – базовая библиотека классов с несколько устарелым набором «тяжелых» (heavyweight) графических компонентов, расположенная в пакете `java.awt` и его подпакетах.

2) Swing – библиотека «легких» (lightweight) графических компонентов, дополняющая и во многом заменяющая библиотеку AWT. Занимает почти двадцать пакетов с префиксом `javax.swing`;

3) Java 2D – часть библиотеки AWT, обеспечивающая рисование графики, выбор цвета, вывод изображений и фигурного текста, а также преобразование их перед выводом.

4) DnD (Drag and Drop) – библиотека классов, позволяющих перемещать объекты из одного компонента в другой с помощью буфера обмена (clipboard). Классы этой библиотеки помещены в пакеты `java.awt.datatransfer` и `java.awt.dnd`;

5) Input Method Framework – классы для создания новых методов ввода/вывода. Они занимают пакеты `java.awt.im` и `java.awt.im.spi`;

6) Accessibility – библиотека классов для взаимодействия с нестандартными устройствами ввода/вывода: клавиатурой Брайля, световым пером и др. Она расположена в пакете `javax.accessibility`.

Интерфейс программирования графики (Java Graphics API) представлен двумя технологиями в Java: AWT и Swing.

Возможности технологии AWT разнообразны – AWT состоит из 12 пакетов. В большинстве приложений используются пакеты `java.awt` и `java.awt.event`. Это наиболее часто используемые пакеты.

Пакета `java.awt` содержит:

1) Классы, представляющие компоненты ГИП (GUI Component Classes). Эти классы представляют такие элементы как кнопки (Button), текстовые поля (TextField), надписи (Label) и пр.

2) Классы, представляющие контейнеры ГИП (GUI Container Classes). Представляют такие элементы как панель (Panel), фрейм (Frame), диалог (Dialog) и пр.

3) Менеджеры расположения элементов (Layout managers). Это элементы `FlowLayout`, `BorderLayout` и `GridLayout`.

4) Вспомогательные классы и графические классы общего назначения (Custom graphics classes). Например классы для представления цвета (Color), шрифтов (Font).

Пакет `java.awt.event`, как видно из названия, содержит классы для работы с системой событий и содержит:

1) Классы событий (Event classes): `ActionEvent`, `MouseEvent`, `KeyEvent` и `WindowEvent`.

2) Интерфейсы слушателей событий (Event Listener Interfaces): `ActionListener`, `MouseListener`, `KeyListener` и `WindowListener`.

3) Классы адаптеров слушателей событий (Event Listener Adapter classes): `MouseAdapter`, `KeyAdapter` и `WindowAdapter`.

AWT обеспечивает интерфейс для разработки графических приложений, независимый от платформы и устройства. Эти приложения запускаются на всех ОС.

В AWT выделяют два типа GUI элементов: компоненты (Component) и контейнеры (Container). На рис. 8.1 представлена диаграмма классов AWT, на которой разделены эти две группы элементов.

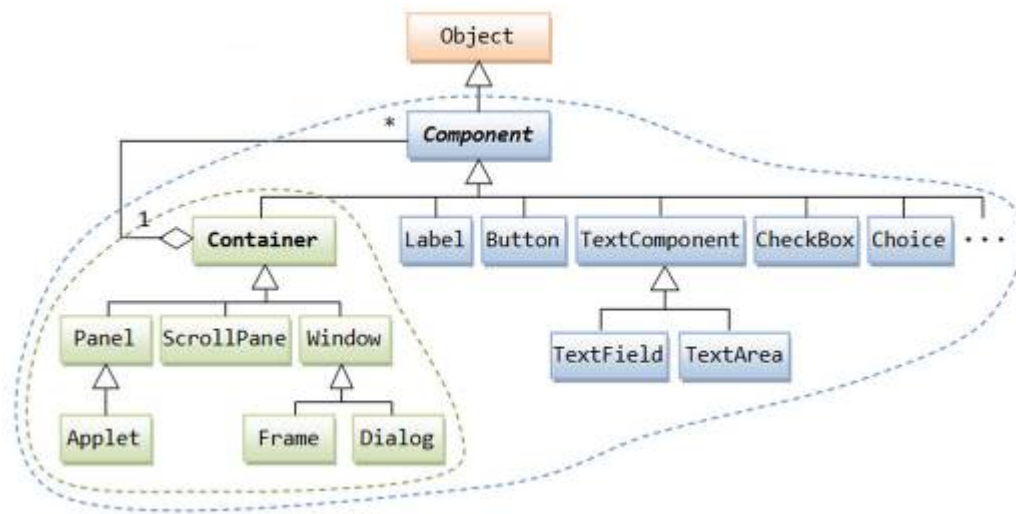


Рисунок 8.1 – Диаграмма классов AWT.

Понимание принципов функционирования библиотеки AWT необходимо для программирования на Java с использованием любой другой библиотеки ГИП, включая Swing. Но детальное описание AWT выходит за рамки данного курса. В данной лабораторной работе акцент сделан на разработке с использованием Swing.

Технология разработки ГИП Swing основана на AWT. На рис. 8.2 представлена диаграмма классов, отражающая преемственность двух технологий.

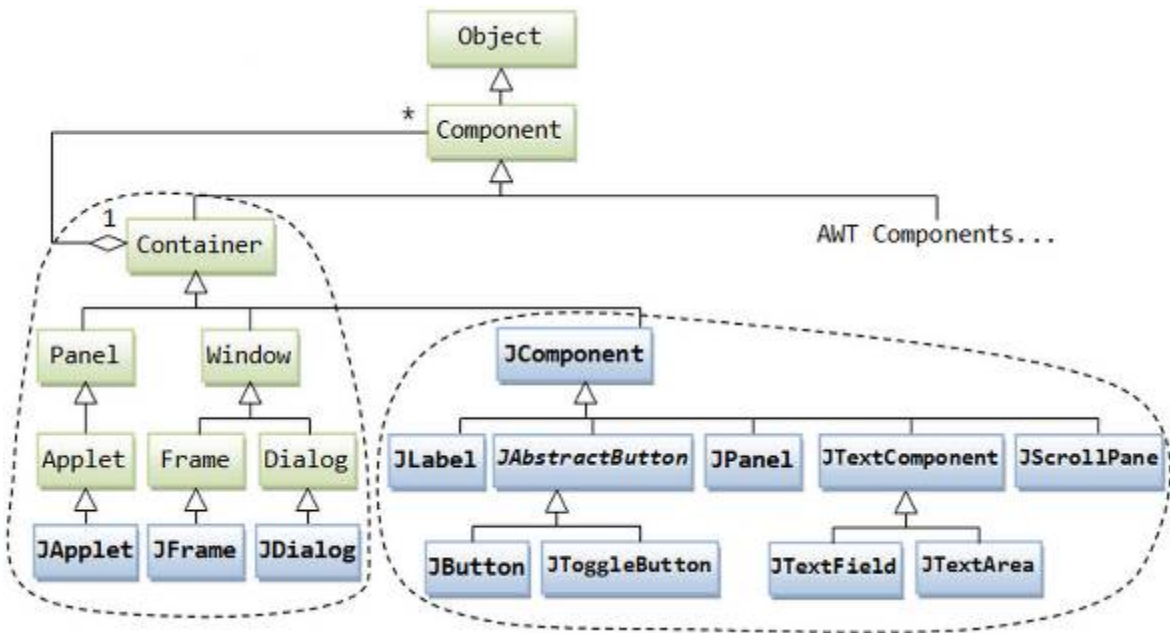


Рисунок 8.2 – Связь AWT и Swing.

Особенности библиотеки Swing:

- 1) Swing написан на «чистом» Java (pure Java).
- 2) Компоненты Swing являются «легковесными». Тогда как компоненты AWT – «тяжеловесные». Это означает, что компоненты Swing вообще не нуждаются в поддержке операционной системы и поэтому гораздо более стабильны и быстры.
- 3) Компоненты Swing поддерживают подключаемые вид и поведение (pluggable look-and-feel). Такое поведение компонентов реализовано на основе архитектурного подхода MVC.
- 4) В компоненты Swing встроена поддержка всплывающих подсказок, рамок, клавиатурных действий, средств для пользователей с ограниченными возможностями.
- 5) Компоненты Swing – это компоненты JavaBeans (модель визуального программирования, основанная на компонентах).
- 6) Приложения Swing используют классы AWT для обработки событий (пакет `java.awt.event`). Swing добавляет новые классы в пакет `javax.swing.event`, но они используются редко.

7) Приложения Swing используют классы AWT для управления размещением компонентов (FlowLayout и BorderLayout пакета java.awt). При этом добавлены новые менеджеры размещения, такие как Springs, Struts, и BoxLayout (пакет javax.swing).

8) В Swing реализована двойная буферизация и оптимизирован механизм прорисовки компонентов.

9) В Swing реализована поддержка много документального интерфейса (JLayeredPane и JInternalFrame).

10) В Swing существенно расширен список поддерживаемых компонентов.

На рис. 8.3 представлена иерархия классов Swing. На рисунке представлены наиболее часто используемые компоненты.

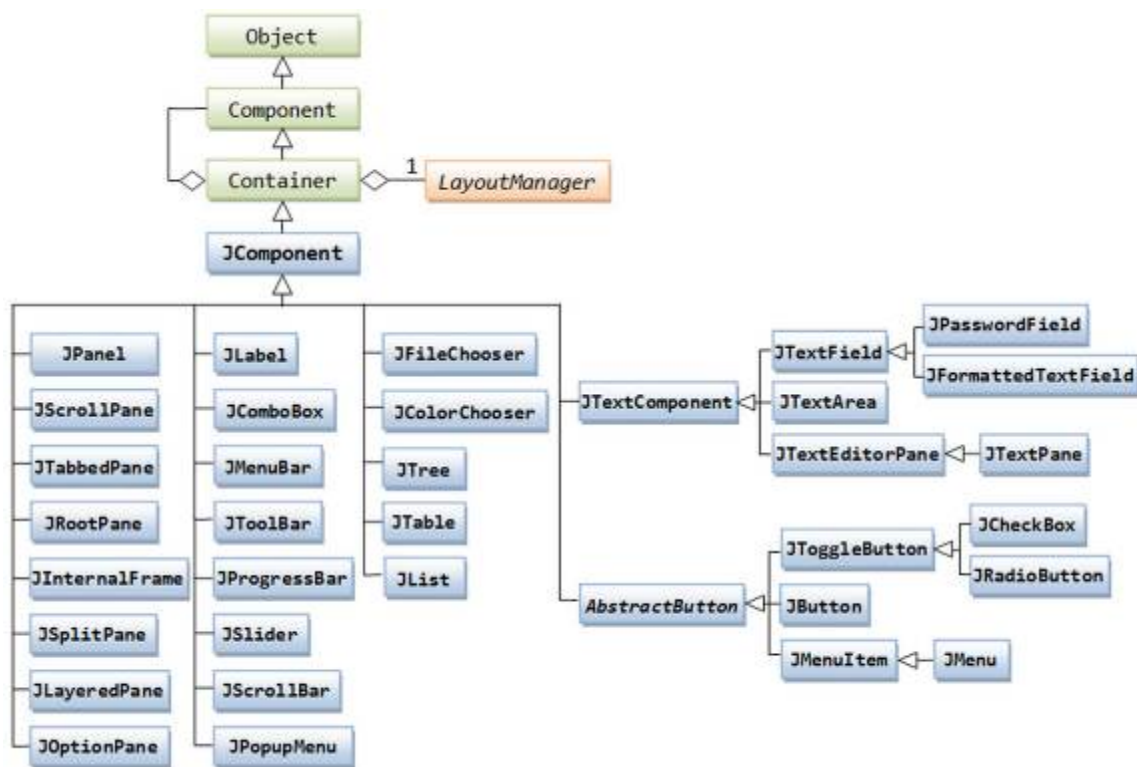


Рисунок 8.3 – Диаграмма классов Swing.

Описание и принципы работы с каждым из представленных компонентов представлены в документации Java. Перед рассмотрением отдельных компонентов пользовательского интерфейса, необходимо освоить работу с главным элементом графического пользовательского интерфейса – окном.

8.3.2 Контейнеры высшего уровня Swing.

Контейнеры высшего уровня (top level containers) представляют собой окна операционной системы, в которых программист может размещать элементы пользовательского интерфейса. К контейнерам высшего уровня относятся окна `JFrame` и `JWindow`, диалоговое окно `JDialog`, а также апплет `JApplet` (не является окном, но предназначен для построения интерфейса в браузере).

Для добавления элементов в окно (например, добавление кнопки на `JFrame`) логично было бы использовать метод `add()`, но контейнеры высшего уровня не позволяют добавлять компоненты напрямую. Для выполнения подобных задач применяется специальная панель `JRootPane`, которая расположена внутри контейнера высшего уровня. Эта панель называется корневой панелью (root panel). Следует понимать, что корневая панель не является отдельным компонентом – она представляет собой набор слоев (панелей), каждый из которых отвечает за определенный функционал. `JRootPane` содержит следующие слои (рис. 8.4):

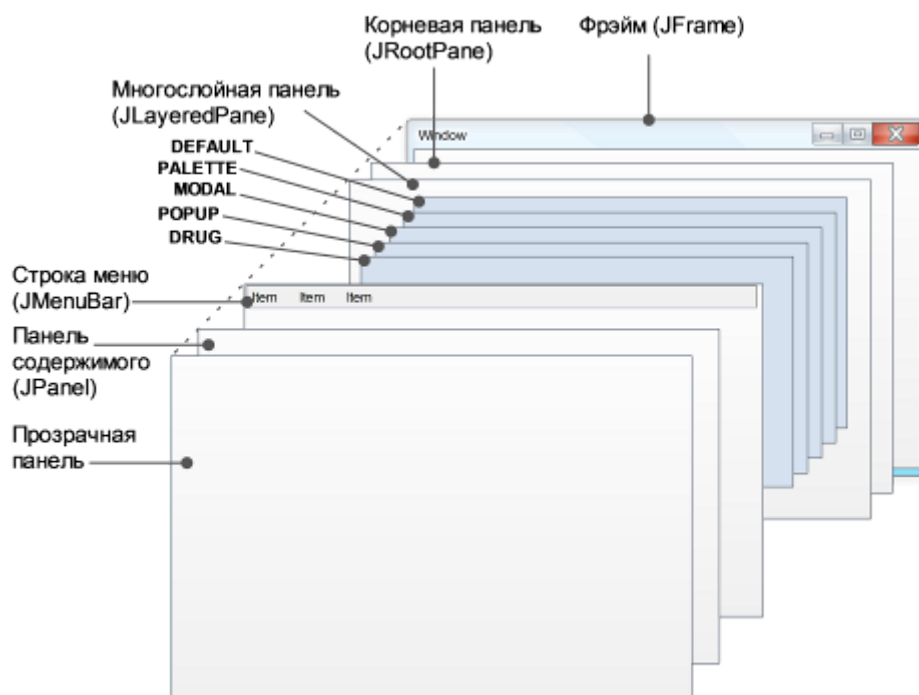


Рисунок 8.4 – Структура `JFrame`.

1. Многослойная панель `JLayeredPane`. Представляет собой совокупность слоев, в которых располагаются визуальные компоненты `Swing`. Каждый слой имеет свой идентификатор и программист может создавать произвольное количество слоев и располагать в них свои элементы. Существуют стандартные (предопределенные слои), номера которых заданы идентификаторами, представленными в таблице 8.2.

Таблица 8.2 – Слои `JLayeredPanel`.

Идентификатор слоя	Описание слоя
<code>DEFAULT_LAYER</code>	Этот слой используется для всех обычных компонентов, которые программист добавляет в контейнер. В нем же располагаются внутренние окна многодокументных приложений.
<code>PALETTE_LAYER</code>	Слой предназначен для размещения так называемых палитр, или окон с набором инструментов, которые обычно перекрывают остальные элементы интерфейса. Создавать такие окна позволяет панель <code>JDesktopPane</code> , которая размещает их как раз в этом слое.
<code>MODAL_LAYER</code>	Судя по названию, разработчики планировали использовать этот слой для размещения легковесных модальных диалоговых окон. Однако такие диалоговые окна пока не реализованы. Этот слой в <code>Swing</code> в настоящее время не используется.
<code>POPUP_LAYER</code>	Наиболее часто используемый слой, служащий для размещения всплывающих меню и подсказок.
<code>DRAG_LAYER</code>	Самый верхний в стопке спой. Предназначен для обслуживания операций перетаскивания (<code>drag and drop</code>).

2. Строка меню (`menu bar`). Строка меню также размещается в многослойной панели, в том же специальном слое `FRAME_CONTENT_LAYER`, и занимает небольшое пространство в верхней части окна, в длину равное размеру окна, ширина используемого пространства при этом зависит от самой строки меню и содержащихся в ней элементов. Корневая панель следит за тем, чтобы панель содержимого и строка меню не

перекрывались. Если строка меню в контейнере высшего уровня не требуется, корневая панель использует все пространство окна для размещения панели содержимого.

3. Панель содержимого (content pane) – часть корневой панели, служащая для размещения компонентов пользовательского интерфейса программы. Она занимает большую часть пространства многослойной панели (за исключением того места, что занимает строка меню). Чтобы панель содержимого не закрывала добавляемые впоследствии в окно компоненты, многослойная панель размещает ее в специальном очень низком слое с названием. `FRAME_CONTENT_LAYER`, с номером -30000. Именно в панель содержимого следует добавлять все компоненты вашего пользовательского интерфейса, это основное отличие контейнеров высшего уровня Swing от их аналогов из AWT. Попытка добавить компоненты напрямую в контейнер (JFrame) высшего уровня Swing приведет к ошибке, так что программисту Java следует запомнить важный вызов:

```
getContentPane().add(Компонент);
```

Панель содержимого – это экземпляр панели JPanel.

5. Прозрачная панель (glass pane). Прозрачная панель используется в приложениях достаточно редко, поэтому по умолчанию корневая панель делает ее невидимой. Данный слой может использоваться для организации автоматизированного тестирования интерфейса приложения, а также для реализации различных анимационных эффектов.

Подробные описания каждого слоя заслуживают отдельного обсуждения и выходит за рамки данной лабораторной работы.

8.3.3 Классы Swing для представления окон.

Окно с рамкой JFrame унаследовано от класса JWindow (в рамках данного курса не рассматривается) и представляет собой наиболее часто используемое в приложениях окно. Окна JFrame обладают рамкой (которая позволяет пользователям легко изменять их размер), заголовком с названием приложения

(хотя можно оставлять этот заголовок пустым), иногда системным меню (позволяющим проводить манипуляции с этим окном) и кнопками для управления окном (чаще всего для его закрытия и свертывания). Именно класс `JFrame` применяется в подавляющем большинстве приложений для размещения компонентов пользовательского интерфейса.

Рассмотрим пример исходного кода, демонстрирующий создание окна:

```

1 package lesson8.Theory;
2
3 import javax.swing.JFrame;
4
5 public class JFrameExample extends JFrame{
6
7     public JFrameExample(){
8         // задание заголовка
9         super("Пример JFrame");
10        // что делать, если окно закрывается
11        setDefaultCloseOperation(EXIT_ON_CLOSE);
12        // иконка для окна
13        setIconImage(getToolkit().getImage("1.png"));
14        // установление размера
15        setSize(300, 100);
16        // показ окна
17        setVisible(true);
18    }
19
20    public static void main(String[] args) {
21        new JFrameExample();
22    }
23
24 }

```

Рисунок 8.5 – Исходный код приложения с ГИП.

На рис. 8.6 показан результат работы программы.

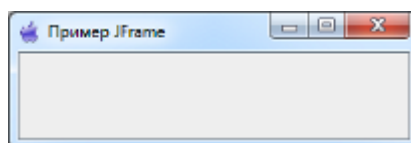


Рисунок 8.6 – Окно приложения, демонстрирующее базовый функционал `JFrame`.

В таблице 8.1 представлены методы класса `JFrame`, которые необходимо освоить в данной лабораторной работе.

Таблица 8.1 – Методы класса `JFrame`.

Метод	Описание
<code>JFrame()</code>	Конструктор. Создает окно, которое является

	невидимым.
JFrame(String title)	Конструктор. Создает окно с заданным заголовком. По-умолчанию окно является невидимым.
setTitle	Метод устанавливает заголовок окна.
setSize(int width, int height)	Устанавливает ширину и высоту окна.
setDefaultCloseOperation(int operation)	Устанавливает действие, которое выполнится при закрытии окна. Запоминать численные значения параметра operation не требуется – для них существуют символьные определения в классе JFrame: DO_NOTHING_ON_CLOSE – не производит никаких действий; HIDE_ON_CLOSE – просто скрывает окно; DISPOSE_ON_CLOSE – скрывает и удаляет окно из памяти, приложение продолжает выполняться; EXIT_ON_CLOSE – после закрытия окна полностью завершает приложение.
setLocationRelativeTo(Component c)	Устанавливает относительную позицию окна (относительно компонента c). Если передано значение null, то окно выравнивается по центру экрана.
void setVisible(boolean b)	Метод показывает (параметр true) или скрывает (false) форму на экране.
getTitle()	Возвращает строку – заголовок формы.

Программисту доступен еще один класс для создания фрейма верхнего уровня – JDialog. Диалоговые окна довольно часто используются в приложениях для получения информации, не имеющей прямого отношения к основному окну, например, для установки параметров, вывода важной вспомогательной информации, получения дополнительной информации от пользователя.

Диалоговые окна могут быть модальными (modal) – они запрещают всем остальным окнам приложения получать информацию от пользователя, пока тот не закончит работу с модальным диалоговым окном. Модальные диалоговые окна полезны в тех ситуациях, когда информация, получаемая от пользователя, коренным образом меняет работу приложения и поэтому до ее получения

продолжение работы невозможно. Внешний вид диалоговых окон повторяет окна с рамкой `JFrame`, но обычно у них меньше элементов управления (чаще всего, имеется только кнопка закрытия окна) и отсутствует системное меню. Как правило, диалоговые окна всегда располагаются поверх основного окна приложения, напоминая о том, что необходимо завершить ввод информации.

Принципы работы с диалоговыми окнами `JDialog` схожи с основными операциями класса `JFrame`. При работе с диалоговыми окнами следует помнить, что они бывают модальными и немодальными. Рассмотрим пример:

```

1 package lesson8.Theory;
2 import java.awt.Dialog;
3 import javax.swing.JDialog;
4
5 public class JDialogExample extends JDialog {
6
7     public JDialogExample(Dialog parent, String caption, boolean isModal){
8
9         super(parent, caption, isModal);
10        setDefaultCloseOperation(DISPOSE_ON_CLOSE);
11        setSize(200, 100);
12        setVisible(true);
13    }
14
15    public static void main(String[] args) {
16        JDialog diag1 = new JDialogExample(null, "Немодальный диалог", false);
17        new JDialogExample(diag1, "Модальный диалог", true);
18    }
19 }

```

Рисунок 8.7 – Исходный код создания диалоговых окон.

Таким образом, очевидно, что работа с классами `JDialog` и `JFrame` отличается не значительно.

8.3.4 Расположение элементов управления в приложениях Java. Менеджеры расположения Swing.

Поддержка менеджеров расположения встроена в базовый класс всех контейнеров `java.awt.Container`. Задать и получить расположение элементов можно методами `setLayout()` и `getLayout()`. В библиотеки существуют стандартные менеджеры расположения (семь различных классов), но если

программист хочет расположить компоненты особым образом, ему необходимо реализовать интерфейс `LayoutManager`.

Стандартные менеджеры расположения: `FlowLayout`, `GridLayout`, `BorderLayout`, `GridBagLayout`, `CardLayout`, `SpringLayout`, `BoxLayout`. Рассмотрим наиболее простые менеджеры.

Полярное расположение `BorderLayout`. Позволяет создать простую структуру окна. При этом окно разбивается на четыре полярных области и одну область, заполняющую центр окна (рис. 8.8).

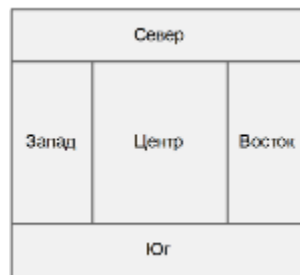


Рисунок 8.8 – Схема расположения областей менеджера `BorderLayout`.

Добавление компонента в контейнер производится методом `add()` с дополнительным параметром, указывающим на место расположения:

1. Значение `BorderLayout.NORTH` или строка «North» – компонент располагается вдоль верхней (северной) границы окна и растягивается на всю его ширину (панель инструментов).

2. Значение `BorderLayout.SOUTH` или строка «South» – компонент располагается вдоль нижней (южной) границы и растягивается на всю ширину окна (строка состояния).

3. Значение `BorderLayout.WEST` или строка «West» – компонент располагается вдоль левой (западной) границы окна и растягивается на всю его высоту, однако при этом учитываются размеры северных и южных компонентов (они имеют приоритет).

4. Значение `BorderLayout.EAST` или строка «East» – компонент располагается вдоль правой (восточной) границы окна.

5. Значение BorderLayout.CENTER или строка «Center» – компонент помещается в центр окна, занимая максимально возможное пространство.

Пример использования расположения BorderLayout представлен на рис. 8.9.

```

7 public class BorderLayoutExample extends JFrame {
8
9     public static void main(String[] args) {
10         new BorderLayoutExample();
11     }
12
13     public BorderLayoutExample(){
14         super("Пример BorderLayout");
15         setSize(400, 300);
16         setDefaultCloseOperation(EXIT_ON_CLOSE);
17         // Получаем панель содержимого
18         Container con = getContentPane();
19         // con.setLayout(new BorderLayout());
20         // Добавляем компоненты
21         con.add(new JButton("Кнопка Север"), "North");
22         con.add(new JButton("Кнопка Юг"), "South");
23         con.add(new JLabel("Надпись Запад"), BorderLayout.WEST);
24         con.add(new JTextField("Текст Восток"), BorderLayout.EAST);
25         con.add(new JEditorPane(JEditorPane.DEFAULT_KEYMAP,
26             "Текстовый редактор"),
27             BorderLayout.CENTER);
28         setIconImage(getToolkit().getImage("1.png"));
29         setVisible(true);
30     }
31 }
32 }

```

Рисунок 8.9 – Листинг с использованием менеджера BorderLayout.

На представленном фрагменте необходимо обратить внимание на строку 18, в которой получается объект панели содержимого. Именно на этом объекте вызываются методы add() в строках 21-25.

В строке 19 вызов метода setLayout() закомментирован. Этот метод устанавливает конкретный менеджер расположения, и все элементы, добавляемые на панель содержимого, будут расставлены в соответствии с данным менеджером расположения. В данном случае этот метод можно не использовать, так как BorderLayout – это менеджер расположения по умолчанию.

В строках 21-25 используется метод `add()`, в качестве второго параметра которого передается характер расположения элемента. Расположение можно определить как строку: «NORTH», «West» и т.д. Но можно использовать предопределенные значения: `BorderLayout.NORTH`, `BorderLayout.WEST`. Очевидны преимущества второго подхода.

Результат выполнения программы представлен на рис. 8.10.

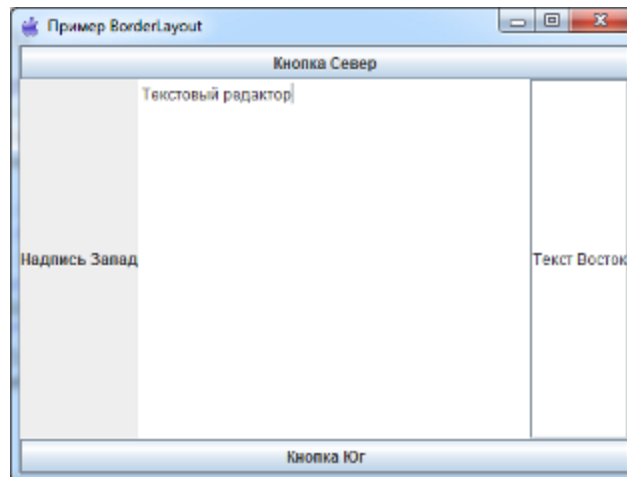


Рисунок 8.10 – Приложение с использованием BorderLayout.

Последовательное расположение FlowLayout. Менеджер задает последовательное расположение компонентов, «стопкой» (вертикально или горизонтально). Класс `FlowLayout` имеет три перегруженные версии:

- без параметров (расположение по умолчанию);
- с заданием выравнивания `FlowLayout(int align)`, где переменная `align` может принимать значения `FlowLayout.LEFT`, `FlowLayout.RIGHT`, `FlowLayout.CENTER`, `FlowLayout.LEADING`, или `FlowLayout.TRAILING`.
- с выравниваем и заданием расстояния между элементами `FlowLayout(int align, int hdist, int vdist)`, где `hdist` и `vdist` – расстояние между элементами, помещаемыми в контейнер, по горизонтали и вертикали соответственно.

Пример использования `FlowLayout` показан на рис. 8.11.

```

1 package lesson8.Theory;
2 import java.awt.Container;
3 import java.awt.FlowLayout;
4 import javax.swing.*;
5
6 public class FlowLayoutExample extends JFrame {
7     public static void main(String[] args) {
8         new FlowLayoutExample();
9     }
10
11     public FlowLayoutExample(){
12         super("Пример FlowLayout");
13         setSize(400, 300);
14         setDefaultCloseOperation(EXIT_ON_CLOSE);
15         // Получаем панель содержимого
16         Container con = getContentPane();
17         // Устанавливаем менеджер расположения
18         con.setLayout(new FlowLayout());
19         // Добавляем компоненты
20         con.add(new JButton("Кнопка Север"));
21         con.add(new JButton("Кнопка Юг"), "South");
22         con.add(new JLabel("Надпись Запад"));
23         con.add(new JTextField("Текст Восток"));
24         con.add(new JEditorPane(JEditorPane.DEFAULT_KEYMAP,
25                                 "Текстовый редактор"));
26         setIconImage(getToolkit().getImage("1.png"));
27         setVisible(true);
28     }
29 }

```

Рисунок 8.11 – Расположение элементов с использованием менеджера FlowLayout.

На рис. 8.12 показан результат работы программы. Следует обратить внимание, что элементы располагаются по горизонтали, а элементы, которым не хватило места в горизонтальном ряду, перемещается на следующую строку.

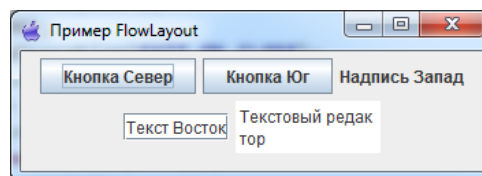


Рисунок 8.12 – Приложение с использованием менеджера FlowLayout.

Табличное расположение GridLayout. Менеджер расположения разделяет контейнер на ячейки, в которые затем помещаются элементы управления. Все ячейки таблицы имеют одинаковый размер, равный максимальному размеру добавляемого элемента.

Конструкторы менеджера расположения:

- конструктор без параметров `GridLayout()` – устанавливает один столбец и одну строку;
- конструктор `GridLayout(int rows, int cols)` с указанием количества строк (`rows`) и количества столбцов (`cols`);
- конструктор `GridLayout(int rows, int cols, int hdist, int vdist)` – задает количество строк (`rows`) и столбцов (`cols`), а также горизонтальное (`hdist`) и вертикальное (`vdist`) расстояния между элементами.

Если вместо `rows` или `cols` передано значение 0, то предполагается, что возможно неограниченное количество строк или столбцов. Сетка будет увеличиваться динамически. Все параметры, задаваемые конструкторами можно изменить с помощью методов объекта `GridLayout`.

Пример использования менеджера показан `GridLayout` на рис. 8.13.

```

7 public class GridLayoutExample extends JFrame{
8     public GridLayoutExample(){
9         super("Пример GridLayout");
10        setSize(500, 500);
11        setDefaultCloseOperation(EXIT_ON_CLOSE);
12        // Получаем панель содержимого
13        Container con = getContentPane();
14        // Устанавливаем менеджер расположения
15        con.setLayout(new GridLayout(10, 10, 1, 1));
16        // Добавляем компоненты
17        for (int i = 0; i < 10; i++)
18            for (int j = 0; j < 10; j++)
19                con.add(new JButton(String.valueOf(i)+String.valueOf(j)));
20        setVisible(true);
21    }
22    public static void main(String[] args) {
23        new GridLayoutExample();
24    }
25 }

```

Рисунок 8.13 – Использование менеджера `GridLayout`.

Результат работы программы представлен на рис. 8.14.

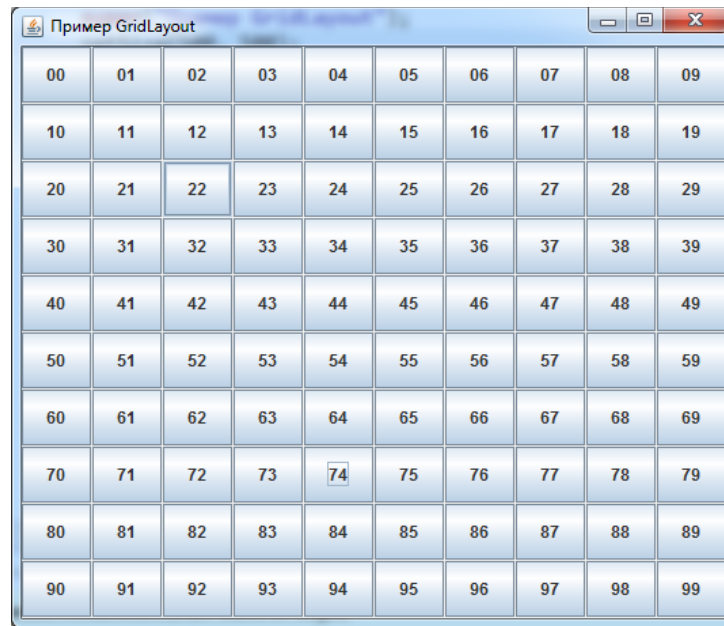


Рисунок 8.14 – Приложение с использованием менеджера GridLayout.

Рассмотрение прочих менеджеров расположения выходит за рамки данной лабораторной работы. Необходимо отметить, что если программист намерен применить абсолютное позиционирование элементов, то следует использовать вызов `setLayout(null)`.

Блочное расположение BoxLayout. Менеджер расположения, обладающий широкими одновременно возможностями GridBagLayout (сложного менеджера с большим количеством настроек) и BorderLayout (простого менеджера, возможности которого были рассмотрены ранее). Это относительно новый компонент (пакет `javax.swing`), который рекомендуется использовать для размещения компонентов в современных приложениях. Конструктору класса `BoxLayout` необходимо указать контейнер, в котором он будет применен, и характер расположения элементов (`Y_AXIS` – по вертикали; `X_AXIS` – по горизонтали). Пример использования `BoxLayout` представлен на рис. 8.15.

```

9 public class BoxLayoutExample extends JFrame {
10     public BoxLayoutExample(int axis){
11         super("Пример BoxLayout");
12         setSize(500, 500);
13         setDefaultCloseOperation(EXIT_ON_CLOSE);
14         // Получаем панель содержимого
15         Container container = getContentPane();
16         // Создаем объект BoxLayout
17         BoxLayout manager = new BoxLayout(container, axis);
18         // Устанавливаем менеджер расположения
19         container.setLayout(manager);
20         // Добавляем компоненты
21         URL imgURL = BoxLayoutExample.class.getResource("B-Favorites.png");
22         JButton btn = new JButton("Favorites", new ImageIcon(imgURL));
23         container.add(btn);
24         imgURL = BoxLayoutExample.class.getResource("B-Logoff.png");
25         btn = new JButton("Logoff", new ImageIcon(imgURL));
26         container.add(btn);
27         imgURL = BoxLayoutExample.class.getResource("B-Previous.png");
28         btn = new JButton("Previous", new ImageIcon(imgURL));
29         container.add(btn);
30         imgURL = BoxLayoutExample.class.getResource("B-Play.png");
31         btn = new JButton("Play", new ImageIcon(imgURL));
32         container.add(btn);
33         imgURL = BoxLayoutExample.class.getResource("B-Next.png");
34         btn = new JButton("Next", new ImageIcon(imgURL));
35         container.add(btn);
36         setVisible(true);
37     }
38     public static void main(String[] args) {
39         new BoxLayoutExample(BoxLayout.X_AXIS);
40         new BoxLayoutExample(BoxLayout.Y_AXIS);
41     }
42 }

```

Рисунок 8.15 – Использование менеджера BoxLayout.

В данном листинге видно, что конструктор BoxLayout отличается от конструкторов менеджеров расположения, рассмотренных ранее. Конструктору в качестве параметра передается контейнер, в котором будет применен данный менеджер.

В результате работы программы создаются две формы с различным расположением элементов (рис. 8.16).



Рисунок 8.16 – Приложение с использованием менеджера VoxLayout.

При создании промышленных приложений менеджеры обычно комбинируют и вкладывают друг в друга.

8.3.5 Контейнер верхнего уровня WPF

Объект `Window` – основной элемент традиционных приложений, в нем отображается их основное содержимое. В Windows Presentation Foundation (WPF) за объектом `Window` скрывается обычное окно Win32. Операционная система не различает окна с WPF- и Win32-содержимым: оформление рисуется одинаково, на панели задач они неотличимы друг от друга и т. д. Оформление (Chrome) – это просто другое название области окна, которая обрамляет клиентскую область и в числе прочего содержит кнопки Minimize (Свернуть), Maximize (Развернуть) и Close (Заккрыть).

Объект `Window` – это абстракция окна Win32 (так же, как класс `Form` в Windows Forms), содержащая ряд простых методов и свойств. Создав объект `Window`, программист может показать его на экране с помощью метода `Show`, скрыть – методом `Hide()` (это то же самое, что присвоить свойству `Visibility` значение `Hidden` или `Collapsed`) и закрыть навсегда – методом `Close()`.

Внешним видом `Window` можно управлять с помощью таких свойств, как `Icon` (иконка приложения), `Title` (интерпретируется как заголовок окна) и

WindowState. Для управления положением на экране служат свойства Left и Top. Более осмысленного поведения можно добиться, присваивая свойству WindowStartupLocation значение CenterScreen или CenterOwner. Короче говоря, с помощью установки свойств можно делать почти все, что принято ожидать от окна. Скажем, если установить для Topmost значение true, то окно будет всегда отображаться поверх других, а если присвоить ShowInTaskbar значение false, то значок окна не будет показываться на панели задач.

Всякий раз, как окно становится активным или неактивным (например, из-за того, что пользователь переключается между разными окнами), возникает событие Activated или Deactivated объекта Window.

На рис. 8.17 представлено определение окна с использованием языка разметки XAML. Данное определение генерируется Visual Studio по умолчанию.

```

1 <Window x:Class="WpfExample.Window1"
2       xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3       xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4       Title="Window1" Height="300" Width="300">
5     <Grid>
6
7     </Grid>
8 </Window>

```

Рисунок 8.17 – Шаблон окна приложения, генерируемого по умолчанию.

Окно может содержать только один дочерний элемент. Для размещения в окне множества дочерних элементов управления в WPF используется подход: сначала в фрейме окна размещается специализированная панель (контейнер компоновки), а затем, в данном контейнере располагаются элементы управления.

8.3.6 Контейнеры компоновки WPF

Для размещения элементов управления в WPF используются специализированные компоненты: контейнеры компоновки. В технологии Windows Forms применяется механизм абсолютного позиционирования

элементов управления: задаются координаты верхнего левого угла элемента (свойства `Top`, `Left`) и его размер (свойства `Width` и `Height`).

В WPF компоновка определяется используемым контейнером. Окно в WPF должно реализовывать следующие принципы:

1. Элементы (такие как элементы управления) не должны иметь явно установленных размеров. Вместо этого они растут, чтобы уместить свое содержимое. Например, кнопка увеличивается при добавлении в нее текста. Можно ограничить элементы управления приемлемыми размерами, устанавливая максимальное и минимальное их значение.

2. Элементы не указывают свою позицию в экранных координатах. Вместо этого они упорядочиваются своим контейнером на основе размера, порядка и (необязательно) другой информации, специфичной для контейнера компоновки. Для добавления пробелов между элементами служит свойство `Margin`.

3. Контейнеры компоновки «разделяют» доступное пространство между своими дочерними элементами. Они пытаются обеспечить для каждого элемента его предпочтительный размер (на основе его содержимого), если только позволяет свободное пространство. Они могут также выделять дополнительное пространство одному или более дочерним элементам.

4. Контейнеры компоновки могут быть вложенными. Типичный пользовательский интерфейс начинается с `Grid` – наиболее развитого контейнера, и содержит другие контейнеры компоновки, которые организуют меньшие группы элементов, такие как текстовые поля с метками, элементы списка, значки в панели инструментов, колонка кнопок и т.д.

Все контейнеры компоновки WPF являются панелями, которые унаследованы от абстрактного класса `System.Windows.Controls.Panel` (рис. 8.18). Класс `Panel` добавляет небольшой набор возможностей всем производным классам.

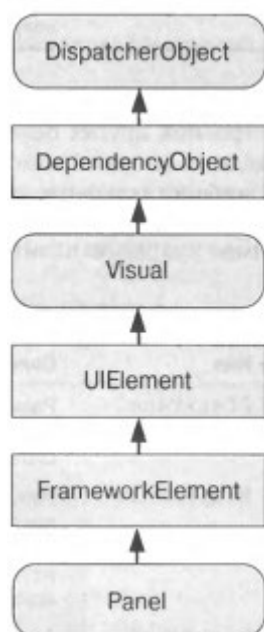


Рисунок 8.18 – Иерархия классов панелей WPF.

Сам по себе базовый класс `Panel` – это абстрактный класс, не допускающий создания объектов. WPF предлагает набор производных от `Panel` классов, которые можно использовать для организации компоновки. Основные контейнеры компоновки перечислены в табл. 8.2. Как и все элементы управления WPF, а также большинство визуальных элементов, эти классы находятся в пространстве имен `System.Windows.Controls`.

Таблица 8.2 – Контейнеры компоновки WPF.

Контейнер	Описание
<code>StackPanel</code>	Размещает элементы в горизонтальном или вертикальном стеке. Этот контейнер компоновки обычно используется в небольших разделах крупного и более сложного окна.
<code>WrapPanel</code>	Размещает элементы в последовательностях строк с переносом. В горизонтальной ориентации <code>WrapPanel</code> располагает элементы в строке слева направо, затем переходит к следующей строке. В вертикальной ориентации <code>WrapPanel</code> располагает элементы сверху вниз, используя дополнительные колонки для дополнения оставшихся элементов.
<code>DockPanel</code>	Выравнивает элементы по краю контейнера.

Grid	Выстраивает элементы в строки и колонки невидимой таблицы. Это один из наиболее гибких и широко используемых контейнеров компоновки.
UniformGrid	Помещает элементы в невидимую таблицу, устанавливая одинаковый размер для всех ячеек. Данный контейнер компоновки используется нечасто.
Canvas	Позволяет элементам позиционироваться абсолютно — по фиксированным координатам. Этот контейнер компоновки более всего похож на традиционный компоновщик Windows Forms, но не предусматривает средств привязки и стыковки. В результате это неподходящий выбор для окон переменного размера.

8.3.7 Контейнер StackPanel

На рис. 8.19 представлен простой пример компоновки элементов с использованием контейнера StackPanel.

```

1 <Window x:Class="WpfExample.Window31"
2     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4     Title="Window31" Height="127" Width="300">
5     <StackPanel>
6         <Button>Кнопка 1</Button>
7         <Button>Кнопка 2</Button>
8         <Button>Кнопка 3</Button>
9         <Button>Кнопка 4</Button>
10    </StackPanel>
11 </Window>

```

Рисунок 8.19 – Использование StackPanel.

Результат подобной компоновки представлен на рис. 8.20. Без дополнительной информации о порядке, координатах и размерах кнопок с помощью контейнера StackPanel удалось расположить элементы вертикально.

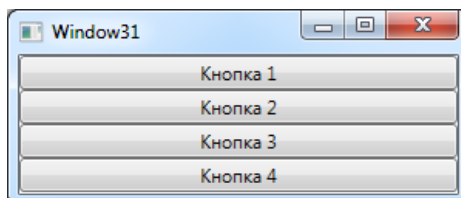


Рисунок 8.20 – Окно с контейнером StackPanel.

По умолчанию панель StackPanel располагает элементы сверху вниз, устанавливая высоту каждого из них такой, которая необходима для отображения его содержимого.

В данном примере это значит, что размер кнопок устанавливается достаточно большим для спокойного размещения текста внутри них. Все элементы растягиваются на полную ширину StackPanel, которая равна ширине окна. Если вы расширите окно, StackPanel также расширится, и кнопки растянутся, чтобы заполнить ее.

StackPanel может также использоваться для упорядочивания элементов в горизонтальном направлении за счет установки свойства Orientation (рис. 8.21):

<StackPanel Orientation=Horizontal>

```

1 <Window x:Class="WpfExample.Window31"
2     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4     Title="Window31" Height="127" Width="300">
5     <StackPanel Orientation="Horizontal">
6         <Button>Кнопка 1</Button>
7         <Button>Кнопка 2</Button>
8         <Button>Кнопка 3</Button>
9         <Button>Кнопка 4</Button>
10    </StackPanel>
11 </Window>

```

Рисунок 8.21 – Использование StackPanel.

Результат представлен на рис. 8.22.

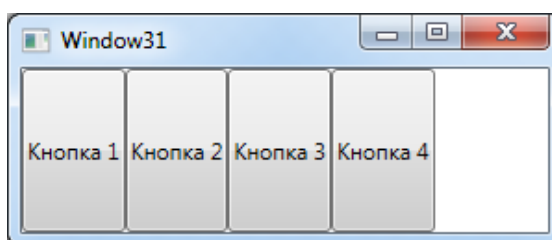


Рисунок 8.22 – Использование горизонтального размещения StackPanel.

Размещение элементов управления определяется контейнером, но элементы управления могут повлиять на этот процесс, явно задавая свойства. Свойства элементов, которые влияют на размещение в контейнере представлены в таблице 8.3.

Таблица 8.3 – Свойства компоновки.

Наименование свойства	Описание
HorizontalAlignment	Определяет позиционирование дочернего элемента внутри контейнера компоновки, когда имеется дополнительное пространство по горизонтали. Доступные значения: Center, Left, Right или Stretch.
VerticalAlignment	Определяет позиционирование дочернего элемента внутри контейнера компоновки, когда имеется дополнительное пространство по вертикали. Доступные значения: Center, Top, Bottom или Stretch.
Margin	Добавляет некоторое пространство вокруг элемента. Свойство Margin – это экземпляр структуры System.Windows.Thickness, с отдельными компонентами для верхней, нижней, левой и правой граней.
MinWidth и MinHeight	Устанавливает минимальные размеры элемента. Если элемент слишком велик, чтобы поместиться в его контейнер компоновки, он будет усечен.
MaxWidth и MaxHeight	Устанавливает максимальные размеры элемента. Если контейнер имеет свободное пространство, элемент не будет увеличен сверх указанных пределов, даже если свойства HorizontalAlignment и VerticalAlignment установлены в Stretch.
Width и Height	Явно устанавливают размеры элемента. Эта установка переопределяет значение Stretch для свойств HorizontalAlignment и VerticalAlignment. Однако данный размер не будет установлен, если выходит за пределы, заданные в MinWidth, MinHeight, MaxWidth и MaxHeight.

8.3.8 Элемент Border

Border не является одной из панелей компоновки, но это удобный элемент, который часто используется.

Класс Border принимает единственную порцию вложенного содержимого (которым часто является панель компоновки) и добавляет фон или рамку вокруг него. Для работы с Border понадобятся свойства, перечисленные в табл. 8.4.

Таблица 8.4 – Свойства класса Border.

Наименование свойства	Описание
Background	С помощью объекта Brush устанавливает фон, который появляется под содержимым. Можно использовать сплошной цвет либо что-нибудь более сложное.
BorderBrush и BorderThickness	Устанавливают цвет рамки, который появляется на границе объекта Border, и толщину рамки. Для отображения рамки потребуется установить оба свойства.
CornerRadius	Позволяет скруглить углы рамки. Чем больше значение CornerRadius, тем заметнее эффект.
Padding	Добавляет пространство между рамкой и содержимым внутри нее.

На рис. 8.23 представлен фрагмент кода XAML, использующий данные свойства.

```

1 <Window x:Class="WpfExample.Window31"
2     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4     Title="Window31" Height="127" Width="300">
5     <Border Background="GreenYellow" BorderBrush="Red"
6         BorderThickness="5" CornerRadius="10" Padding="10">
7         <StackPanel Orientation="Vertical">
8             <Button>Кнопка 1</Button>
9             <Button>Кнопка 2</Button>
10            <Button>Кнопка 3</Button>
11            <Button>Кнопка 4</Button>
12        </StackPanel>
13    </Border>
14
15 </Window>

```

Рисунок 8.23 – Использование свойств Border.

На рис. 8.24 показано результирующее окно.

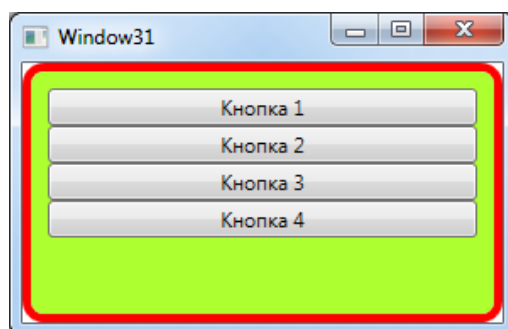


Рисунок 8.24 – Результат применения Border.

8.3.9 Элементы WrapPanel и DockPanel

Панель `WrapPanel` располагает элементы управления в доступном пространстве – по одной строке или колонке за раз. По умолчанию свойство `WrapPanel.Orientation` установлено в `Horizontal`; элементы управления располагаются слева направо, затем – в следующих строках. Установка значения `Vertical` для свойства `WrapPanel.Orientation` приводит к размещению элементов в нескольких колонках.

Подобно `StackPanel`, панель `WrapPanel` действительно предназначена для управления мелкими деталями пользовательского интерфейса, а не компоновкой всего окна. Например, `WrapPanel` можно использовать для удержания вместе кнопок в элементе управления, подобном панели инструментов.

Панель `DockPanel` обеспечивает более интересный вариант компоновки. Эта панель растягивает элементы управления вдоль одной из внешних границ. Простейший способ представить это – вообразить панель инструментов, которая присутствует в верхней части многих Windows-приложений. Такие панели инструментов пристыковываются к верхней части окна. Как и в случае `StackPanel`, пристыкованные элементы должны выбрать один аспект компоновки. Например, если вы пристыковать кнопку к верхней части `DockPanel`, она растянется на всю ширину `DockPanel`, но получит высоту, которая ей понадобится (на основе своего содержимого и свойства `MinHeight`). С другой стороны, если пристыковать кнопку к левой стороне контейнера, ее высота будет растянута для заполнения контейнера, но ширина будет установлена по необходимости.

Дочерние элементы `DockPanel` прикрепляются к одной из сторон панели посредством задания присоединенного свойства `Dock`. Элементы на самом деле не имеют этого свойства – они приобретают его после помещения в контейнер.

На рис. 8.25 показан пример использования `DockPanel`.

```

1 <Window x:Class="WpfExample.DockPanelExample"
2     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4     Title="DockPanelExample" Height="300" Width="300">
5     <DockPanel>
6         <Button Content="кнопка 1" DockPanel.Dock="Top"/>
7         <Button Content="кнопка 2" DockPanel.Dock="Bottom"/>
8         <Button Content="кнопка 3" DockPanel.Dock="Left"/>
9         <Button Content="кнопка 4" DockPanel.Dock="Right"/>
10        <Button Content="кнопка 5"/>
11    </DockPanel>
12 </Window>

```

Рисунок 8.25 – Использование DockPanel.

На рис. 8.26 представлен результат.

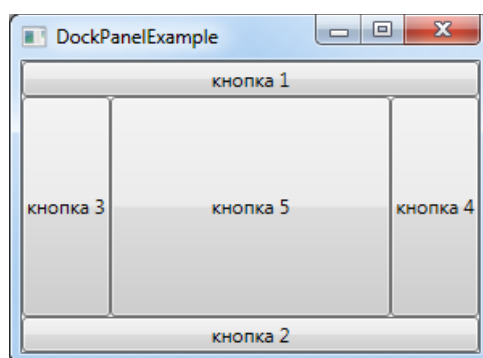


Рисунок 8.26 – Контейнер DockPanel в окне приложения.

8.3.10 Контейнер Grid

Элемент управления Grid – это наиболее мощный контейнер компоновки в WPF. Большая часть того, что можно достичь с помощью других элементов управления компоновкой, также возможно и в Grid. Контейнер Grid является идеальным инструментом для разбиения окна на меньшие области, которыми можно управлять с помощью других панелей.

При добавлении в Visual Studio нового документа XAML для окна автоматически добавляются дескрипторы Grid в качестве контейнера первого уровня, вложенного внутрь корневого элемента Window.

Хотя Grid задуман как невидимый элемент, можно установить свойство Grid.ShowGridLines в true и получить наглядное представление о нем. Это средство на самом деле не предназначено для украшения окна. В действительности это средство для облегчения отладки, которое предназначено

для того, чтобы наглядно показать, как Grid разделяет пространство на отдельные области. Благодаря ему, появляется возможность точно контролировать то, как Grid выбирает ширину колонок и высоту строк.

Создание компоновки на основе Grid – поэтапный процесс. Сначала выбирается необходимое количество колонок и строк. Затем каждому содержащемуся элементу назначается соответствующая строка и колонка, тем самым помещая его в правильное место.

Колонки и строки создаются путем заполнения объектами коллекции Grid.ColumnDefinitions и Grid.RowDefinitions. Например, если необходимо создать две строки и три колонки, то используются следующие дескрипторы:

```

1 <Window x:Class="WpfExample.GridExample"
2     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4     Title="GridExample" Height="300" Width="300">
5     <Grid>
6         <Grid Background="GreenYellow">
7             <Grid.ColumnDefinitions>
8                 <ColumnDefinition/>
9                 <ColumnDefinition/>
10                <ColumnDefinition/>
11            </Grid.ColumnDefinitions>
12            <Grid.RowDefinitions>
13                <RowDefinition/>
14                <RowDefinition/>
15            </Grid.RowDefinitions>
16            <Button Grid.Column="0" Grid.Row="0" Content="Кнопка (0, 0)"/>
17            <Button Grid.Column="1" Grid.Row="1" Content="Кнопка (1, 1)"/>
18            <Button Grid.Column="2" Grid.Row="2" Content="Кнопка (2, 2)"/>
19            <Button Grid.Column="2" Grid.Row="0" Content="Кнопка (2, 0)"/>
20            <Button Grid.Column="0" Grid.Row="2" Content="Кнопка (0, 2)"/>
21        </Grid>
22    </Grid>
23 </Window>
24

```

Рисунок 8.27 – Контейнер Grid.

Результирующее окно представлено на рис. 8.28.

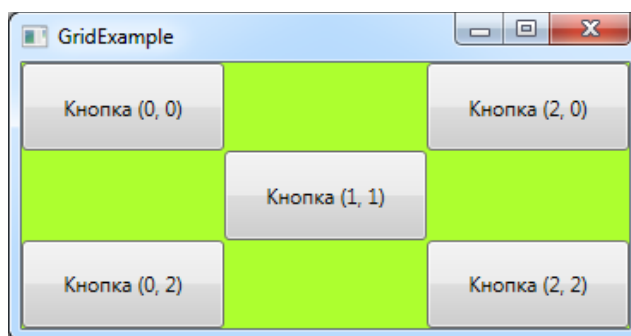


Рисунок 8.28 – Расположение кнопок в контейнере Grid.

Таким образом, в подходах, применяемых в рамках технологий Swing (Java SE) и Windows Presentation Foundation, прослеживается определенное сходство: принципы компоновки элементов; фреймы высшего уровня. На сегодняшний день технология Microsoft превосходит Java SE в плане удобства разработки Windows-приложений, но технология Java SE, в свою очередь, не ограничена одной операционной системой. Также следует учитывать стоимость программных средств. Инструменты Swing имеет перспективы использования в рамках технологии Java EE, то есть библиотека Swing не ограничена только клиентскими приложениями Java SE.

Отделение интерфейсов от кода в технологии WPF привело к появлению дополнительного языка XAML, который также применяется для разработки веб-приложений (технология Silverlight).

8.4. Оборудование и материалы

Для выполнения лабораторной работы рекомендуется использовать персональный компьютер со следующими характеристиками: 64-разрядный (x64) процессор с тактовой частотой 1 ГГц и выше, оперативная память – 1 Гб и выше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система Windows 7 и выше, Java Development Kit 8 и выше; IDE Eclipse for Java SE (or

Java EE) developers. Для выполнения данной лабораторной работы необходима интегрированная среда разработки Microsoft Visual Studio 2012 и выше.

8.5. Указания по технике безопасности

Студенты должны следовать общепринятой технике безопасности для пользователей персональных компьютеров. Не следует самостоятельно производить ремонт технических средств, установку и удаление программного обеспечения. В случае обнаружения неисправностей необходимо сообщить об этом администратору компьютерного класса (обслуживающему персоналу лаборатории).

8.6. Методика и порядок выполнения работы

В рамках выполнения лабораторной работы необходимо освоить два подхода для программирования приложений с использованием GUI.

Первый подход предполагает проектирование фрейма, а затем вызов окна из функции `main()`. Второй подход предполагает определение функции `main()` непосредственно в классе окна (фрейма) как метода данного класса. Такое использование классов экранных форм не реализуется в WPF. Это особенность разработки графических приложений Java SE. Рассмотрим оба подхода.

Реализация первого подхода с функцией `main()`, осуществляющей создание окна приложения и управление им. Для этого необходимо выполнить следующую последовательность действий (приведен пример реализации в среде Eclipse):

1. Создайте проект приложения Java. После создания проекта и, если необходимо, пакетов, необходимо добавить класс главного окна с именем `FrameWithoutMain`. Листинг данного класса представлен на рис. 8.29.

```

3 import java.awt.Container;
4 import java.awt.event.ActionEvent;
5 import java.awt.event.ActionListener;
6
7 import javax.swing.JFrame;
8 import javax.swing.JButton;
9
10 public class FrameWithoutMain extends JFrame {
11
12     private JButton btn;
13
14     public FrameWithoutMain(){
15         super("Окно без функции main");
16         setSize(400, 200);
17         setDefaultCloseOperation(EXIT_ON_CLOSE);
18         Container cont = getContentPane();
19
20         btn = new JButton("Log");
21         btn.addActionListener(new ActionListener() {
22             @Override
23             public void actionPerformed(ActionEvent e) {
24                 System.out.println("Button <Log> pressed!");
25             }
26         });
27
28         cont.add(btn, BorderLayout.CENTER);
29     }
30 }

```

Рисунок 8.29 – Определение фрейма.

2. Создайте класс, содержащий функцию main(). На рис. 8.30 представлен код функции main(), которая определена в классе FrameExample.

```

1 package lesson8.method;
2
3 public class FrameExample {
4
5     public static void main(String[] args) {
6         FrameWithoutMain window = new FrameWithoutMain();
7         window.setVisible(true);
8     }
9
10 }

```

Рисунок 8.30 – Управление окном приложения.

3. Запустите приложение. Рассмотрите представленные варианты исходного кода.

Рассмотрим второй подход применения класса JFrame, когда функция main() определяется непосредственно в классе окна. Для изучения данного

механизма создайте проект Java и реализуйте в нем класс `FrameWithMain` (рис. 8.31).

```

1 package lesson8.method;
2 import java.awt.Container;
3 import java.awt.Dimension;
4 import java.awt.event.ActionEvent;
5 import java.awt.event.ActionListener;
6 import javax.swing.Box;
7 import javax.swing.BoxLayout;
8 import javax.swing.JButton;
9 import javax.swing.JFrame;
10
11 public class FrameWithMain extends JFrame {
12
13     JButton btn;
14
15     public FrameWithMain() {
16         super("Окно с методом main");
17         setSize(400, 200);
18         setDefaultCloseOperation(EXIT_ON_CLOSE);
19         Container cont = getContentPane();
20         BoxLayout box = new BoxLayout(cont, BoxLayout.X_AXIS);
21         cont.setLayout(box);
22
23         btn = new JButton("Log");
24         btn.setPreferredSize(new Dimension(200, 100));
25         btn.addActionListener(new ActionListener() {
26             @Override
27             public void actionPerformed(ActionEvent e) {
28                 System.out.println("Button <Log> pressed!");
29             }
30         });
31
32         cont.add(Box.createHorizontalGlue());
33         cont.add(btn);
34         cont.add(Box.createHorizontalGlue());
35         setLocationRelativeTo(null);
36         setVisible(true);
37     }
38
39     public static void main(String[] args) {
40         new FrameWithMain();
41     }
42 }

```

Рисунок 8.31 – Класс окна, содержащий функцию `main()`.

Данный класс окна, фактически, отображает сам себя. При этом класс инкапсулирует функцию `main()`. В коде на рис. 8.31 обратите внимание на использование контейнера компоновки `BoxLayout`.

Таким образом, можно создавать окна в среде Eclipse. Некоторые среды разработки обладают графическими инструментами создания окон (NetBeans, MS Visual Studio).

Рассмотрим пример решения учебной задачи с использованием классов библиотеки Swing.

Задание. Реализовать приложение для вычисления значения выражения:

$$Z = \begin{cases} 1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \frac{1}{5} - \dots; \\ \sum_{i=1}^N i^2. \end{cases}$$

В данном выражении необходимо предоставить пользователю возможность выбора формулы расчета Z .

Решение. Решение поставленной задачи производится с использованием среды Eclipse.

1. Перед этапом программирования необходимо спроектировать интерфейс приложения. На рис. 8.32 представлен проект экранной формы приложения.

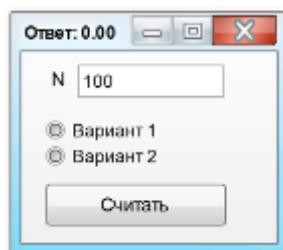


Рисунок 8.32 – Проект экранной формы.

2. После создания схемы реализуем ее в коде. На рис. 8.33 представлен код класса окна приложения.

3. После реализации внешнего вида главного окна приложения, создадим метод, который будет отвечать за вычисление результата. Данный метод – обработчик нажатия кнопки «Считать». Добавьте соответствующий класс слушателя и наполните метод слушателя кодом (рис. 8.34).

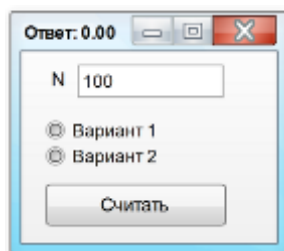


Рисунок 8.33 – Код главного окна приложения.

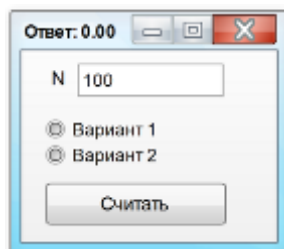


Рисунок 8.34 – Код метода слушателя, реализующий вычисление выражения.

Индивидуальное задание.

Вариант	Выражение для вычисления
1	$Z = \begin{cases} 1 - \frac{X}{2} + \frac{Y^2}{6} - \frac{X^3}{24} + \frac{Y^4}{120} - \dots; \\ \sum_{i=1}^N \sum_{j=1}^R \frac{3+j}{i}. \end{cases}$
2	$Z = \begin{cases} -\frac{1}{2} + \frac{Y}{6} - \frac{X^2}{24} + \frac{Y^3}{120} - \dots; \\ \sum_{i=1}^N \sum_{j=1}^R \frac{i^2 + j}{i^j}. \end{cases}$
3	$Z = \begin{cases} -\frac{1}{1} + \frac{Y}{2} - \frac{X^2}{3} + \frac{Y^3}{4} - \dots; \\ \sum_{i=1}^N \sum_{j=1}^R \frac{j^2 + i}{i^j + j}. \end{cases}$
4	$Z = \begin{cases} -\frac{Y}{2} + \frac{Y^2}{6} - \frac{X^3}{24} + \frac{Y^4}{120} - \dots; \\ \sum_{i=1}^N \sum_{j=1}^R \frac{a \cdot i + b \cdot j}{c \cdot i^j}. \end{cases}$

5	$Z = \begin{cases} -\frac{p^2}{2} + \frac{p^3}{6} - \frac{p^4}{24} + \frac{p^5}{120} - \dots; \\ \sum_{i=1}^N \sum_{j=1}^R \frac{i^2 + b \cdot j}{c \cdot i^3}. \end{cases}$
6	$Z = \begin{cases} -\frac{X}{2} + \frac{p \cdot X^2}{6} - \frac{p^2 \cdot X^3}{24} + \frac{p^3 \cdot X^4}{120} - \dots; \\ \sum_{i=1}^N \sum_{j=1}^R \frac{i^2 + j^2}{i^3 + j^3}. \end{cases}$
7	$Z = \begin{cases} -\frac{X}{2} + \frac{X^2}{6} - \frac{X^3}{24} + \frac{X^4}{120} - \dots; \\ \sum_{i=1}^N \sum_{j=1}^R \frac{i^2}{i^3 + j^3}. \end{cases}$
8	$Z = \begin{cases} -\frac{1}{2} + \frac{X}{6} - \frac{X^2}{24} + \frac{X^3}{120} - \dots; \\ \sum_{i=1}^N \sum_{j=1}^R \frac{a \cdot i}{i^3 + j^3}. \end{cases}$
9	$Z = \begin{cases} -\frac{f}{2} + \frac{X \cdot f^2}{6} - \frac{X^2 \cdot f^3}{24} + \frac{X^3 \cdot f^4}{120} - \dots; \\ \sum_{i=1}^N \sum_{j=1}^R \frac{a \cdot (i+j)}{i \cdot (i+2)}. \end{cases}$
10	$Z = \begin{cases} 1 - \frac{X}{2} + \frac{Y^2}{6} - \frac{X^3}{24} + \frac{Y^4}{120} - \dots; \\ \sum_{i=1}^N \sum_{j=1}^R \frac{3+j}{i}. \end{cases}$
11	$Z = \begin{cases} -\frac{1}{2} + \frac{Y}{6} - \frac{X^2}{24} + \frac{Y^3}{120} - \dots; \\ \sum_{i=1}^N \sum_{j=1}^R \frac{i^2 + j}{a^j}. \end{cases}$
12	$Z = \begin{cases} -\frac{1}{1} + \frac{Y}{2} - \frac{X^2}{3} + \frac{Y^3}{4} - \dots; \\ \sum_{i=1}^N \sum_{j=1}^R \frac{j^2 + i}{i^j + j}. \end{cases}$

13	$Z = \begin{cases} -\frac{Y}{2} + \frac{Y^2}{6} - \frac{X^3}{24} + \frac{Y^4}{120} - \dots; \\ \sum_{i=1}^N \sum_{j=1}^R \frac{a \cdot i + b \cdot j}{c \cdot i^j}. \end{cases}$
14	$Z = \begin{cases} -\frac{p^2}{2} + \frac{p^3}{6} - \frac{p^4}{24} + \frac{p^5}{120} - \dots; \\ \sum_{i=1}^N \sum_{j=1}^R \frac{i^2 + b \cdot j}{c^i \cdot i^3}. \end{cases}$
15	$Z = \begin{cases} -\frac{X}{2} + \frac{p \cdot X^2}{6} - \frac{p^2 \cdot X^3}{24} + \frac{p^3 \cdot X^4}{120} - \dots; \\ \sum_{i=1}^N \sum_{j=1}^R \frac{i^2 + j^2}{i^3 + j^3}. \end{cases}$
16	$Z = \begin{cases} -\frac{X}{2} + \frac{2 \cdot X^2}{6} - \frac{3 \cdot X^3}{24} + \frac{4 \cdot X^4}{120} - \dots; \\ \sum_{i=1}^N \sum_{j=1}^R \frac{(i+1)^2}{i^3 + j^3}. \end{cases}$
17	$Z = \begin{cases} -\frac{1}{2} + \frac{X}{6} - \frac{X^2}{24} + \frac{X^3}{120} - \dots; \\ \sum_{i=1}^N \sum_{j=1}^R \frac{a \cdot i}{i^3 + j^3}. \end{cases}$
18	$Z = \begin{cases} -\frac{f}{2} + \frac{X \cdot f^2}{6} - \frac{X^2 \cdot f^3}{24} + \frac{X^3 \cdot f^4}{120} - \dots; \\ \sum_{i=1}^N \sum_{j=1}^R \frac{a \cdot (i+j)}{i \cdot (i+2)}. \end{cases}$
19	$Z = \begin{cases} -\frac{1}{1} + \frac{Y}{3} - \frac{X^2}{5} + \frac{Y^3}{7} - \dots; \\ \sum_{i=1}^N \sum_{j=1}^R \frac{j^2 + 2}{i^j + 3}. \end{cases}$
20	$Z = \begin{cases} -\frac{Y}{2} + \frac{Y^2}{6} - \frac{X^3}{24} + \frac{Y^4}{120} - \dots; \\ \sum_{i=1}^N \sum_{j=1}^R \frac{a \cdot i + b \cdot j}{c \cdot i^j}. \end{cases}$

21	$Z = \begin{cases} -\frac{p^2}{1 \cdot 2} + \frac{p^3}{2 \cdot 6} - \frac{p^4}{3 \cdot 24} + \frac{p^5}{4 \cdot 120} - \dots; \\ \sum_{i=1}^N \sum_{j=1}^R \frac{i^2 + j}{c^i \cdot i^3}. \end{cases}$
22	$Z = \begin{cases} -\frac{X}{2} + \frac{p \cdot X^2}{6} - \frac{p^2 \cdot X^3}{24} + \frac{p^3 \cdot X^4}{120} - \dots; \\ \sum_{i=1}^N \sum_{j=1}^R \frac{i^2 + j^2}{i^3}. \end{cases}$
23	$Z = \begin{cases} -\frac{3 \cdot Y}{2} + \frac{5 \cdot Y^2}{6} - \frac{7 \cdot X^3}{24} + \frac{9 \cdot Y^4}{120} - \dots; \\ \sum_{i=1}^N \sum_{j=1}^R \frac{i + j}{c \cdot i^j}. \end{cases}$
24	$Z = \begin{cases} -\frac{3 \cdot p^2}{2} + \frac{5 \cdot p^3}{6} - \frac{7 \cdot p^4}{24} + \frac{9 \cdot p^5}{120} - \dots; \\ \sum_{i=1}^N \sum_{j=1}^R \frac{i^2 + b}{c^i \cdot (i+1)^3}. \end{cases}$
25	$Z = \begin{cases} -\frac{X}{2} + \frac{p \cdot X^2}{6} - \frac{p^2 \cdot X^3}{24} + \frac{p^3 \cdot X^4}{120} - \dots; \\ \sum_{i=1}^N \sum_{j=1}^R \frac{(i+j)^2}{i^3 + j^3}. \end{cases}$

8.7 Содержание отчета и его форма

Отчет по лабораторной работе должен содержать:

1. Номер и название лабораторной работы; задачи лабораторной работы.
3. Ответы на контрольные вопросы.
4. Диаграмма классов, экранные формы (консольный вывод) и листинг программного кода с комментариями, показывающие порядок выполнения лабораторной работы, и результаты, полученные в ходе её выполнения.

Отчет о выполнении лабораторной работы, подписанный студентом, сдается преподавателю.

8.8. Контрольные вопросы

1. Назовите основные типы контейнеров компоновки.
2. Назовите основные компоненты JFC.
3. Как соотносятся библиотеки Swing и AWT? В каких пакетах содержатся классы данных библиотек? Назовите часто используемые классы данных библиотек.
4. Назовите основные методы, доступные в классе JFrame.
5. Поясните назначение метода setDefaultCloseOperation. Какие варианты поведения формы можно реализовать с его использованием?
6. Какие возможные подходы к созданию графического приложения на основе Swing можно выделить? Какой подход наиболее удобен?
7. Поясните механизм указания контейнера компоновки для окна приложения. Какие методы и каких объектов программист должен использовать в этом процессе?

8.9 Список литературы

Для выполнения лабораторной работы, при подготовке к защите, а также для ответа на контрольные вопросы рекомендуется использовать следующие источники: [3-6].

ЛАБОРАТОРНАЯ РАБОТА 9. СОБЫТИЯ.

9.1. Цель и содержание

Цель лабораторной работы: изучить принципы обработки событий в рамках технологий Java SE и Windows Presentation Foundation.

Задачи лабораторной работы:

- научиться обрабатывать события в приложениях Java SE;
- научиться обрабатывать события в приложениях WPF;
- изучить основные элементы механизма событий Swing;
- научиться использовать механизм маршрутизируемых событий WPF.

9.2. Формируемые компетенции

ОПК-4; ОПК-5; ОПК-6.

9.3. Теоретическая часть

9.3.1 Общие сведения о системе обработки событий Java.

В большинстве современных технологий (.NET Framework, Java) разработки приложений с ГИП используется специализированный механизм – механизм событий. Говорят, что графические приложения – это приложения управляемые событиями.

При рассмотрении событийной модели Java необходимо выделить три группы компонентов:

1. Источники событий. Источниками событий чаще всего выступают различные элементы управления (кнопки, списки и их элементы, текстовые поля, контейнеры, окна и т.д.). Объекты-источники событий способны информировать объекты-слушатели событий (посылать им сообщение или генерировать событие). Объекты-источники посылают информацию об

изменении своего состояния (кнопка нажата, элемент списка выделен, текст в текстовом поле изменен и т.д.).

2. События. Объект-источник должен переслать некоторую информацию объектам-слушателям. Упрощенно можно утверждать, что именно эта пересылаемая информация оформляется в виде события – объекта специального класса, содержащего данные для детального описания возникшего события. Говорят, что сведения о событии инкапсулируются в объекте события.

3. Слушатели событий (подписчики). Это объекты, которые принимают сообщение от источников событий. Для того, чтобы слушатель получил сообщение о возникновении события, он должен быть подписан на «рассылку» событий. В объекте-источнике производится регистрация слушателей (подписчиков). Слушателем может являться любой объект, но на практике указывается такой объект, который может отреагировать на событий надлежащим образом. Такая модель делегирования событий является очень гибкой.

На рис. 9.1 представлен механизм обработки событий в Java (отношение между задействованными классами).

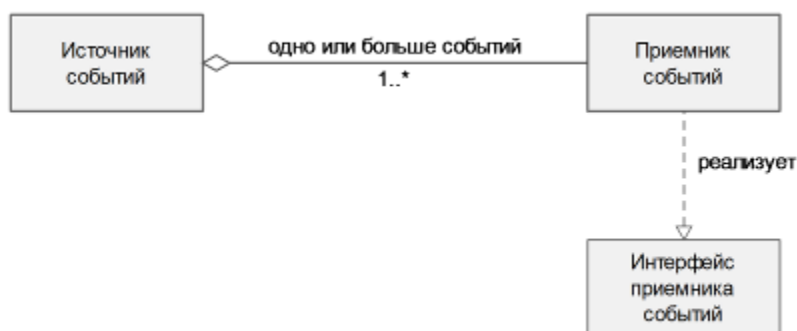


Рисунок 9.1 – Результат работы приложения.

На рис. 9.1 видно, что в механизме присутствуют интерфейсы слушателей событий. Следует учитывать, что существует множество

различных классов событий, при этом все классы событий описываются подклассами, производными от класса `java.util.ObjectEvent`.

На рис. 9.2 представлена диаграмма классов событий.

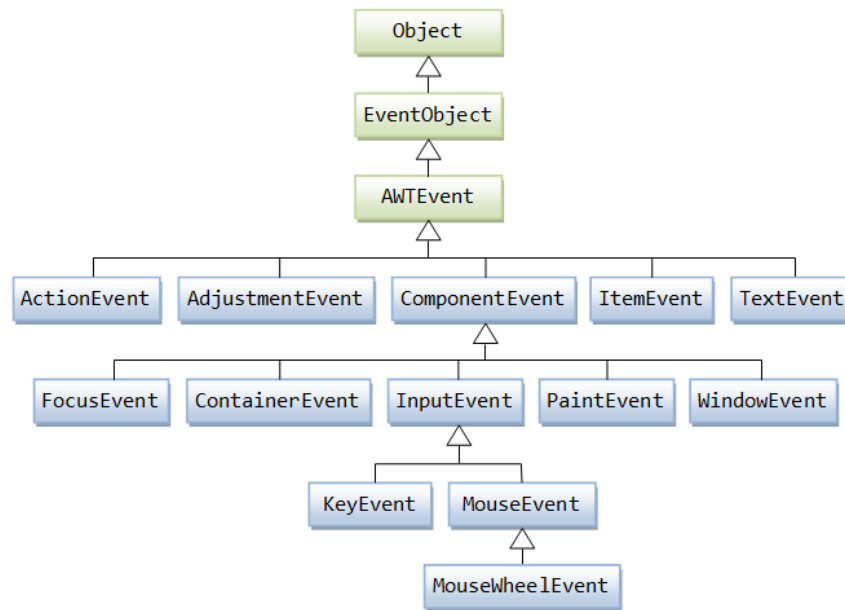


Рисунок 9.2 – Диаграмма классов событий.

Для каждого типа события предусмотрен свой класс – это значит, что объект-слушатель должен реализовать специальный метод (или набор методов) для корректной обработки этого события.

События в Java иногда подразделяют на низкоуровневые (low-level events) и высокоуровневые (high-level events).

К низкоуровневым событиям относят те, что происходят непосредственно в результате действий пользователя: это движения мыши, передача фокуса, выделение элемента списка, нажатия клавиш и т. Все низкоуровневые события унаследованы от класса `java.util.AWTEvent`. Они поступают в Java-программу от операционной системы или от внутренних механизмов виртуальной машины.

Высокоуровневые события происходят в результате изменения состояния компонента. Такие события поступают не от операционной системы, а создаются самим компонентом. Процесс создания события еще называют запуском (fire). Во многих компонентах Swing можно встретить методы вида

fireXXX(). Именно в таких методах создаются объекты с информацией о событиях, которые затем рассылаются объектам-слушателям.

В таблице 9.1 представлены события и методы, которые должен реализовать объект-слушатель события для обработки соответствующего события.

Таблица 9.1 – События Java и методы их обработки.

Класс события	Интерфейс слушателя	Методы слушателя	Источник события
ContainerEvent	ContainerListener	componentAdded(), componentRemoved()	Все контейнеры (наследники класса java.awt.Container)
FocusEvent	FocusListener	focusGained(), focusLost()	Все компоненты
KeyEvent	KeyListener	keyPressed(), keyReleased(), keyTyped()	Все компоненты
MouseEvent	MouseListener	mouseClicked(), mouseEntered(), mouseExited(), mousePressed(), mouseReleased()	Все компоненты
	MouseMotionListener	mouseDragged(), mouseMoved()	Все компоненты
MouseWheelEvent	MouseWheelListener	mouseWheelMoved()	Все компоненты
WindowEvent	WindowListener	windowActivated(), windowClosed(), windowClosing(), windowDeactivated(), windowDeiconified(), windowIconified(), windowOpened()	Окна (наследники java.awt.Window), например, JFrame или JDialog)
ActionEvent	ActionListener	actionPerformed()	Компоненты, у которых есть какое-то стандартное действие.
ChangeEvent	ChangeListener	stateChanged	Некоторые компоненты Swing. Многие модели используют это событие для связи с UI-представителями.
PropertyChangeEvent		propertyChange()	Компоненты JavaBeans (в том числе компоненты Swing).
ItemEvent	ItemListener	itemStateChanged()	Событие выделения элемента списка.

Класс события	Интерфейс слушателя	Методы слушателя	Источник события
ComponentEvent	ComponentListener	componentHidden() componentMoved() componentResized() componentShown()	Событие связано с состоянием (размер, положение и т.п.) компонента.
TextEvent	TextListener	textValueChanged()	Событие связано с изменением текста компонента.

В таблице 9.1 представлены не все классы событий, а наиболее распространенные. Каждый метод интерфейса принимает в качестве параметра соответствующий объект класса события: `keyPressed(KeyEvent)`, `mouseClicked(MouseEvent)`, `windowOpened(WindowEvent)` и т.д. Представленные классы содержатся в пакетах `java.awt.event` и `javax.swing.event`.

При изучении интерфейсов некоторых слушателей становится очевидным, что при их реализации на практике программисту придется реализовать каждый метод интерфейса. При этом может понадобиться только один метод. Для таких случаев в библиотеке Java предусмотрены адаптеры (adapters) – специальные классы, реализующие интерфейс определенного слушателя, а значит все его методы. Методы слушателя адаптер оставляет пустыми, без всякой реализации, что позволяет программисту вместо реализации интерфейса слушателя наследовать адаптер от класса адаптера. При наследовании в дочерний класс переходят все методы базового класса, и остается переопределить только те из них, которые требуется.

9.3.2 Методы организации механизма обработки событий.

Существует несколько стратегий структурной организации программы для использования событий Java SE.

1. Применение вложенных классов. Использование внутренних классов для обработки событий в технологии Java SE широко распространено. Данный подход обладает существенным преимуществом: обеспечение доступа методов внутреннего класса к полям класса, содержащего источник события (чаще всего это окно `JFrame`). Это означает, что при наступлении события обработчику

становятся доступны не только свойства источника события (например, нажатой кнопки), но и дополнительная информация: состояния соседних компонентов пользовательского интерфейса (текст в текстовых полях, выбранные элементы в списке, отмеченные флажки и т.д.). Пример такой реализации представлен на рис. 9.3

```

1  package lr8_Event;
2  import javax.swing.*;
3  import java.awt.*;
4  import java.awt.event.*;
5
6  public class FrameWithInnerListener extends JFrame {
7      // кнопка формы
8      private JButton btnOK;
9
10     public FrameWithInnerListener() {
11         super("Слушатель как внутренний класс");
12         setDefaultCloseOperation(EXIT_ON_CLOSE); // Действие при закрытии
13         setSize(400, 100); // Установка размера окна
14         getContentPane().setLayout(new FlowLayout()); // Пока не учили!!!
15         btnOK = new JButton("Нажмите"); // Создание кнопки
16         btnOK.setSize(29, 100); // Установка размера кнопки
17         getContentPane().add(btnOK); // Пока не учили!!!
18
19         // Добавление объекта-слушателя внутреннего класса:
20         btnOK.addActionListener(new ButtonOKListener());
21         setVisible(true); // Вывод окна на экран
22     }
23
24     // Определение класса-слушателя как внутреннего
25     class ButtonOKListener implements ActionListener{
26         public void actionPerformed(ActionEvent e) {
27             System.out.println("Нажата кнопка \"OK\"");
28         }
29     }
30
31     // Главная функция
32     public static void main(String[] args){
33         new FrameWithInnerListener();
34     }

```

Рисунок 9.3 – Реализация внутреннего класса-слушателя.

2. Применение анонимных классов. Данный метод обработки событий используется крайне редко. Часто такой подход используется для временного кода (во время разработки прототипов). Это самый быстрый путь создания обработчика событий. Класс-слушатель, методы которого используются для обработки события, является анонимным (anonymous class). Такой класс не

создается заранее в отдельном файле *.java. Анонимный класс слушателя объявляется и реализуется в момент создания его объекта. Данный подход показан на рис. 9.4

```

1  package lr8_Event;
2  import javax.swing.*;
3  import java.awt.*;
4  import java.awt.event.*;
5
6  public class FrameWithInnerListener extends JFrame {
7      // кнопка формы
8      private JButton btnOK;
9
10     public FrameWithInnerListener() {
11         super("Слушатель как внутренний класс");
12         setDefaultCloseOperation(EXIT_ON_CLOSE); // Действие при закрытии
13         setSize(400, 100); // Установка размера окна
14         getContentPane().setLayout(new FlowLayout()); // Пока не учили!!!
15         btnOK = new JButton("Нажмите"); // Создание кнопки
16         btnOK.setSize(29, 100); // Установка размера кнопки
17         getContentPane().add(btnOK); // Пока не учили!!!
18
19         // Добавление слушателя с использованием пользовательского метода:
20         btnOK.addActionListener(getListenerForButton());
21         setVisible(true); // Вывод окна на экран
22     }
23
24     // Метод, возвращающий объект-слушатель
25     public ActionListener getListenerForButton() {
26         // Возврат объекта анонимного класса
27         // Класс создается и наполняется в момент необходимости (после new)
28         return new ActionListener() {
29             public void actionPerformed(ActionEvent e) {
30                 System.out.println("Нажата кнопка \"OK\"");
31             }
32         };
33     }
34
35     // Главная функция
36     public static void main(String[] args) {
37         new FrameWithInnerListener();
38     }
39 }

```

Рисунок 9.4 – Обработка события с использованием анонимного класса.

Применять такой подход не рекомендуется ни при решении учебных задач, ни при реализации практических задач. Очевидно, что для каждого элемента окна, для каждого события будет создан свой анонимный класс. Отсюда вытекают два негативных последствия:

1. «Макаронный код». Большое количество кода, который тяжело упорядочивать; некоторые блоки кода реализуют схожие задачи.

2. Отсутствие разделение на уровни. Разработчик должен стремиться проектировать приложения в рамках многоуровневой архитектуры (multi tier architecture), чтобы отделить и уменьшить связность элементов, относящихся к разным функциональным группам: модель данных, представления, логика приложения.

3. Применение фабрики слушателей. Предыдущие подходы обладали существенными недостатками, которые в полной мере проявляются при разработке сложных приложений. Анонимные классы приводят к «разбуханию кода». Использование внутренних классов делает исходный код трудно читаемым. Для повышения структурной гибкости программы используют подход на основе шаблона «фабрика» (factory). Данный метод заключается в использовании специального шаблона: объекты-слушатели возвращаются только методами одного класса – фабрикой слушателей. На рис. 9.5 представлено определение класса-фабрики.

```

1  package lr8_Event;
2  import java.awt.event.*;
3
4  public class FactoryOfListeners {
5      // Метод фабрики слушателей, возвращающий объект-слушатель
6      public ActionListener getListenerForButton() {
7          return new ActionListener() {
8              public void actionPerformed(ActionEvent e) {
9                  System.out.println("Нажата кнопка \"OK\\\"");
10             }
11         };
12     }
13 }

```

Рисунок 9.5 – Класс фабрики (файл FactoryOfListeners.java).

Класс FactoryOfListeners может возвращать любое количество слушателей, может реализовывать определенную логику.

На рис. 9.6 показан класс окна приложения, использующий данный подход.

```

1  package lr8_Event;
2  import javax.swing.*;
3  import java.awt.*;
4  import java.awt.event.*;
5
6  public class FrameWithInnerListener extends JFrame {
7      // кнопка формы
8      private JButton btnOK;
9      private FactoryOfListeners factory = new FactoryOfListeners();
10
11     public FrameWithInnerListener() {
12         super("фабрика слушателей");
13         setDefaultCloseOperation(EXIT_ON_CLOSE); // Действие при закрытии
14         setSize(400, 100); // Установка размера окна
15         getContentPane().setLayout(new FlowLayout()); // Пока не учили!!!
16         btnOK = new JButton("Нажмите"); // Создание кнопки
17         btnOK.setSize(29, 100); // Установка размера кнопки
18         getContentPane().add(btnOK); // Пока не учили!!!
19
20         // Добавление слушателя с использованием фабрики слушателей:
21         btnOK.addActionListener(factory.getListenerForButton());
22         setVisible(true); // Вывод окна на экран
23     }
24
25     // Главная функция
26     public static void main(String[] args){
27         new FrameWithInnerListener();
28     }
29 }

```

Рисунок 9.6 – Обработка события с использованием фабрики слушателей.

Таким образом, все слушатели для окна приложения можно разместить в одном классе.

4. Применение диспетчера (перенаправления) событий. Данный подход предполагает создание класса, внутри которого происходит определение: какой слушатель требуется конкретному источнику (рис. 9.7).

```

1  package lr8_Event;
2  import javax.swing.*;
3  import java.awt.*;
4  import java.awt.event.*;
5
6  public class FrameWithInnerListener extends JFrame {
7      // кнопка формы
8      private JButton btnOK;
9      private JButton btnCancel;
10
11     public FrameWithInnerListener() {
12         super("Диспетчер слушателей");
13         setDefaultCloseOperation(EXIT_ON_CLOSE); // Действие при закрытии
14         setSize(400, 100); // Установка размера окна
15         getContentPane().setLayout(new FlowLayout()); // Пока не учили!!!
16         btnOK = new JButton("OK"); // Создание кнопки OK
17         btnCancel = new JButton("Cancel"); // Создание кнопки Cancel
18         getContentPane().add(btnOK); // Пока не учили!!!
19         getContentPane().add(btnCancel); // Пока не учили!!!
20         // Добавление слушателей с использованием диспетчера слушателей:
21         DispatcherOfListeners dispatcher = new DispatcherOfListeners();
22         btnOK.addActionListener(dispatcher);
23         btnCancel.addActionListener(dispatcher);
24         setVisible(true); // Вывод окна на экран
25     }
26
27     class DispatcherOfListeners implements ActionListener{
28         public void actionPerformed(ActionEvent e) {
29             if (e.getSource() == btnOK) getListenerForOK();
30             if (e.getSource() == btnCancel) getListenerForCancel();
31         }
32     }
33
34     private void getListenerForOK() {
35         System.out.println("Нажата кнопка \"OK\"");
36     }
37
38     private void getListenerForCancel() {
39         System.out.println("Нажата кнопка \"Cancel\"");
40     }
41
42     // Главная функция
43     public static void main(String[] args){
44         new FrameWithInnerListener();
45     }
46 }

```

Рисунок 9.7 – Обработка события с использованием диспетчера.

Использование диспетчера позволяет не создавать отдельный класс слушателя для каждого события, а использовать один класс диспетчера (DispatcherOfListeners) и для каждого события реализовывать только метод – обработчик события (методы getListenerForOk и getListenerForCancel). В данном примере класс диспетчера объявлен как внутренний, но он может быть реализован как независимый класс.

На рис. 9.8 показан результат работы программы.

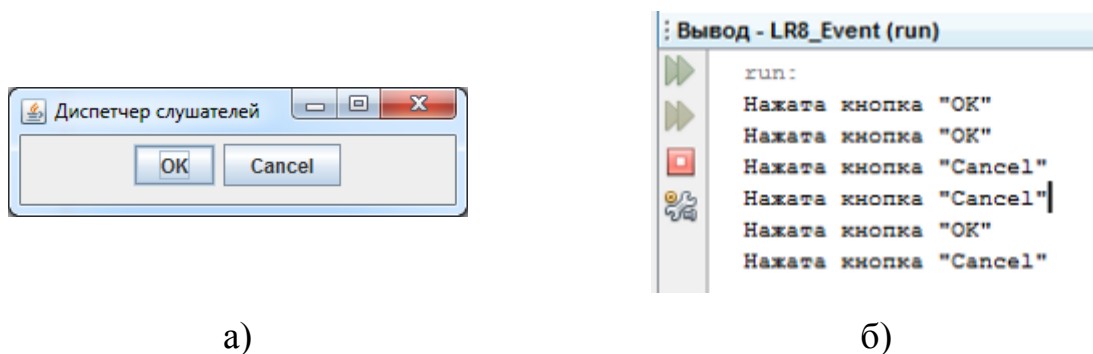


Рисунок 9.8 – Программа обработки событий с использованием диспетчера:
а) окно программы; б) вывод в консоль посредством обработчиков событий.

9.3.3 Модель событий WPF

Программист, работающий в .NET, знаком с понятием события: это сообщение, которое посылается объектом (например, элементом WPF) для уведомления кода о том, что произошло что-то важное. WPF дополняет модель событий .NET новой концепцией маршрутизации событий.

Маршрутизация позволяет событию возникать в одном элементе, а генерироваться в другом: например, щелчок на кнопке панели инструментов генерируется в панели инструментов, а затем в содержащем эту панель окне, и только тогда передается на обработку коду.

Маршрутизация событий дает возможность писать лаконичный и простой код, который может обрабатывать события в наиболее удобном для этого месте. Она необходима также для работы с моделью содержимого WPF, позволяющей создавать простые элементы (например, кнопки) из десятков отдельных ингредиентов, каждый из которых имеет свой собственный набор событий.

Создать обработчик события (на примере кнопки) можно несколькими способами:

1. Добавить соответствующий атрибут события в разметку XAML (рис. 9.9).

```

1 <Window x:Class="WpfExample.EventExample"
2       xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3       xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4       Title="EventExample" Height="300" Width="300">
5     <Grid>
6       <Button Width="150" Height="150">
7         <Border BorderThickness="2" BorderBrush="Black">
8           <Image Source="Images/1.png" Stretch="Fill" Width="100" Height="100"
9             MouseLeave="Image_OnMouseLeave"
10            MouseEnter="Image_OnMouseEnter"/>
11         </Border>
12       </Button>
13     </Grid>
14 </Window>

```

Рисунок 9.9 – Добавление обработчика в разметке XAML.

В данном случае добавлены два обработчика: для события попадания мыши в область рисунка, и для события покидания указателем мыши пределов рисунка.

2. Событие можно соединить и с кодом. Вот эквивалент приведенного на рис. 9.10 кода разметки XAML:

```

20 public partial class EventExample : Window
21 {
22     0 references
23     public EventExample()
24     {
25         InitializeComponent();
26         Image.MouseEnter += Image_OnMouseEnter;
27         Image.MouseLeave += Image_OnMouseLeave;
28     }

```

Рисунок 9.10 – Добавление обработчика в коде C#.

Если необходимо отсоединить обработчик события, то сделать это можно ТОЛЬКО в коде:

```

40 private void Button_OnClick(object sender, RoutedEventArgs e)
41 {
42     Image.MouseEnter -= Image_OnMouseEnter;
43     Image.MouseLeave -= Image_OnMouseLeave;
44 }

```

Рисунок 9.11 – Отключение обработчика в коде C#.

9.3.4 Маршрутизация событий WPF

Многие элементы управления в WPF являются элементами управления содержимым, которые могут иметь разный тип и разный объем вложенного содержимого. На рис. 9.12 показан подобный элемент:

```

1 <Window x:Class="WpfExample.RoutedEventEx"
2     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4     Title="RoutedEventEx" Height="300" Width="300">
5     <Label HorizontalAlignment="Center">
6         <StackPanel VerticalAlignment="Center">
7             <TextBlock Margin="10">
8                 Пример маршрутизируемого события
9             </TextBlock>
10            <Button Width="150">
11                <Button.Content>
12                    <StackPanel>
13                        <Image Source="Images/1.png" Width="100"/>
14                        <Border>
15                            <TextBlock FontWeight="ExtraBold" FontSize="14">
16                                Кнопка с яблоком
17                            </TextBlock>
18                        </Border>
19                    </StackPanel>
20                </Button.Content>
21            </Button>
22        </StackPanel>
23    </Label>
24 </Window>

```

Рисунок 9.12 – Вложенность элементов управления WPF.

На рис. 9.13 показано дерево элементов WPF для данного окна.

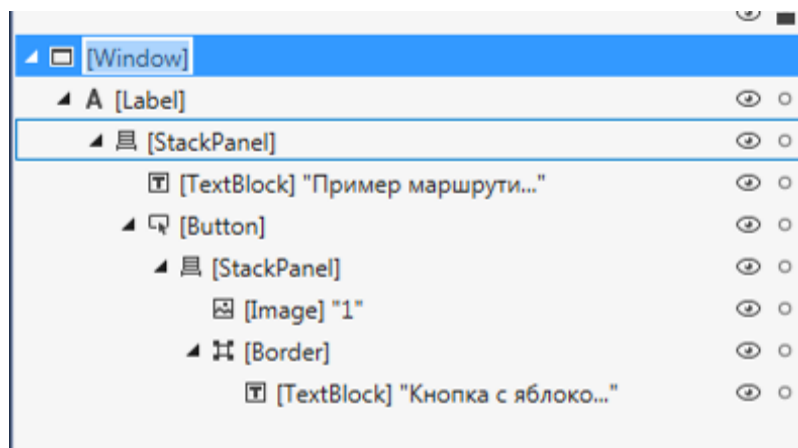


Рисунок 9.13 – Вложенность элементов управления WPF.

На рис. 9.14 представлен результат подобной компоновки элементов управления.

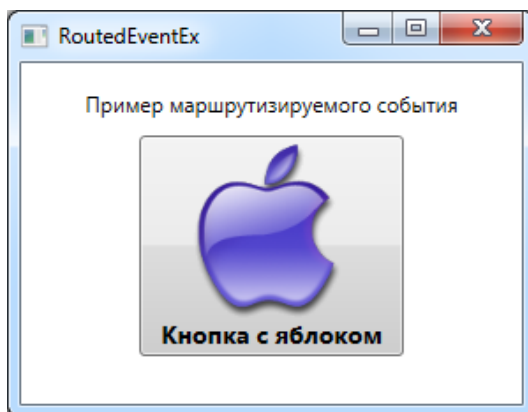


Рисунок 9.14 – Результирующее окно WPF.

Каждый компонент, помещаемый в окно WPF, является наследником класса `UIElement`, включая `Label`, `StackPanel`, `TextBlock` и `Image`. Класс `UIElement` определяет несколько ключевых событий. Например, каждый класс, являющийся потомком `UIElement`, обеспечивает события `MouseUp` и `MouseDown`.

При щелчке мышью на изображении возникнут следующие последствия: сгенерируется событие `Image.MouseDown`, затем событие `Image.MouseUp`. Может возникнуть ситуация, когда щелчки мыши необходимо обрабатывать единообразно, хотя событие может сгенерировать любой элемент управления.

Очевидно, что программист может к событиям `MouseDown` и `MouseUp` каждого элемента можно привязать один и тот же обработчик. Это усложнит код и замедлит работу приложения. Но технология WPF предоставляет удобный механизм обработки подобных событий.

Маршрутизируемые события бывают трех видов:

1. Прямые (direct) события подобны обычным событиям .NET. Они возникают в одном элементе и не передаются в другой. Например, прямым является событие `MouseEnter`, которое возникает, когда указатель мыши наводится на элемент.

2. Пузырьковые (bubbling) события поднимаются по иерархии содержания. Например, пузырьковым событием является `MouseDown`. Оно возникает в элементе, на котором был произведен щелчок, потом передается от этого элемента к родителю, затем к родителю этого родителя, и т.д., пока WPF не достигнет вершины дерева элементов.

3. Туннелируемые (tunneling) события опускаются по иерархии содержания. Они позволяют предварительно просматривать (и, возможно, останавливать) событие, прежде чем оно дойдет до подходящего элемента управления. Например, `PreviewKeyDown` позволяет перехватить нажатие клавиши, сначала на уровне окна, а затем в более специфических контейнерах, вплоть до элемента, содержавшего фокус в момент нажатия клавиши.

Поскольку события `MouseUp` и `MouseDown` являются пузырьковыми событиями, можно определить, что произойдет в примере с составной меткой. При щелчке на изображении с яблоком (рис. 9.15) событие `MouseDown` возникнет в следующем порядке:

1. `Image.MouseDown`.
2. `StackPanel.MouseDown`
3. `Button.MouseDown`.
4. `StackPanel.MouseDown`.
5. `Label.MouseDown`.

После того как событие `MouseDown` возникнет в метке, оно передается следующему элементу управления (в данном случае это окно `Window`). Окно находится на самом верху иерархии содержания и в самом конце в последовательности пузырькового распространения события. Здесь последний шанс обработать пузырьковое событие наподобие `MouseDown`. Если пользователь отпускает кнопку мыши, в такой же последовательности возникает событие `MouseUp`.

Пузырьковые события не обязательно обрабатывать в одном месте: например, ничто не мешает обрабатывать события `MouseDown` и `MouseUp` на

каждом уровне. Однако, как правило, для каждой задачи выбирается наиболее подходящая маршрутизация событий.

Класс `RoutedEventArgs`. Объект данного класса передается в качестве параметра обработчика события. При обработке пузырькового события параметр отправителя содержит ссылку на последнее звено в цепочке. Например, если событие перед обработкой всплывает от изображения до метки, то параметр отправителя будет ссылаться на объект метки.

В некоторых случаях требуется знать, где первоначально произошло событие. Эту информацию, а также другие подробности, можно получить из свойств класса `RoutedEventArgs` (таблица 9.2).

Таблица 9.2 – Свойства класса `RoutedEventArgs`.

Имя свойства	Описание
<code>Source</code>	Указывает, какой объект сгенерировал событие. Если речь идет о событии клавиатуры, то это элемент управления, имевший фокус ввода в момент возникновения события (например, когда была нажата клавиша). В случае события мыши это самый верхний элемент под указателем мыши в момент возникновения события (например, когда был произведен щелчок кнопкой мыши).
<code>OriginalSource</code>	Указывает, какой объект первоначально сгенерировал событие. Как правило, совпадает с <code>Source</code> . Однако в некоторых случаях <code>OriginalSource</code> спускается глубже по дереву объектов, чтобы дойти до внутреннего элемента, являющегося частью элемента более высокого уровня. Например, если вы щелкнете кнопкой мыши близко к границе окна, то получите объект <code>Window</code> в качестве источника события и <code>Border</code> в качестве первоначального источника. Это объясняется тем, что <code>Window</code> состоит из отдельных меньших элементов.
<code>RoutedEvent</code>	Предоставляет объект <code>RoutedEvent</code> для события, сгенерированного вашим обработчиком события (например, статический объект <code>UIElement.MouseUpEvent</code>). Эта информация бывает полезна при обработке разных событий одним и тем же обработчиком.

Handled	Позволяет остановить процесс пузырькового распространения или туннельного распространения события. Если элемента управления заносит в свойство Handled значение true, событие прекращает продвижение и не будет возникать в любых других элементах.
---------	---

Туннелируемые события работают так же, как и пузырьковые, но в обратном направлении. Например, если бы событие `MouseUp` было туннельным (а это не так), то при щелчке на изображении в примере с меткой событие `MouseUp` возникло бы сначала в окне, затем в элементе `Label`, затем в `StackPanel` и так далее до достижения источника `Image`, т.е. изображения кнопки.

Туннелируемые события легко распознать: они начинаются со слова `Preview`. Более того, WPF обычно определяет события попарно. Это означает, что если имеется пузырьковое событие `MouseUp`, то, скорее всего, существует и туннелируемое событие `PreviewMouseUp`. Туннелируемое событие всегда возникает перед пузырьковым событием. На рис. 9.15 показаны направления распространения соответствующих типов событий.



Рисунок 9.15 – Схематичное изображение механизма событий WPF.

События WPF и механизм событий Java SE представляют собой похожие технологии, хотя используются различные терминологии. Следует понимать,

что подписка обработчика события WPF, фактически, представляет собой аналог слушателя Swing.

Еще один аспект, который следует учитывать при разработке приложений для Windows: механизм событий не ограничивается только одной областью – графическими приложениями. События – это универсальная технология, позволяющая организовать взаимодействие произвольных объектов одного или нескольких приложений. В следующих лабораторных работах будут рассмотрены более сложные аспекты программирования приложений с использованием механизма событий.

9.4. Оборудование и материалы

Для выполнения лабораторной работы рекомендуется использовать персональный компьютер со следующими характеристиками: 64-разрядный (x64) процессор с тактовой частотой 1 ГГц и выше, оперативная память – 1 Гб и выше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система Windows 7 и выше, Java Development Kit 8 и выше; IDE Eclipse for Java SE (or Java EE) developers. Для выполнения данной лабораторной работы необходима интегрированная среда разработки Microsoft Visual Studio 2012 и выше.

9.5. Указания по технике безопасности

Студенты должны следовать общепринятой технике безопасности для пользователей персональных компьютеров. Не следует самостоятельно производить ремонт технических средств, установку и удаление программного обеспечения. В случае обнаружения неисправностей необходимо сообщить об этом администратору компьютерного класса (обслуживающему персоналу лаборатории).

9.6. Методика и порядок выполнения работы

1. Создайте приложение Java.
2. Добавьте экранную форму (класс, производный от JFrame).
3. С использованием средств Java решите следующую задачу.

Условие задачи. Написать приложение для вычисления выражения

$$S = \sum_{i=1}^N \sum_{j=1}^K \frac{a^i \cdot x^2 + b^j \cdot y^2}{i \cdot j}.$$

Для каждого неизвестного в правой части реализовать

элемент управления для ввода значения. Вывод значения полученного выражения реализовать в нескольких форматах. Результат вычисления отображается сразу после изменения входных данных.

Решение. Для решения задачи выполните следующие шаги в среде Netbeans IDE.

4. Рассмотрим переменные в правой части, которые пользователь должен вводить в программу, и определим тип элемента управления для каждого: a (JComboBox), b (JSpinner), N (JComboBox), K (JSpinner) и т.д. На рис. 9.16 представлена результирующая форма.

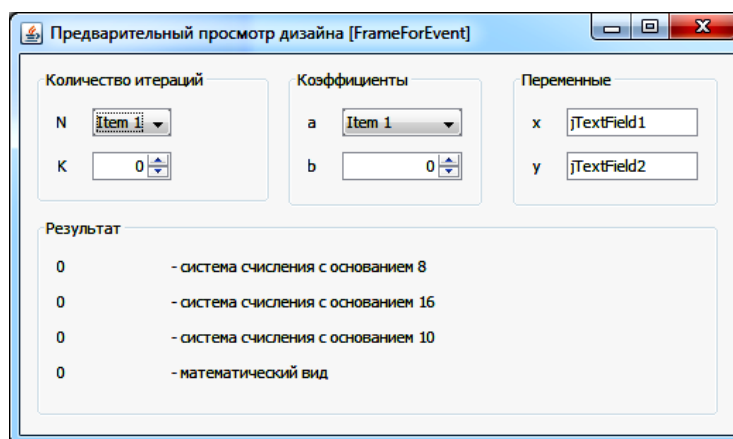


Рисунок 9.16 – Форма приложения для решения задачи.

5. Создадим функцию, которая будет рассчитывать значение выражения по введенным пользователем значениям (рис. 9.17).

```

private void Calc() {
    // считываем значения из элементов
    int N = Integer.parseInt((String)elemN.getSelectedItem());
    int K = (int)elemK.getValue();
    double a = Double.parseDouble((String)elemA.getSelectedItem());
    int b = (int)elemB.getValue();
    double x = Double.parseDouble(elemX.getText());
    double y = Double.parseDouble(elemY.getText());
    double Z = 0;
    // Вычисляем значение выражения
    for (int i = 1; i <= N; i++)
        for (int j = 1; j <= K; j++)
        {
            Z += (Math.pow(a, i) * x * x + Math.pow(b, j) * y * y) /
                (i * j);
        }
    // Выводим результат
    lblRes10.setText(String.format("%12.3f", Z));
    lblRes8.setText(String.format("%o", (int)Z));
    lblRes16.setText(String.format("%x", (int)Z));
    lblResMath.setText(String.format("%5.3e", Z));
}

```

Рисунок 9.17 – Метод расчета выражения и вывода результатов.

6. Создадим обработчик события изменения выбранного значения для элемента elemN (элемент JComboBox для ввода значения N). Для этого щелкните правой кнопкой мыши на элементе JComboBox и выберите пункт «Событие – Item – ItemStateChanged» контекстного меню. В открывшемся обработчике события введите код:

```

private void elemNItemStateChanged(java.awt.event.ItemEvent evt) {
    // TODO add your handling code here:
    Calc();
}

```

Рисунок 9.18 – Обработчик изменения значения N.

7. Добавьте обработчик события stateChanged изменения значения в elemK (элемент типа JSpinner для представления значения K). В обработчик события добавьте вызов метода Calc().

8. Создайте обработчики, реагирующие на изменения значений каждого элемента управления.

9. Запустите приложение, убедитесь в его работоспособности.

10. Спроектируйте форму в соответствии со своим вариантом индивидуального задания. Создайте требуемые обработчики и выведите результат вычисления в разных форматах.

Индивидуальное задание.

1.	$Z = \sum_{i=1}^N \sum_{j=1}^K \frac{p^i \cdot x^3 + f^j \cdot y^2}{i \cdot j}.$	2.	$Z = \sum_{i=1}^N \sum_{j=1}^K \frac{t^i \cdot x^i + f^j \cdot y^j}{t \cdot i \cdot j}.$
3.	$Z = \sum_{i=1}^N \sum_{j=1}^K \frac{p^i \cdot y^j}{i \cdot j}$	4.	$Z = \sum_{i=1}^N \sum_{j=1}^K \frac{\sin(x) \cdot x^i + f^j \cdot y^j}{i \cdot j}$
5.	$Z = \sum_{i=1}^N \sum_{j=1}^K \frac{\sin(x) \cdot x^i + \cos(y) \cdot y^j}{i \cdot j}$	6.	$Z = \sum_{i=1}^N \sum_{j=1}^K \frac{ x^i + \cos(y) \cdot y^j}{i \cdot j}$
7.	$Z = \sum_{i=1}^N \sum_{j=1}^K \frac{i \cdot x^i + j \cdot y^j}{i \cdot j}$	8.	$Z = \sum_{i=1}^N \sum_{j=1}^K \frac{i \cdot \cos(x^i) + j \cdot \sin(y^j)}{i \cdot j}$
9.	$Z = \sum_{i=1}^N \sum_{j=1}^K \frac{\cos(x^i) + j}{i \cdot j}$	10.	$Z = \sum_{i=1}^N \sum_{j=1}^K \frac{\cos(y^i) + j \cdot x}{i \cdot j}$
11.	$Z = \sum_{i=1}^N \sum_{j=1}^K \frac{\sin(y^i) + i \cdot x}{(i+1) \cdot j}$	12.	$Z = \sum_{i=1}^N \sum_{j=1}^K \frac{\sin(y \cdot x) + i \cdot y}{(i+1) \cdot (j+2)}$
13.	$Z = \sum_{i=1}^N \sum_{j=1}^K \frac{2 \cdot x^i + 3 \cdot y^j}{i^2 \cdot j^2}$	14.	$Z = \sum_{i=1}^N \sum_{j=1}^K \frac{i \cdot x^{2 \cdot i} + j \cdot y^{2 \cdot j}}{i^2 \cdot j^2}$
15.	$Z = \sum_{i=1}^N \sum_{j=1}^K \frac{p^{i-1} \cdot x + f^{j-1} \cdot y^2}{i \cdot j}$	16.	$Z = \sum_{i=1}^N \sum_{j=1}^K \frac{a^{i-1} \cdot x^i + f^j \cdot y^j}{(i+1) \cdot j}$
17.	$Z = \sum_{i=1}^N \sum_{j=1}^K \frac{5 \cdot x + 3 \cdot y}{i^2 \cdot j^2}$	18.	$Z = \sum_{i=1}^N \sum_{j=1}^K \frac{\cos(y^i) + j \cdot \sin(x)}{i \cdot (j-1)}$
19.	$Z = \sum_{i=1}^N \sum_{j=1}^K \frac{ x^i + \cos(y) \cdot y^j}{(i+1) \cdot j}$	20.	$Z = \sum_{i=1}^N \sum_{j=1}^K \frac{(p-1)^i \cdot y^j}{i \cdot j}$

9.7. Содержание отчета и его форма

Отчет по лабораторной работе должен содержать:

1. Номер и название лабораторной работы; задачи лабораторной работы.

3. Ответы на контрольные вопросы.

4. Диаграмма классов, экранные формы (консольный вывод) и листинг программного кода с комментариями, показывающие порядок выполнения лабораторной работы, и результаты, полученные в ходе её выполнения.

Отчет о выполнении лабораторной работы, подписанный студентом, сдается преподавателю.

9.8. Контрольные вопросы

1. Какие основные компоненты событийной модели можно выделить в технологии Java? Опишите каждый элемент.

2. Назовите основные пути построения приложения, основанного на событийной модели.

3. Опишите механизм обработки событий с использованием внутреннего класса-слушателя и в случае применения анонимных классов.

4. Какой механизм обработки событий применяется в среде NetBeans по умолчанию при добавлении события в визуальном конструкторе?

5. Опишите механизм обработки событий с использованием фабрики слушателей.

6. Поясните механизм обработки событий с применением диспетчера слушателей.

7. Поясните основные отличия механизмов обработки событий в технологиях Java SE и WPF.

8. Поясните значение следующих терминов: «маршрутизируемое событие», «пузырьковое событие», «туннелируемое событие», «класс-слушатель», «источник события».

9. Какой подход для создания механизма событий необходимо использовать, чтобы реализовать приложение по принципам многоуровневой архитектуры.

9.9. Список литературы

Для выполнения лабораторной работы, при подготовке к защите, а также для ответа на контрольные вопросы рекомендуется использовать следующие источники: [5-7].

ЛАБОРАТОРНАЯ РАБОТА 10. РАЗРАБОТКА ПРИЛОЖЕНИЙ В РАМКАХ ШАБЛОНА MODEL-VIEW-CONTROLLER

10.1. Цель и содержание

Цель лабораторной работы: изучить принципы функционирования элементов управления Swing на основе шаблона Model-View-Controller.

Задачи лабораторной работы:

- научиться использовать модели и представления элементов управления Swing;
- научиться проводить декомпозицию приложения с графическим интерфейсом в рамках шаблона MVC;
- научиться использовать многоуровневую архитектуру для разработки средств автоматизации решения задач.

10.2. Формируемые компетенции

ОПК-4; ОПК-5; ОПК-6.

10.3. Теоретическая часть

10.3.1 Программирование в рамках шаблона MVC на примере кнопки.

Каждый компонент Swing, производный от JComponent, построен по архитектуре подключаемого внешнего вида и поведения (pluggable look and feel architecture). В рамках данной архитектуры выполнено разделение данных компонента (component data) от его модели поведения (component behavior) и внешнего вида (component look). Таким образом, программист может определить особый внешний вид компонента, особенности его поведения, а также подключить компонент к модели данных.

На рис. 10.1 представлена схема компонента Swing. Компонент `JComponent` – это класс, с которым непосредственно взаимодействует приложение. Модель (Model) и представление (UI) скрыты (инкапсулированы) внутри компонента и используются им через внутренние механизмы. В предыдущих лабораторных работах мы работали непосредственно с классом компонента.

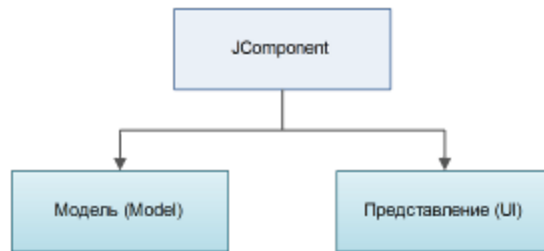


Рисунок 10.1 – Внутреннее представление компонента Swing.

Модель (Model) – класс, который содержит все внутренние атрибуты компонента (нажата кнопка или нет, активна или нет, какой пункт списка выбран и т.д.).

Представление (UI) – обрабатывает события от пользователя, осуществляет рисование требуемого вида компонента. В данном случае в представлении (UI) заключены функции представления (view) и контроллера (controller) стандартного шаблона MVC.

Два рассмотренных класса взаимодействуют друг с другом через механизм событий. Рассмотрим пример использования класса модели кнопки.

На рис. 10.2 представлен листинг программы, демонстрирующий создание четырех кнопок `JButton` и одной кнопки `JToggleButton`. В строке 21 получаем модель (объект класса `ButtonModel`) от кнопки `btn1`.

```

1  package lr10buttonmodel;
2  import javax.swing.*;
3  import java.awt.*;
4  import java.awt.event.*;
5
6  public class LR10ButtonModel extends JFrame {
7      public LR10ButtonModel() {
8          // Настраиваем форму
9          super("Button Model");
10         setSize(200, 200);
11         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
12         setLocationRelativeTo(null);
13         setVisible(true);
14
15         // Определяем выравнивание компонентов
16         Container c = this.getContentPane();
17         c.setLayout(new FlowLayout(FlowLayout.CENTER));
18
19         // Создаем первую кнопку и получаем ее модель
20         JButton btn1 = new JButton("Кнопка 1");
21         ButtonModel model = btn1.getModel();
22
23         // Создаем остальные кнопки на основе модели первой
24         JButton btn2 = new JButton("Кнопка 2");
25         btn2.setModel(model);
26         JButton btn3 = new JButton("Кнопка 3");
27         btn3.setModel(model);
28         JButton btn4 = new JButton("Кнопка 4");
29         btn4.setModel(model);
30
31         // Добавляем кнопки на панель
32         c.add(btn1);
33         c.add(btn2);
34         c.add(btn3);
35         c.add(btn4);
36
37         // Создаем кнопку изменения состояния модели
38         JToggleButton btnOnOff = new JToggleButton("Отключить модель", true);
39         btnOnOff.addItemListener(new ItemListener() {
40             @Override
41             public void itemStateChanged(ItemEvent e) {
42                 if(e.getStateChange() == 1){
43                     btnOnOff.setText("Отключить модель");
44                     model.setEnabled(true);
45                 } else {
46                     btnOnOff.setText("Включить модель");
47                     model.setEnabled(false);
48                 }
49             }
50         });
51         c.add(btnOnOff);
52     }
53
54     public static void main(String[] args) {
55         new LR10ButtonModel();
56     }
57 }

```

Рисунок 10.2 – Применение модели кнопки.

В строках кода 23 – 35 создаются оставшиеся три кнопки и методом `add` все кнопки добавляются на форму. В строке 38 создается кнопка `JToggleButton` с двумя состояниями (нажата или отпущена). Кнопка `btnOnOff` отображает состояние модели: если модель кнопок `JButton` активна, то кнопка `btnOnOff` нажата, иначе – отпущена. Для `btnOnOff` в строках 39 – 50 создается анонимный класс-слушатель `ItemListener`. Метод `itemStateChanged` данного класса переопределяется с целью установки нового текста на управляющую кнопку, а также для отключения модели.

Следует также обратить внимание на класс слушателя для `btnOnOff`: в данном случае не применяется простейший класс `ActionListener`, так как данный класс обрабатывает просто событие нажатия кнопки, а не смены ее состояния. Запустите приложение. Поясните, почему все кнопки отключаются и включаются одновременно. Поясните также, почему при нажатии любой кнопки `JButton` остальные тоже нажимаются.

10.3.2 Создание приложения в рамках архитектуры Model – View – Controller.

Рассмотрим пример приложения, в котором произведено разделение модели, представление и контроллера (то есть спроектированное с использованием архитектурного шаблона MVC).

Задание. Разработать приложение, отражающее процессы поступления товара в магазине. Пользователь приложения формирует редактируемый список товарных групп («Продукты», «Бытовая химия», «Одежда», «Фрукты», «Полиграфия»). По факту поступления товара пользователь формирует запись со следующими полями: «Наименование товара», «Партия», «Товарная группа», «Цена» «Количество». Описание полей представлены в таблице 15.1. В программе должен быть реализован автоматический расчет сумм по каждой партии и товарной группе.

Таблица 15.1 – Поля записи регистра поступления товаров

Поле записи	Описание	Тип данных
Наименование товара	Пользователь заполняет самостоятельно.	String
Партия	Документ (приходная накладная) в соответствии с которым принят товар. Пользователь заполняет самостоятельно.	String
Товарная группа	Пользователь может выбрать только из имеющегося справочника товарных групп.	String
Цена	Пользователь вводит самостоятельно.	float
Количество	Пользователь вводит самостоятельно.	float

Таблица 10.1 – Поля записи регистра поступления товаров.

Поле записи	Описание	Тип данных
Наименование товара	Пользователь заполняет самостоятельно.	String
Партия	Документ (приходная накладная) в соответствии с которым принят товар. Пользователь может ввести самостоятельно или выбрать из уже имеющихся партий.	String
Товарная группа	Пользователь может выбрать только из имеющегося справочника товарных групп.	String
Цена	Пользователь вводит самостоятельно.	float
Количество	Пользователь вводит самостоятельно.	float

Решение. Для решения этой задачи целесообразно применение технологий баз данных (или файлового ввода-вывода), но подобные технологии будут рассмотрены в следующих лабораторных работах. Вся информация не будет сохраняться.

1. Создайте проект приложения Java с именем «LR10MVC» с главным классом.

2. Начнем реализацию задачи с проектирования интерфейса. Создайте новый пакет «view» и добавьте новую форму JFrame с именем «MainFrame» в данный пакет. На рис. 10.3 показан внешний вид формы в режиме проектирования, а также имена переменных программы, ассоциированных с визуальными компонентами.

В соответствии с условием задания у элемента comboDoc (выпадающий список «Партия») необходимо установить свойство editable в значение true. На рис. 10.4 представлено дерево элементов формы.

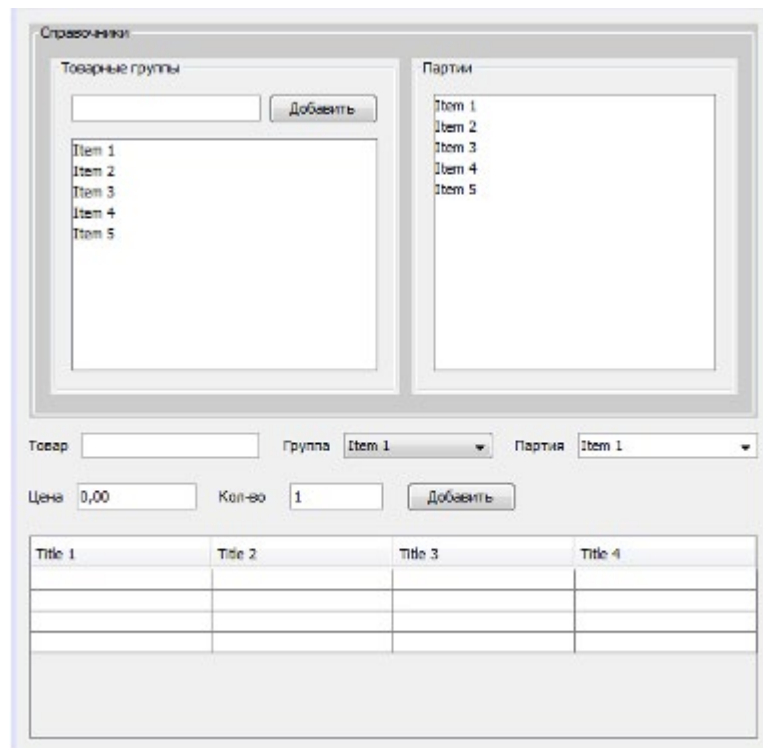


Рисунок 10.3 – Главное окно приложения.

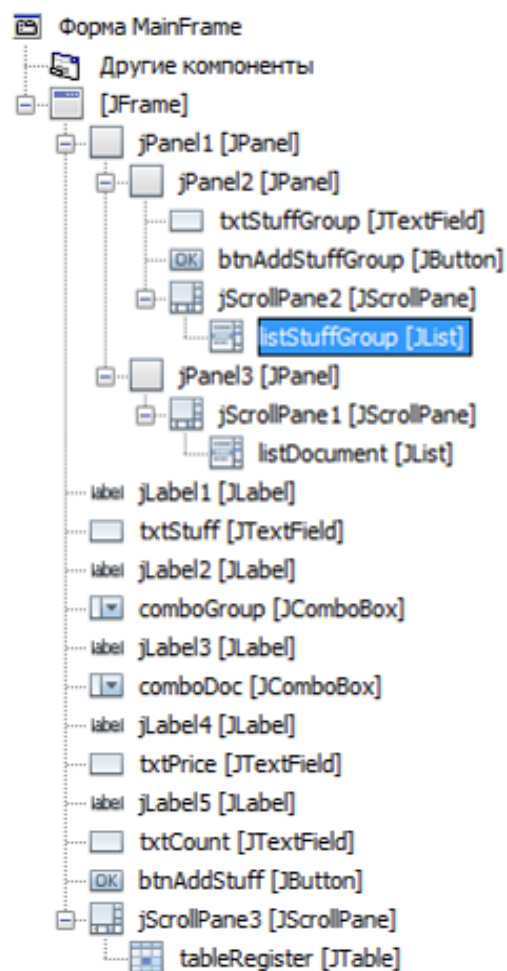


Рисунок 10.4 – Окно «Навигатор».

3. Реализуем классы, которые будут представлять модель данных (data model). Для представления справочников товаров, списка партий, и записей приходного регистра целесообразно использование отдельных классов.

4. Создайте пакет с именем «model»

5. Для представления справочника «Товарный группы» создадим класс StuffGroups в пакете model и реализуем в нем методы инициализации, добавления, удаления и возврата всех записей. На рис. 10.5 представлен листинг данного класса.

```

1  package model;
2  import javax.swing.DefaultListModel;
3
4  public class StuffGroups extends DefaultListModel<String> {
5      // Метод возвращает новый объект StuffGroups
6      public static StuffGroups createNewStuffGroups() {
7          StuffGroups gr = new StuffGroups();
8          return gr;
9      }
10 }
```

Рисунок 10.5 – Класс StuffGroups.

6. Для представления справочника «Партии» добавьте класс Documents в пакет model (рис. 10.6).

```

1  package model;
2  import javax.swing.DefaultListModel;
3
4  public class Documents extends DefaultListModel<String>{
5      // Метод возвращает новый объект Documents
6      public static Documents createNewDocuments() {
7          Documents d = new Documents();
8          return d;
9      }
10 }
```

Рисунок 10.6 – Класс Documents.

7. Для представления приходного регистра предназначен класс Register (рис. 10.7).

```

1  package model;
2  import javax.swing.table.DefaultTableModel;
3
4  public class Register extends DefaultTableModel {
5      public static Register createNewRegister() {
6          Register r = new Register();
7          r.setColumnIdentifiers(
8              new String[] { "Товар", "Группа",
9                           "Партия", "Цена", "Кол-во", "Сумма" });
10         return r;
11     }
12 }

```

Рисунок 10.7 – Класс Register.

8. Классы Register, Documents и StuffGroups объединим в один класс модели MyModel (в пакете model). Листинг класса MyModel представлен на рис. 10.8.

```

1  package model;
2  import javax.swing.DefaultComboBoxModel;
3
4  public class MyModel {
5      // Закрытые данные модели
6      private Register rData;
7      private StuffGroups sData;
8      private Documents dData;
9      private DefaultComboBoxModel sDataCombo, dDataCombo;
10     public MyModel() {
11         // Инициализация данных модели
12         rData = Register.createNewRegister();
13         sData = StuffGroups.createNewStuffGroups();
14         dData = Documents.createNewDocuments();
15         sDataCombo = new DefaultComboBoxModel();
16         dDataCombo = new DefaultComboBoxModel();
17         // Начальное заполнение
18         addNewStuffGroup("Продукты"); addNewStuffGroup("Полиграфия");
19         addNewStuffGroup("Одежда"); addNewStuffGroup("Бытовая химия");
20     }
21     // Открытые данные модели
22     public Register getRegister() { return rData; };
23     public StuffGroups getStuffGroups() { return sData; };
24     public Documents getDocuments() { return dData; };
25     // Модели для полей со списком
26     public DefaultComboBoxModel getDocumentsCombo() { return dDataCombo; };
27     public DefaultComboBoxModel getStuffGroupsCombo() { return sDataCombo; };

```

```

28 // Добавление товарной категории
29 public void addNewStuffGroup(String s){
30     if (!sData.contains(s)){
31         // Добавляем в модель ListBox
32         sData.add(0, s);
33         // Добавляем в модель ComboBox
34         sDataCombo.addElement(s);
35     }
36 }
37 // Добавление партии
38 public void addNewDocument(String s){
39     if (!dData.contains(s)){
40         // Добавляем в модель ListBox
41         dData.add(0, s);
42         // Добавляем в модель ComboBox
43         dDataCombo.addElement(s);
44     }
45 }
46 // Добавление записи
47 public void addNewStuff(String name, String group,
48     String doc, float price, float count){
49     rData.addRow(new String[]{ name, group, doc,
50         String.valueOf(price), String.valueOf(count)});
51 }
52 }

```

Рисунок 10.8 – Класс MyModel.

В данном классе модели данных реализованы только методы добавления данных, а также отсутствует механизм сохранения данных в файлах или в базе данных.

После реализации элементов модели и представления, свяжем эти компоненты контроллером. Контроллер реализуем через механизм событий.

9. Для привязки модели к интерфейсу модифицируем класс MainFrame (рис. 10.9).

```

1  package view;
2  import model.*;
3
4  public class MainFrame extends javax.swing.JFrame {
5
6      private MyModel model;
7
8      public MainFrame(MyModel pModel) {
9          // Установка модели
10         this.model = pModel;
11
12         // Вызов сгенерирован средой разработки
13         initComponents();
14
15         // Задание моделей для компонентов формы
16         if (this.model != null){
17             listStuffGroup.setModel(model.getStuffGroups());
18             comboDoc.setModel(model.getDocumentsCombo());
19             comboGroup.setModel(model.getStuffGroupsCombo());
20             listDocument.setModel(model.getDocuments());
21             tableRegister.setModel(model.getRegister());
22         }
23     }

```

Рисунок 10.9 – Изменения в коде класса MainFrame.

На рис. 10.9 видно, что в класс MainFrame добавлено поле model типа MyModel. В конструкторе класса MainFrame изменения следующие: добавлен параметр конструктора, добавлен код (строки 16 – 21) инициализации модели элементов управления формы.

10. После изменения конструктора необходимо изменить функцию main приложения как показано на рис. 10.10.

```

1  package lr10mvc;
2  import model.*;
3  import view.*;
4
5  public class LR10MVC {
6
7      public static void main(String[] args) {
8          MainFrame view = new MainFrame(new MyModel());
9          view.setVisible(true);
10     }
11 }

```

Рисунок 10.10 – Функция main.

В данном варианте в конструктор главной формы передается новый объект модели.

11. Запустите приложение обратите внимание на привязку списка товарных групп. Исправьте ошибки.

12. Реализуйте обработчики событий нажатия кнопок добавления новой товарной группы и добавления строки в таблицу поступления товаров (рис. 10.11).

```

248 private void btnAddStuffGroupActionPerformed(java.awt.event.ActionEvent evt) {
249     String text = txtStuffGroup.getText();
250     model.addNewStuffGroup(text);
251 }
252
253 private void btnAddStuffActionPerformed(java.awt.event.ActionEvent evt) {
254     // Получаем значения, введенные пользователем
255     // Название товара
256     String stuffName = txtStuff.getText();
257     // Наименование товарной группы
258     String stuffGroup = comboGroup.getSelectedItem().toString();
259     // Партия
260     String doc = comboDoc.getSelectedItem().toString();
261     // Цена
262     float price = Float.parseFloat(txtPrice.getText());
263     // Количество
264     float count = Float.parseFloat(txtCount.getText());
265     // Добавляем данные в модель
266     // в модель таблицы
267     model.addNewStuff(stuffName, stuffGroup, doc, price, count);
268     // в модель списка документов
269     model.addNewDocument(doc);
270 }

```

Рисунок 10.11 – Обработчики событий.

13. Запустите приложение, обратите внимание на синхронное отображение данных в списках, списках выбора и таблице (рис. 10.12).

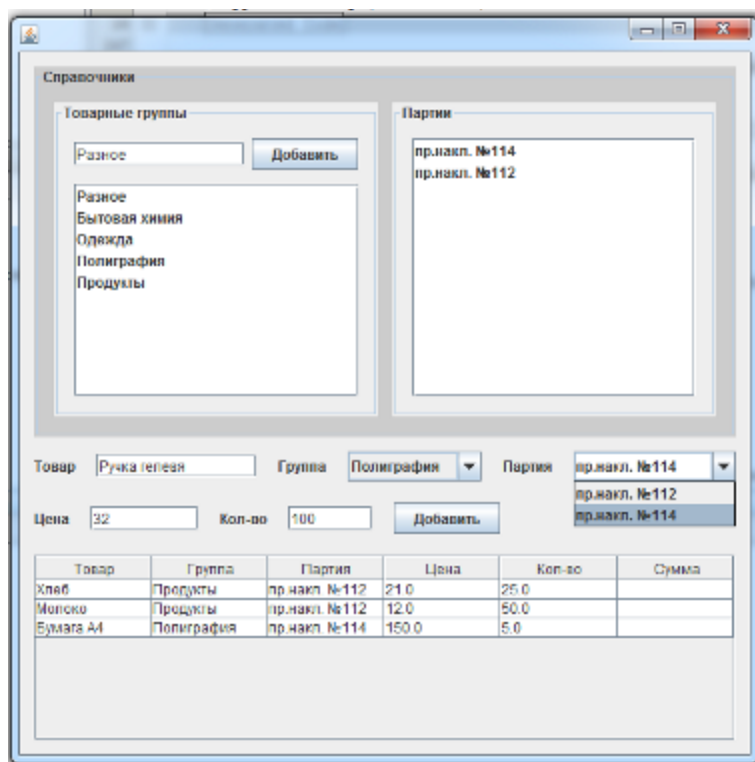


Рисунок 10.12 – Внешний вид работающего приложения.

10.4. Оборудование и материалы

Для выполнения лабораторной работы рекомендуется использовать персональный компьютер со следующими характеристиками: 64-разрядный (x64) процессор с тактовой частотой 1 ГГц и выше, оперативная память – 1 Гб и выше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система Windows 7 и выше, Java Development Kit 8 и выше; IDE Eclipse for Java SE (or Java EE) developers.

10.5. Указания по технике безопасности

Студенты должны следовать общепринятой технике безопасности для пользователей персональных компьютеров. Не следует самостоятельно производить ремонт технических средств, установку и удаление программного обеспечения. В случае обнаружения неисправностей необходимо сообщить об

этом администратору компьютерного класса (обслуживающему персоналу лаборатории).

10.6. Методика и порядок выполнения работы

1. Выполните поэтапную разработку приложения в соответствии с алгоритмом, представленным в подразделе 10.3.2.

2. В представленном алгоритме не рассчитывается сумма поступившей партии (рис. 10.12). Доработайте приложение, чтобы сумма рассчитывалась автоматически.

3. Реализуйте приложение в модели MVC в соответствии с индивидуальным вариантом.

4. При реализации приложения данные можно вводить в форму, то есть не требуется обязательная реализация хранения данных в файле.

Индивидуальное задание.

Вариант	Задание
1	Исходный файл содержит список преподавателей в формате «ФИО, должность, кафедра, заработная плата» (произвольное количество строк): <pre>1 Иванов С.Е. доцент ИСТ 56.00 2 Петров А.А. профессор ИБАС 100.23 3 Новиков П.С. ст. преподаватель ИСТ 35.00 4 Ганотченко Е.П. профессор ИБАС 200.45 5 Петров А.А. профессор ИБАС 105.67</pre> Необходимо вычислить: количество преподавателей; сумму зарплат преподавателей по каждой кафедре.
2	Исходный файл содержит список преподавателей в формате «Модель, производитель, стоимость, пробег» (произвольное количество строк): <pre>1 BAS-2107; Автоваз; 140000.57; 16000 2 Гога-1; Хуань Вей Трейд; 50.00; 1000 3 BAS-5000; Автоваз; 1400000.00; 1000 4 X35; Hyundai; 750000; 1</pre> Необходимо вычислить: суммарную стоимость всех автомобилей; автомобиль с наименьшим пробегом.

3	Исходный файл содержит список преподавателей в формате «Транспорт; пункт отправки; пункт назначения; стоимость билета» (произвольное количество строк): <pre>1 Самолет*Москва*Пекин*30000.00 2 Автобус*Ставрополь*Москва*3000.00 3 Автобус*Невинномысск*Ставрополь*10.00 4 Автобус*Москва*Владивосток*50000 5 Самолет*СПб*Москва*1500</pre> Необходимо вычислить: количество автобусных рейсов; суммарную стоимость билетов рейсов самолетов; самый дорогой билет.
4	Исходный файл содержит список преподавателей в формате «ФИО, группа, курс, количество задолженностей» (произвольное количество строк): <pre>1 Иванов Е.А.*ИСТ-б-о-121*3*2 2 Петров А.А.*ИСТ-б-о-111*4*1 3 Кононов А.Е.*БАС-б-о-111*4*0 4 Катченко Е.А.*БАС-б-о-121*3*1</pre> Необходимо вычислить: суммарное количество задолженностей по каждому курсу; количество студентов без задолженностей.
5	Исходный файл содержит список преподавателей в формате «Товар, товарная группа, цена закупки, цена продажи» (произвольное количество строк): <pre>1 Молоко, 0.5 л & Продукты & 10.00 & 23.00 2 Хлеб белый & Продукты & 15.00 & 22.00 3 Молоко, 1.0 л & Продукты & 18.00 & 30.00 4 Туалетный утенок & Химия & 11.00 & 17.58 5 Фейри, 0.5 л & Химия & 17.00 & 88.00</pre> Необходимо вычислить: суммарную стоимость товаров по каждой группе; самый дорогой товар в каждой товарной группе.
6	Исходный файл содержит список преподавателей в формате «Товар, товарная группа, цена, склад» (произвольное количество строк): <pre>1 Кабель сетевой & Сетевое оборудование & 10.00 & Пулково 1 2 Коммутатор ASUS X100 & Сетевое оборудование & 15.00 & Склад №7 3 ПО Windows 8.1, 1.0 л & ПО & 18.00 & Склад №19 4 ПО Office 365 & ПО & 11.00 & Склад №19 5 Коннектор 234511 & Сетевое оборудование & 17.00 & Склад №7</pre> Необходимо вычислить: среднюю стоимость товаров по каждой группе; склад, содержащий максимальное количество продукции.
7	Исходный файл содержит список преподавателей в формате «Тип строения, количество комнат, метраж, стоимость» (произвольное количество строк): <pre>1 Дом*6*150*3500000 2 Квартира*1*49*790000 3 Квартира*2*59*1500000 4 Квартира*3*70*3000000 5 Комната*1*12*500000</pre> Необходимо вычислить: для каждого строения рассчитать стоимость квадратного метра; суммарную стоимость объектов по каждому типу.

8	Исходный файл содержит список преподавателей в формате «Название книги, жанр, цена» (произвольное количество строк): <pre>1 Привет с того света Детектив 150.00 2 Кто подставил кролика Детектив 200.10 3 Почемушка Детская литература 60.22 4 Капитошка Детская литература 230.14 5 Звезда 112 Фантастика 12.90</pre> Необходимо вычислить: самую дешевую книгу в каждом жанре; среднюю стоимость книг в каждом жанре.
9	Исходный файл содержит список преподавателей в формате «Товар, группа, стоимость, процент скидки» (произвольное количество строк): <pre>1 Молоко \ Продукты \ 40.00 \ 1 2 Тряпка \ Хозтовары \ 10.00 \ 5 3 Порошок \ Хозтовары \ 750.00 \ 10 4 Мясо \ Продукты \ 350 \ 5 5 Швабра \ Хозтовары \ 900.00 \ 5</pre> Необходимо вычислить: цену каждого товара за вычетом скидки; суммарную стоимость товаров по каждой группе.
10	Исходный файл содержит список преподавателей в формате «Товар, цена, количество, магазин» (произвольное количество строк): <pre>1 Рис 1 кг. / 120.11 / 54 / Магнит 2 Ручка шариковая / 23.00 / 15 / ТЦ Карандаш 3 Тетрадь 12 л. / 15.00 / 200 / ГеоКнига 4 Тетрадь 60 л. / 90.70 / 300 / ГеоКнига 5 Торт 1 кг. / 450 / 4 / Магнит</pre> Необходимо вычислить: среднюю цену товара по каждому магазину; суммарную стоимость для каждого товара.
11	Исходный файл содержит список преподавателей в формате «ФИО, должность, кафедра, заработная плата» (произвольное количество строк): <pre>1 Иванов С.Е. доцент ИСТ 56.00 2 Петров А.А. профессор ИВАС 100.23 3 Новиков П.С. ст. преподаватель ИСТ 35.00 4 Ганотченко Е.П. профессор ИВАС 200.45 5 Петров А.А. профессор ИВАС 105.67</pre> Необходимо вычислить: суммарный фонд заработной платы; среднюю зарплату по каждой кафедре.
12	Исходный файл содержит список преподавателей в формате «Модель, производитель, стоимость, пробег» (произвольное количество строк): <pre>1 BAS-2107; Автоваз; 140000.57; 16000 2 Гога-1; Хуань Вей Трейд; 50.00; 1000 3 BAS-5000; Автоваз; 1400000.00; 1000 4 X35; Hyundai; 750000; 1</pre> Необходимо вычислить: средний пробег по каждому производителю; среднюю стоимость автомобилей по каждому производителю.

13	Исходный файл содержит список преподавателей в формате «Транспорт; пункт отправки; пункт назначения; стоимость билета» (произвольное количество строк): <pre>1 Самолет*Москва*Пекин*30000.00 2 Автобус*Ставрополь*Москва*3000.00 3 Автобус*Невинномысск*Ставрополь*10.00 4 Автобус*Москва*Владивосток*50000 5 Самолет*СПб*Москва*1500</pre> Необходимо вычислить: среднюю стоимость билета по каждому виду транспорта; количество рейсов из Москвы.
14	Исходный файл содержит список преподавателей в формате «ФИО, группа, курс, количество задолженностей» (произвольное количество строк): <pre>1 Иванов Е.А.*ИСТ-б-о-121*3*2 2 Петров А.А.*ИСТ-б-о-111*4*1 3 Кононов А.Е.*БАС-б-о-111*4*0 4 Катченко Е.А.*БАС-б-о-121*3*1</pre> Необходимо вычислить: суммарное количество задолженностей по каждой группе; среднее количество задолженностей по каждому курсу.
15	Исходный файл содержит список преподавателей в формате «Товар, товарная группа, цена закупки, цена продажи» (произвольное количество строк): <pre>1 Молоко, 0.5 л & Продукты & 10.00 & 23.00 2 Хлеб белый & Продукты & 15.00 & 22.00 3 Молоко, 1.0 л & Продукты & 18.00 & 30.00 4 Туалетный утенок & Химия & 11.00 & 17.58 5 Фейри, 0.5 л & Химия & 17.00 & 88.00</pre> Необходимо вычислить: суммарную прибыль от продажи товаров; среднюю цену закупки по каждой группе.
16	Исходный файл содержит список преподавателей в формате «Товар, товарная группа, цена, склад» (произвольное количество строк): <pre>1 Кабель сетевой & Сетевое оборудование & 10.00 & Пулково 1 2 Коммутатор ASUS X100 & Сетевое оборудование & 15.00 & Склад №7 3 ПО Windows 8.1, 1.0 л & ПО & 18.00 & Склад №19 4 ПО Office 365 & ПО & 11.00 & Склад №19 5 Коннектор 234511 & Сетевое оборудование & 17.00 & Склад №7</pre> Необходимо вычислить: среднюю стоимость товаров по каждому складу; суммарную стоимость по каждой группе.
17	Исходный файл содержит список преподавателей в формате «Тип строения, количество комнат, метраж, стоимость» (произвольное количество строк): <pre>1 Дом*6*150*3500000 2 Квартира*1*49*790000 3 Квартира*2*59*1500000 4 Квартира*3*70*3000000 5 Комната*1*12*500000</pre> Необходимо вычислить: вывести объект с максимальной стоимостью квадратного метра; среднюю стоимость квадратного метра по объектам типа «Квартира».

18	Исходный файл содержит список преподавателей в формате «Название книги, жанр, цена» (произвольное количество строк): <pre> 1 Привет с того света Детектив 150.00 2 Кто подставил кролика Детектив 200.10 3 Почемушка Детская литература 60.22 4 Капитошка Детская литература 230.14 5 Звезда 112 Фантастика 12.90 </pre> Необходимо вычислить: суммарную стоимость книг по каждому жанру; среднюю стоимость книг по жанру «Фантастика».
19	Исходный файл содержит список преподавателей в формате «Товар, группа, стоимость, процент скидки» (произвольное количество строк): <pre> 1 Молоко \ Продукты \ 40.00 \ 1 2 Тряпка \ Хозтовары \ 10.00 \ 5 3 Порошок \ Хозтовары \ 750.00 \ 10 4 Мясо \ Продукты \ 350 \ 5 5 Швабра \ Хозтовары \ 900.00 \ 5 </pre> Необходимо вычислить: среднюю цену товара (с учетом скидки) по каждой товарной группе; товар с минимальной скидкой.
20	Исходный файл содержит список преподавателей в формате «Товар, цена, количество, магазин» (произвольное количество строк): <pre> 1 Рис 1 кг. / 120.11 / 54 / Магнит 2 Ручка шариковая / 23.00 / 15 / ТЦ Карандаш 3 Тетрадь 12 л. / 15.00 / 200 / ГеоКнига 4 Тетрадь 60 л. / 90.70 / 300 / ГеоКнига 5 Торт 1 кг. / 450 / 4 / Магнит </pre> Необходимо вычислить: суммарную стоимость товаров в каждом магазине; магазин с минимальным товарным ассортиментом.

10.7. Содержание отчета и его форма

Отчет по лабораторной работе должен содержать:

1. Номер и название лабораторной работы; задачи лабораторной работы.
3. Ответы на контрольные вопросы.
4. Экранные формы (консольный вывод) и листинг программного кода с комментариями, показывающие порядок выполнения лабораторной работы, и результаты, полученные в ходе её выполнения.

Отчет о выполнении лабораторной работы, подписанный студентом, сдается преподавателю.

10.8. Контрольные вопросы

1. Для чего используется шаблон Model-View-Controller?
2. Опишите назначение каждого элемента в шаблоне MVC.
3. Какие методы позволяют устанавливать и получать модель элемента управления Swing?
4. Опишите механизм реализации модели для элементов управления Swing. Какие классы могут выступать в качестве модели для элемента управления JComboBox, JTable, JListBox?
5. Можно ли реализовать класс модели, который будет подходить для всех элементов управления? Обоснуйте ответ.

1.9. Список литературы

Для выполнения лабораторной работы, при подготовке к защите, а также для ответа на контрольные вопросы рекомендуется использовать следующие источники: [1-3].

ЛАБОРАТОРНАЯ РАБОТА 11. ВЗАИМОДЕЙСТВИЕ С ФАЙЛОВОЙ СИСТЕМОЙ

11.1. Цель и содержание

Цель лабораторной работы: научиться разрабатывать приложения с использованием файлового ввода-вывода.

Задачи лабораторной работы:

- научиться программировать приложения с использованием файлового ввода-вывода;
- научиться использовать потоки Java;
- научиться применять потоки для взаимодействия с файлами различного типа.

11.2. Формируемые компетенции

ОПК-4; ОПК-5; ОПК-6.

11.3. Теоретическая часть

В любом высокоуровневом языке программирования уделяется особое внимание работе с файловой системой. Сохранение данных в файлы и загрузка данных из файла во время работы приложения существенно оптимизирует работу приложения. До выполнения данной лабораторной работы студенты были ограничены в используемом инструментарии. Обработка информации с использованием программ Java носила учебный характер. С изучением возможностей файлового ввода-вывода, учащимся предоставляется возможность перейти на новый качественный уровень – построение приложений для обработки больших объемов данных, то есть возможен переход от учебных заданий к промышленным проектам. Следует понимать,

что использование файлового хранилища (а в перспективе и баз данных) позволит в полной мере реализовать архитектуру MVC.

11.3.1 Роль потоков ввода-вывода

В Java API чтение информации из файлов происходит не напрямую, а через специальный промежуточный объект – поток. Объект, через который программист может читать последовательность байтов, – поток вывода. Поток, через который программист записывает байты, – поток ввода.

Источниками чтения и записи выступают файлы. Но технология потоков в Java и .NET Framework распространена на сетевые соединения и области памяти, которые также могут выступать в качестве источников ввода-вывода.

Очевидно, что перед программистом может возникнуть следующая задача: выбор способа чтения или записи. Можно читать (писать) по одному биту, байту, символу и т.д (различные примитивы). Выбор стратегии зависит от особенностей решаемой задачи.

Для записи (чтения) различных примитивов существуют различные классы. Основанием иерархии для классов ввода-вывода служат классы `InputStream` и `OutputStream`. Рассмотрим основные возможности класса `InputStream` (таблица 11.1).

Методы `read()` и `write()` блокируют доступ к потоку до завершения считывания (записи) байта. Если доступ не предоставляется немедленно (например при взаимодействии через сетевое соединение), то блокируется текущий поток соединения.

В классе `InputStrem` имеются абстрактные версии методов `read()` и `write()`, а также их неабстрактные аналоги.

Таблица 11.1 – Основные возможности `InputStream`.

Метод	Описание
<code>abstract int read()</code>	Считывает байт данных и возвращает его. При достижении конца потока возвращает <code>-1</code> . Метод абстрактный. Разработчик конкретного класса потока может

	переопределить данный метод (например для предоставления какой-либо дополнительной информации). Например в потомке <code>FileInputStream</code> данный метод читает 1 байт из файла; Поток <code>System.in</code> (имеет тип <code>InputStream</code>) позволяет читать данные с клавиатуры.
<code>int read(byte[] b)</code>	Читает байты в байтовый массив. Возвращает фактическое количество считанных байтов. Возвращает -1 при достижении конца потока. Данный метод считывает максимум <code>b.length</code> байтов.
<code>int read(byte[] b, int off, int len)</code>	Метод аналогичен предыдущему. Параметры: <code>b</code> – массив, в который помещаются данные; <code>off</code> – смещение в массиве <code>b</code> , обозначающее позицию, с которой должно начинаться размещение в нем байтов; <code>len</code> – максимальное количество считываемых байтов.
<code>long skip(long n)</code>	Метод пропускает <code>n</code> байтов в потоке ввода. Возвращает фактическое количество пропущенных байтов.
<code>int available()</code>	Возвращает количество неблокированных байтов потока.
<code>void close()</code>	Метод закрывает поток ввода.
<code>void mark(int lim)</code>	Метод позволяет установить маркер на определенной позиции <code>lim</code> в потоке ввода.
<code>void reset()</code>	Метод возвращает к последнему маркеру.
<code>boolean markSupported()</code>	Установку маркеров поддерживают не все потоки. Данный метод возвращает <code>true</code> если поток поддерживает работу с маркерами.

Таблица 11.2 – Основные возможности `OutputStream`.

<code>abstract int write(int b)</code>	Абстрактный метод записи байта.
<code>void write(byte[] b)</code>	Методы осуществляют запись байтов в поток из массива <code>b</code> . Параметры аналогичны соответствующим параметрам метода <code>read()</code> .
<code>void write(byte[] b, int off, int len)</code>	
<code>void close()</code>	Закрывает поток вывода.
<code>void flush()</code>	Метод очищает поток вывода. При этом отправляются данные из буфера в источник.

Таким образом, классы `InputStream` и `OutputStream` обладают только базовым функционалом для осуществления байтового ввода-вывода. Полные имена соответствующих классов: `java.io.InputStream` и `java.io.OutputStream`. Данные потоки не предоставляют никакого удобства для работы с символьными данными.

11.3.2 Иерархия потоков ввода-вывода

В Java существует более 50 типов, реализующих потоки ввода-вывода. Все они реализованы в виде иерархии.

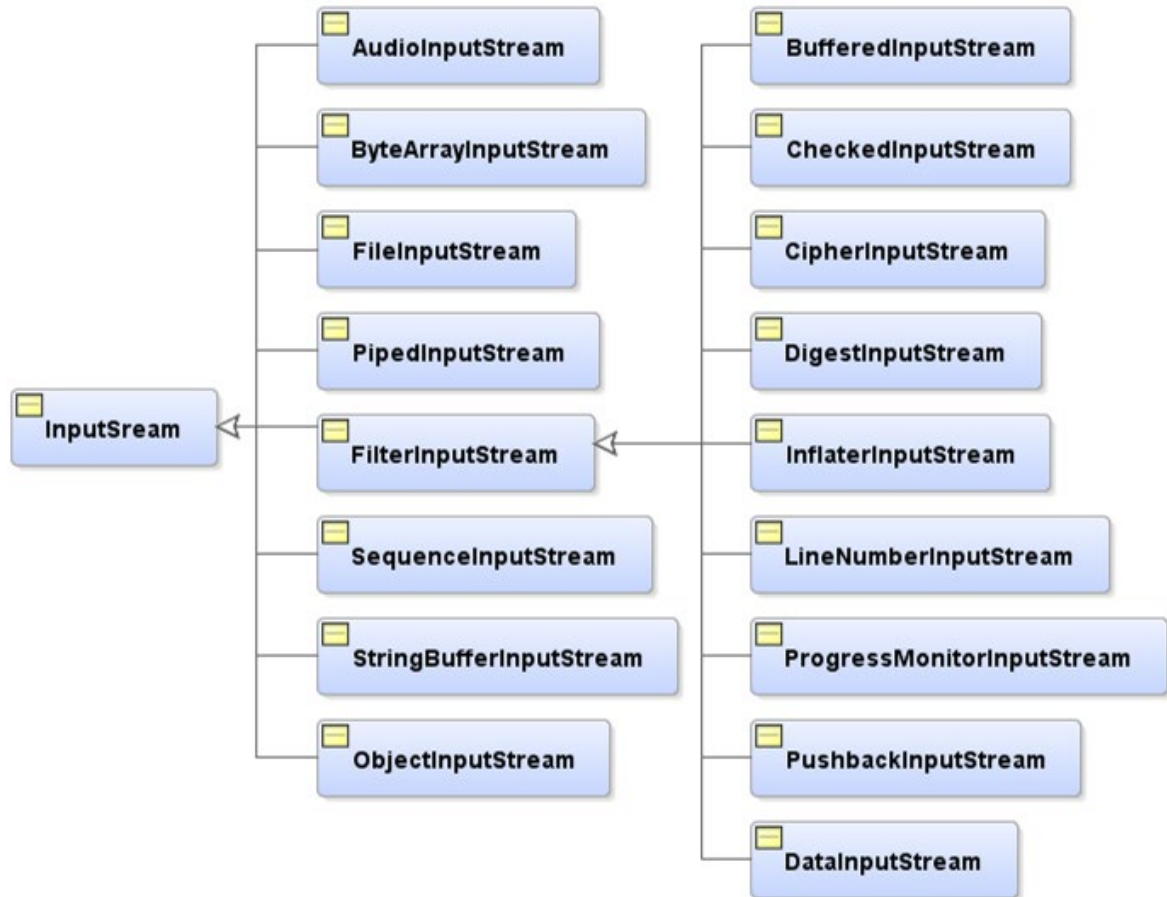


Рисунок 11.1 – Фрагмент иерархии классов, производных от InputStream.

Для классов, обрабатывающих байты и символы, существуют отдельные иерархии. Классы InputStream и OutputStream образуют иерархии классов для работы с байтами (рис. 11.1-11.3)

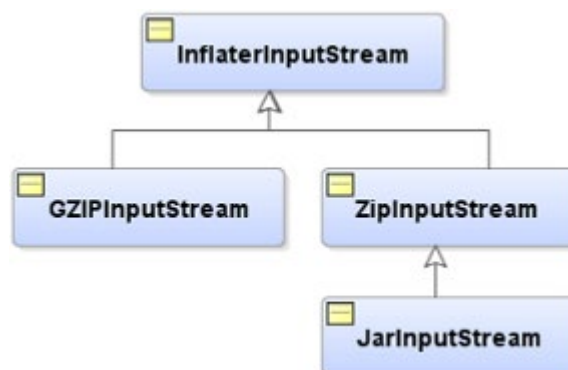
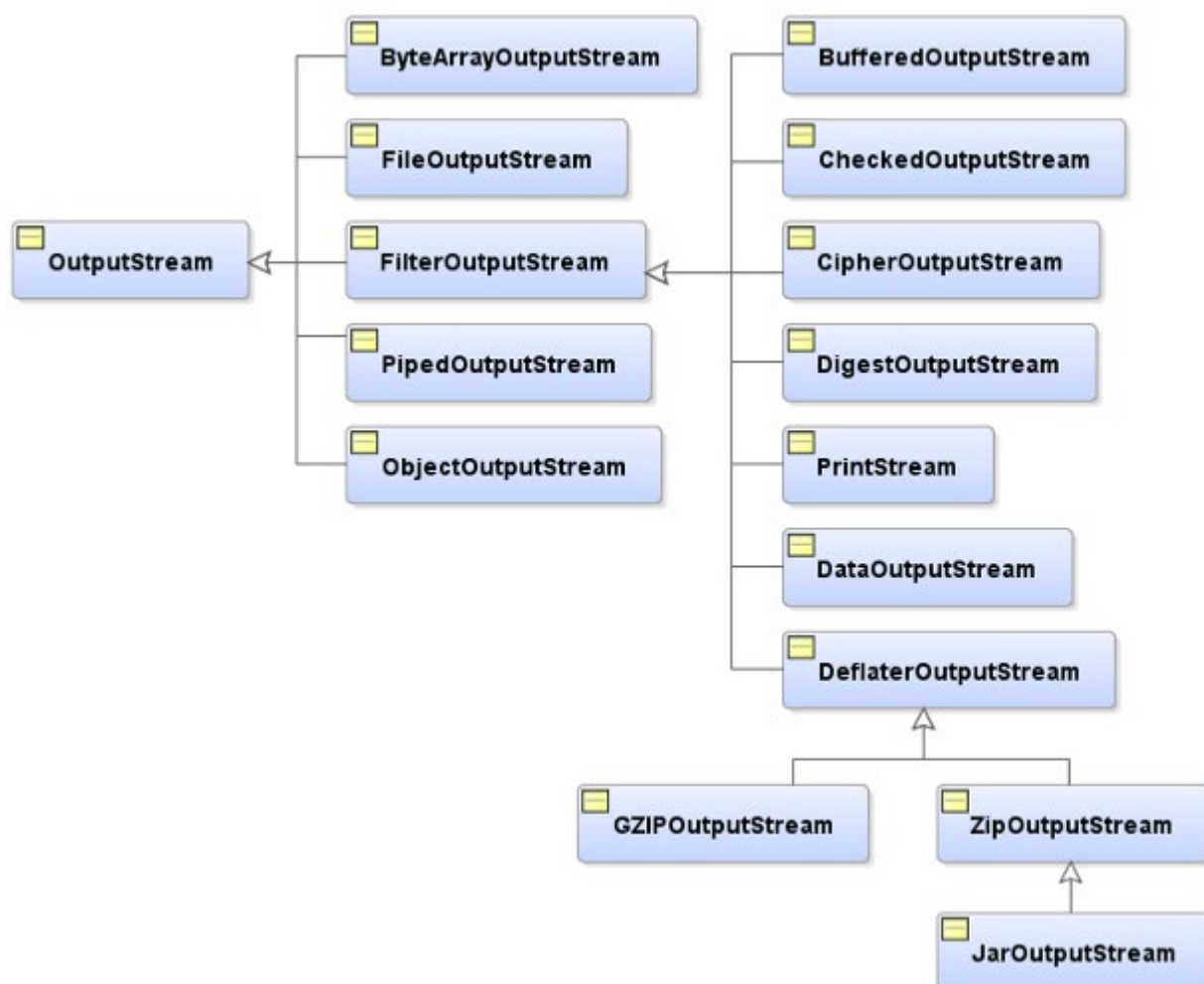


Рисунок 11.2 – Фрагмент иерархии классов, производных от `InflaterInputStream`.Рисунок 11.3 – Фрагмент иерархии классов, производных от `OutputStream`.

Для записи и чтения текста в формате Unicode необходимо использовать подклассы иерархий `Reader` и `Writer` (рис. 11.4, 11.5). Методы абстрактных классов `Reader` и `Writer` похожи на методы классов `InputStream` и `OutputStream`.

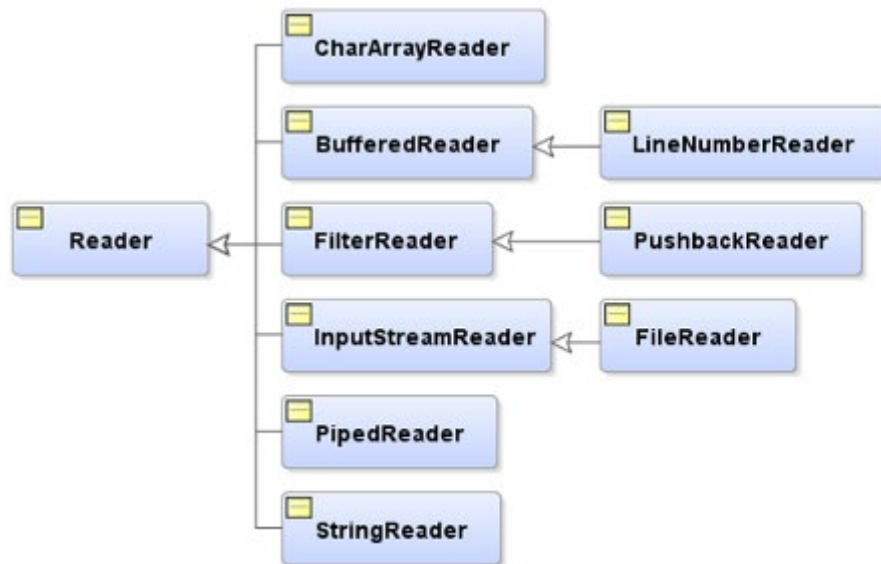


Рисунок 11.4 – Иерархии классов, производных от `Reader`.

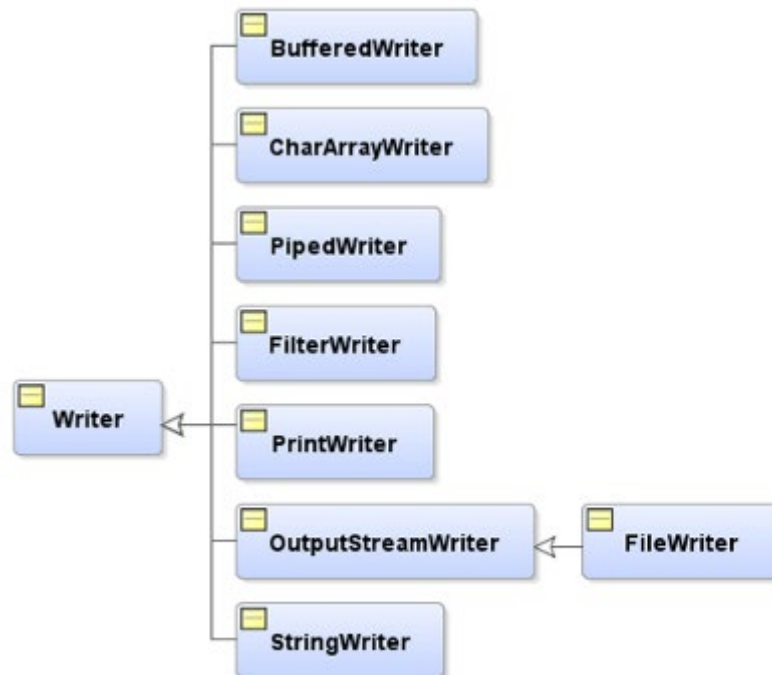


Рисунок 11.5 – Иерархии классов, производных от `Writer`.

11.3.3 Композиция потоковых фильтров

Классы `FileInputStream` и `FileOutputStream` позволяют создавать потоки и соединять их с конкретными файлами на диске (рис. 11.6).

```

1 package lesson11_1;
2 import java.io.FileInputStream;
3
4 public class Starter {
5     public static void main(String[] args) {
6         try {
7             FileInputStream fis = new FileInputStream("data.dat");
8         } catch (Exception e) {
9             System.out.println("Файл не найден!!!");
10        }
11    }
12 }
13 }

```

Рисунок 11.6 – Ассоциация объекта потока и конкретного экземпляра файла.

Следует обратить внимание, что файловые операции, обязательно необходимо заключать в блок обработки исключений `try ... catch ...` при указании имен (путей) файлов необходимо указывать полный путь. Указание отдельного имени файла (рис. 11.6, строка 7) указывает на тот факт, что поиск файла «data.dat» будет производиться в текущем каталоге пользователя. Для того, чтобы определить какой именно каталог является текущим каталогом пользователя необходимо использовать метод `System.getProperty("user.dir")`.

Как потомки абстрактных классов `InputStream` и `OutputStream` классы `FileInputStream` и `FileOutputStream` поддерживают чтение и запись только на уровне байтов. Поэтому, для объекта `fis`, созданного в представленном коде можно реализовать операцию `read()` чтения отдельных байтов или массивов байтов.

Работа с отдельными байтами не является эффективной и сопряжена с большими затратами времени для программиста. Поэтому в Java существует механизм разделения обязанностей: один класс читает байты, другой – составляет эти байты в высокоуровневые примитивы (числа, символы).

Подобный механизм представлен на рис. 11.7. Следует обратить внимание, что более высокоуровневый поток обертывает низкоуровневый.

```

1 package lesson11_1;
2 import java.io.DataInputStream;
3 import java.io.FileInputStream;
4
5 public class Starter {
6     public static void main(String[] args) {
7         try {
8
9             String user_dir = System.getProperty("user.dir");
10            System.out.println(user_dir);
11
12            FileInputStream fis = new FileInputStream("data.dat");
13            DataInputStream dis = new DataInputStream(fis);
14            double a1 = dis.readDouble();
15
16        } catch (Exception e) {
17            System.out.println("Файл не найден!!!");
18        }
19    }
20 }

```

Рисунок 11.7 – Композиция потоков.

При программировании с использованием файлового ввода-вывода каждое обращение к файловой системе инициирует новую процедуру чтения, то есть каждый вызов `read()` приводит к запросу к операционной системы на выдачу очередного байта. Это очень затратное по ресурсам чтение. В Java используются буферизованные потоки: запрашиваются целые блоки байтов, которые размещаются в специальном буфере. Требование применить буферизованное чтение приводит к использованию `BufferedInputStream` в качестве одного из уровней вложенности потоков (рис. 11.8).

```

1 package lesson11_1;
2 import java.io.BufferedInputStream;
3 import java.io.DataInputStream;
4 import java.io.FileInputStream;
5
6 public class Starter {
7     public static void main(String[] args) {
8         try {
9             DataInputStream dis = new DataInputStream(
10                new BufferedInputStream(
11                    new FileInputStream("data.dat")));
12            double d = dis.readDouble();
13
14        } catch (Exception e) {
15            System.out.println("Произошла какая-то ошибка");
16        }
17    }
18 }

```

Рисунок 11.8 – Реализации буферизованного чтения при композиции потоков.

Если необходимо реализовать процедуру условного чтения или чтения с предусловием, то используется класс `PushbackInputStream` (рис. 11.9). Данный класс позволяет прочитать байт методом `read()`, а затем вернуть его обратно методом `unread()`.

```

2 import java.io.BufferedInputStream;
3 import java.io.DataInputStream;
4 import java.io.FileInputStream;
5 import java.io.PushbackInputStream;
6
7 public class Starter {
8     public static void main(String[] args) {
9         try {
10             PushbackInputStream pbis = new PushbackInputStream(
11                 new BufferedInputStream(
12                     new FileInputStream("data.dat")));
13             int i = pbis.read();
14
15             System.out.println(i);
16
17             if(i != '1')
18                 pbis.unread(i);
19
20         } catch (Exception e) {
21             System.out.println("Произошла какая-то ошибка");
22         }
23     }
24 }

```

Рисунок 11.9 – Чтение с упреждением в механизме композиции потоков.

Очевиден механизм композиции потоков: объекты потоков передаются в конструкторы друг друга для достижения требуемой функциональности. В большинстве языков программирования возможности буферизованного чтения (записи) реализуется автоматически. Но в Java неудобство явной композиции потоков позволяет добиться большей гибкости. Например, используя рассмотренную логику чтения с использованием механизма композиции потоков можно легко организовать чтение вещественных чисел с использованием буфера из файла, сжатого с использованием zip-архива.

11.3.4 Работа с текстовыми потоками

При работе с текстовыми файлами следует учитывать кодировки. Вывод в двоичном формате работает быстрее, но текстовые потоки более удобны.

Методы записи в текстовые файлы. Для вывода текста в файл можно использовать класс `PrintWriter` (рис. 11.10). В строках 8 и 9 приводятся различные варианты создания потока данного типа.

```

1 package lesson11_1;
2 import java.io.FileWriter;
3 import java.io.PrintWriter;
4
5 public class Starter {
6     public static void main(String[] args) {
7         try {
8             PrintWriter pw = new PrintWriter("info.txt");
9             //PrintWriter pw = new PrintWriter(new FileWriter("info.txt"));
10            // или
11            // данные для записи
12            int a = 10;
13            String s = "Nikolaev E.I.";
14            float ff = 7.78f;
15            // запись
16            pw.println(a);
17            pw.println(s);
18            pw.println(ff);
19            pw.close();
20
21            System.out.println("Данные успешно записаны.");
22
23        } catch (Exception e) {
24            System.out.println("Произошла какая-то ошибка");
25        }
26    }
27 }

```

Рисунок 11.10 – Запись строк в файл.

Для записи информации с использованием класса `PrintWriter` используются методы `print()`, `println()`, `printf()`, то есть те же методы, что и для вывода в консоль. Метод `println()` дописывает символ конца строки. Для операционной системы Windows – это «`\r\n`», для Unix – «`\n`». Символ конца строки можно узнать с использованием вызова `System.getProperty("line.separator")`. Методы записи класса `PrintWriter` не генерируют исключений. Если необходимо проверить наличие ошибок нужно вызвать метод `checkError()`.

Методы чтения данных из текстовых файлов. Для осуществления чтения данных из текстовых файлов используются различные подходы. Самый простой подход – использование методов класса `Scanner`. На рис.11.11 представлен пример программы, использующей данный подход.

```

1 package lesson11_2;
2
3 import java.nio.file.Paths;
4 import java.util.Scanner;
5 import java.io.FileInputStream;
6
7 public class StringReaderExample {
8
9     public static void main(String[] args) {
10         //Scanner s = new Scanner(System.in);
11         try {
12             //Scanner file_in = new Scanner(Paths.get("scan.txt"));
13             Scanner file_in = new Scanner(new FileInputStream("scan.txt"));
14             String str = file_in.nextLine();
15             System.out.println(str);
16         } catch (Exception e) {
17             System.out.println("File operation error!");
18         }
19     }
20 }

```

Рисунок 11.11 – Реализация чтения из файла с использованием класса `Scanner`.

Следует обратить внимание на строки 12 и 13 (рис. 11.11), которые демонстрируют различные варианты ассоциации объекта `Scanner` и файла. Объект `Scanner` используется для ввода информации с консоли, но в данном случае поток ассоциируется не с консолью, а с файловым потоком. Доступны все методы класса `Scanner` для чтения различных данных: `nextLine()`, `nextInt()`, `next()` и т.д.

Второй способ – использование `BufferedReader`. Данный класс определяет метод `ReadLine()` для считывания строки текста из файла (рис. 11.12). В представленном фрагменте происходит построчное чтение текстового файла с использованием `BufferedReader`.

```

1 package lesson11_2;
2
3 import java.io.BufferedReader;
4 import java.io.FileInputStream;
5 import java.io.InputStreamReader;
6
7 public class StringReaderExample {
8
9     public static void main(String[] args) {
10         try {
11             BufferedReader br = new BufferedReader(
12                 new InputStreamReader(
13                     new FileInputStream("scan.txt")));
14             String str;
15             while((str = br.readLine()) != null)
16             {
17                 System.out.println(str);
18             }
19         } catch (Exception e) {
20             System.out.println("File operation error!");
21         }
22     }
23 }

```

Рисунок 11.12 – Реализация чтения из файла с использованием класса `BufferedReader`.

Очевидно, что данный подход обладает существенным недостатком – отсутствуют методы чтения числовых данных. Строки, считываемые методом `ReadLine()` программисту приходится «парсить». Поэтому использование класса `Scanner` является более предпочтительным.

11.3.5 Использование двоичных файлов

На рис. 11.1-11.4 представлены только классы потоков, но в системе потоков Java существуют также интерфейсы, предоставляющие различные методы. На рис. 11.13 представлены интерфейсы и их связи с классами.

Интерфейс `DataOutput` содержит методы для записи примитивов: чисел, логических значение, символов и т.д.: `writeChars()`, `writeByte()`, `writeInt()`, `writeShort()`, `writeLong()`, `writeFloat()`, `writeDouble()`, `writeChar()`, `writeBoolean()`, `writeUTF()`.

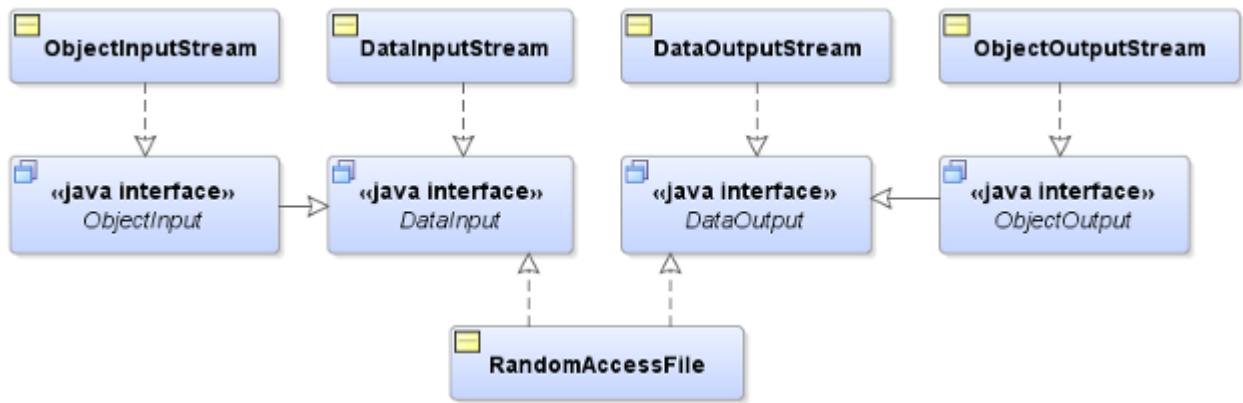


Рисунок 11.13 – Интерфейсы в системе ввода-вывода.

Метод `writeInt()` записывает целочисленное значение в формате 4-байтового двоичного значения; метод `writeDouble()` – в виде 8-байтового значения. При открытии файла результата записи окажется неудобным для чтения, но операции записи (считывания) выполняются гораздо быстрее в двоичном формате.

Метод `writeUTF()` записывает строковые данные с использованием версии 8-разрядного формата Unicode. Строка сначала кодируется как UTF-16, затем конвертируется в UTF-8. Такой сложный алгоритм обусловлен необходимостью обеспечения обратной совместимости с виртуальными машинами Java. На современном этапе следует использовать метод `writeChars()` вместо `writeUTF()`.

Для чтения данных из двоичных файлов существуют методы интерфейса `DataInput`: `readChar()`, `writeByte()`, `readInt()`, `readShort()`, `readLong()`, `readFloat()`, `readDouble()`, `readBoolean()`, `readUTF()`.

Класс `DataInputStream` предлагает конкретную реализацию интерфейса `DataInput`. Для чтения данных из конкретного файла необходимо применить композицию потоков `DataInputStream` и `FileInputStream`.

Соответственно, для записи файлов используется композиция `FileOutputStream` и `DataOutputStream` (реализует интерфейс `DataOutput`).

Двоичные потоки широко применяются для повышения скорости работы с объектами файловой системы.

11.3.6 Использование файлов с произвольным доступом

Класс `RandomAccessFile` позволяет позиционировать указатель и записывать данные в файл в требуемой позиции.

Все файлы на диске имеют возможность произвольного доступа, но при использовании сетевых потоков ввода-вывода произвольный доступ недоступен. У любого файла с произвольным доступом имеется указатель файла – позиция следующего байта (читаемого или записываемого). Для позиционирования указателя предназначен метод `seek()`. Метод `getFilePointer()` возвращает текущую позицию в файле.

Класс `RandomAccessFile` реализует интерфейсы `DataInput` и `DataOutput`, то есть поддерживает как запись, так и чтение.

На рис. 11.14 представлен фрагмент программы, демонстрирующий использование возможностей `RandomAccessFile`.

```

1 package lesson11_2;
2
3 import java.io.RandomAccessFile;
4
5 public class StringReaderExample {
6
7     public static void main(String[] args) {
8         try {
9             RandomAccessFile raf = new RandomAccessFile("scan.txt", "rw");
10            // устанавливаем указатель на требуемую позицию
11            raf.seek(3);
12            char c = raf.readChar();
13            System.out.println(c);
14
15        } catch (Exception e) {
16            System.out.println("File operation error!");
17        }
18    }
19 }

```

Рисунок 11.14 – Использование потоков с произвольным доступом.

В таблице 11.3 представлены методы класса `RandomAccessFile`.

Таблица 11.3 – Основные возможности RandomAccessFile.

Метод	Описание
RandomAccessFile(String file, String mode)	Конструктор класса потока. Возможные значения параметра mode: r – режим только для чтения; rw – режим чтения и записи; rws – режим чтения и записи с синхронной записью на диск данных и метаданных при каждом обновлении; rwd – режим чтения и записи с синхронной записью на диск только данных.
long getFilePointer()	Получение значения текущего положения файлового указателя.
void seek(long pos)	Устанавливает файловый указатель в позицию pos.
long length()	Получение длины файла в байтах.

11.3.7 Использование ZIP-архивов в программах

Архивы ZIP позволяют хранить несколько файлов в уплотненном формате. У каждого архива есть заголовок, содержащий информацию о методе сжатия и метainформацию о файле. В Java для работы с архивом применяется класс ZipInputStream (чтение файла). Архив читается по отдельным записям. В таблице 11.4 представлены методы работы с архивами.

Таблица 11.4 – Основные возможности класса ZipInputStream.

Метод	Описание
ZipInputStream(InputStream in)	Создает объект типа ZipInputStream, который распаковывает данные из указанного InputStream.
ZipEntry getNextEntry()	Возвращает объект ZipEntry для следующей записи. Если записей нет, то вернется null.
void closeEntry()	Закрывает текущую открытую запись в ZIP-архиве.

Для записи архивных ZIP-файлов используется класс ZipOutputStream. Для каждой записи, которая заносится в ZIP-архив, создается объект ZipEntry. Требуемое имя файла передается конструктору класса ZipEntry.

Таблица 11.5 – Основные возможности класса ZipOutputStream.

Метод	Описание
ZipOutputStream(OutputStream in)	Создает объект ZipOutputStream, который позволяет записывать уплотненные данные в поток вывода типа ZipOutputStream.
void putNextEntry(ZipEntry ze)	Записывает данные из указанной записи типа ZipEntry в поток вывода и устанавливает его в положение для вывода следующей порции данных. После этого данные могут быть выведены в этот поток с помощью метода write().
void closeEntry()	Закрывает текущую открытую запись в ZIP-архиве. Для перехода к следующей записи используется метод putNextEntry().
void setLevel(int level)	Устанавливает степень сжатия для последующих записей. По умолчанию устанавливается значение степени сжатия Deflater.DEFAULT_COMPRESSION. Генерирует исключение типа IllegalArgumentException, если заданная степень сжатия недействительна. level – степень сжатия от 0 (NO_COMPRESSION) до 9 (BEST_COMPRESSION).
void setMethod(int method)	Устанавливает алгоритм сжатия по умолчанию для любых записей, выводимых в поток типа ZipOutputStream, без указания конкретного алгоритма сжатия данных. Параметр method определяет алгоритм сжатия данных: DEFLATED или STORED.

При работе с архивами используется вспомогательный класс ZipEntry. Именно на этом классе основана работа с архивными потоками. В таблице 11.6 показаны основные методы класса ZipEntry.

Таблица 11.6 – Основные возможности класса ZipEntry.

Метод	Описание
ZipEntry(String name)	Создает запись в Zip-архиве с заданным именем. Параметр name – имя записи.
long getCrc()	Возвращает значение контрольной суммы CRC32 для данной записи типа ZipEntry.

String getName()	Возвращает имя данной записи.
long getSize()	Возвращает размер данной записи без сжатия или значение -1, если размер записи без уплотнения неизвестен.
boolean isDirectory()	Возвращает логическое значение true, если данная запись является каталогом.
void setMethod(int method)	Задаёт алгоритм сжатия записей (DEFLATED или STORED).
void setSize(long size)	Устанавливает размер данной записи. Требуется только в том случае, если задан алгоритм сжатия данных STORED. Параметр size – размер данной записи без сжатия.
void setCrc(long crc)	Устанавливает контрольную сумму CRC32 для данной записи. Для вычисления этой суммы должен использоваться класс CRC32. Требуется только в том случае, если задан алгоритм сжатия данных STORED. Параметр crc – контрольная сумма данной записи.

11.4. Оборудование и материалы

Для выполнения лабораторной работы рекомендуется использовать персональный компьютер со следующими характеристиками: 64-разрядный (x64) процессор с тактовой частотой 1 ГГц и выше, оперативная память – 1 Гб и выше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система Windows 7 и выше, Java Development Kit 8 и выше; IDE Eclipse for Java SE (or Java EE) developers.

11.5. Указания по технике безопасности

Студенты должны следовать общепринятой технике безопасности для пользователей персональных компьютеров. Не следует самостоятельно производить ремонт технических средств, установку и удаление программного обеспечения. В случае обнаружения неисправностей необходимо сообщить об

этом администратору компьютерного класса (обслуживающему персоналу лаборатории).

11.6. Методика и порядок выполнения работы

В данной лабораторной работе необходимо реализовать приложение с графическим пользовательским интерфейсом в соответствии с вариантом индивидуального задания из лабораторной работы №10. Приложение, полученное в ходе выполнения предыдущей лабораторной работы, следует изменить таким образом, чтобы классы модели считывали и сохраняли данные в текстовых файлах.

11.7. Содержание отчета и его форма

Отчет по лабораторной работе должен содержать:

1. Номер и название лабораторной работы; задачи лабораторной работы.
3. Ответы на контрольные вопросы.
4. Экранные формы (консольный вывод) и листинг программного кода с комментариями, показывающие порядок выполнения лабораторной работы, и результаты, полученные в ходе её выполнения.

Отчет о выполнении лабораторной работы, подписанный студентом, сдается преподавателю.

11.8. Контрольные вопросы

1. Что такое поток ввода-вывода?
2. Поясните структуру дерева иерархии потоков. Назовите основные потоки файлового ввода-вывода.
3. Для чего предназначены потоки `FileInputStream` и `FileOutputStream`? Назовите основные методы данных классов.

4. Какие классы потоков для работы с текстовыми файлами вы знаете?

Поясните назначение методов данных классов.

5. Опишите классы потоков для работы с двоичными файлами.

6. Поясните особенности классов потоков для реализации произвольного доступа к файлам. Опишите методы данных классов.

7. Опишите типы, которые используются при организации взаимодействия с ZIP-файлами.

2.9. Список литературы

Для выполнения лабораторной работы, при подготовке к защите, а также для ответа на контрольные вопросы рекомендуется использовать следующие источники: [1], [2].

ЛАБОРАТОРНАЯ РАБОТА 12. МНОГОПОТОЧНЫЕ ПРИЛОЖЕНИЯ JAVA

12.1. Цель и содержание

Цель лабораторной работы: изучить принципы построения многопоточных приложений с использованием технологии Java SE.

Задачи лабораторной работы:

- научиться создавать потоки;
- научиться работать с состояниями потоков и их свойствами;
- научиться управлять потоками в приложениях Java.

12.2. Формируемые компетенции

ОПК-4; ОПК-5; ОПК-6.

12.3. Теоретическая часть

Многозадачность операционной системы – возможность одновременно выполнять несколько программ. Например, можно печатать документы во время их редактирования или приема электронной почты. Вполне возможно, что современный компьютер оснащен более чем одним центральным процессором, но число одновременно выполняющихся процессов не ограничивается количеством процессоров. Операционная система выделяет время процессора квантами для каждого процесса, создавая впечатление параллельного выполнения программ.

Многопоточные программы расширяют принцип многозадачности, перенося его на один уровень ниже, чтобы отдельные приложения могли выполнять многие задачи одновременно. Каждая задача обычно называется

поток, т.е. потоком управления. Программы, способные одновременно выполнять больше одного потока, называются многопоточными.

Следует отличать систему со многими процессами и многими потоками. Различие следующее: если у каждого процесса имеется собственный набор переменных, то потоки могут разделять одни и те же общие данные. Разделяемые переменные обеспечивают более высокую эффективность взаимодействия потоков и облегчают связь между потоками. Кроме того, в некоторых операционных системах потоки являются более «легковесными», чем процессы, – они требуют меньших издержек на свое создание и уничтожение по сравнению с процессами.

Многопоточная обработка имеет исключительную практическую ценность. Например, браузер должен уметь одновременно загружать многие изображения. Веб-серверу нужно обслуживать параллельные запросы. Программы с графическим пользовательским интерфейсом имеют отдельные потоки для сбора событий в пользовательском интерфейсе из среды операционной системы.

Средства работы с потоками существуют в различных объектно-ориентированных языках. Программирование многопоточных приложений обладает высокой сложностью.

12.3.1 Основы использования потоков в Java

Рассмотрим пример задачи, демонстрирующий преимущества многопоточности. На рис. 12.1 показан исходный код приложения, имитирующего выполнение затратного по времени приложения.

```

4 import java.awt.Dimension;
5 import java.awt.event.ActionEvent;
6 import java.awt.event.ActionListener;
7
8 import javax.swing.Box;
9 import javax.swing.BoxLayout;
10 import javax.swing.JButton;
11 import javax.swing.JFrame;
12
13 public class TaskExample extends JFrame {
14     public TaskExample() {
15         super("Состояние: ");
16         setDefaultCloseOperation(EXIT_ON_CLOSE);
17         setSize(400, 150);
18         Container cont = getContentPane();
19         BoxLayout box = new BoxLayout(cont, BoxLayout.X_AXIS);
20         cont.setLayout(box);
21         JButton btnStart = new JButton("Log");
22         btnStart.setPreferredSize(new Dimension(200, 100));
23         btnStart.addActionListener(new ActionListener() {
24             @Override
25             public void actionPerformed(ActionEvent e) {
26                 Task(50);
27             }
28         });
29         cont.add(Box.createHorizontalGlue());
30         cont.add(btnStart);
31         cont.add(Box.createHorizontalGlue());
32         setLocationRelativeTo(null);
33         setVisible(true);
34     }

```

Рисунок 12.1 – Определение формы приложения.

На рис. 12.1 в строке 26 в качестве обработчика события нажатия кнопки вызывается метод Task(long millis). Определение данного метода приводится на рис. 12.2.

```

36     public void Task(long millis){
37         System.out.println("Start task!");
38         for(int i = 1; i <= 1000; i++)
39         {
40             System.out.println("Current " + i);
41             try { Thread.sleep(millis); }
42             catch (InterruptedException e) {}
43         }
44         System.out.println("End task!");
45     }

```

Рисунок 12.2 – Метод Task(long millis).

При нажатии на кнопку «Log» начнет выполняться метод, выполняющий 1000 итераций. В каждой итерации выполняется задержка 50 миллисекунд. До

момента завершения выполнения задачи все приложение будет заблокировано: его нельзя завершить, ни один элемент управления не реагирует на события.

Для преодоления полученного недостатка приложения необходимо изменить приложение. На рис. 12.3 представлен код класса MyTask. В данный класс вынесена вся логика (ресурсоемкая задача).

```

1 package lesson12_theory;
2
3 public class MyTask implements Runnable {
4     private long millis;
5     private long id;
6
7     MyTask(long m){
8         millis = m;
9     }
10
11     @Override
12     public void run() {
13         id = Thread.currentThread().getId();
14         System.out.println("Start task!");
15         for(int i = 1; i <= 1000; i++)
16         {
17             System.out.println("Task " + id + ":" + i);
18             try { Thread.sleep(millis); }
19             catch (InterruptedException e) {}
20         }
21         System.out.println("End task!");
22     }
23 }

```

Рисунок 12.3 – Код метода MyTask.

Обработчик нажатия кнопки «Log» представлен на рис. 12.4.

```

23 btnStart.addActionListener(new ActionListener() {
24     @Override
25     public void actionPerformed(ActionEvent e) {
26         Runnable task = new MyTask(10);
27         Thread t = new Thread(task);
28         t.start();
29     }
30 });

```

Рисунок 12.4 – Запуск задачи.

При такой реализации запуска длительной задачи, кнопка «Log» будет активна постоянно, и пользователь сможет запустить несколько потоков одновременно (рис. 12.5).

```

Task 18:93
Task 18:94
Task 18:95
Task 18:96
Task 18:97
Start task!
Task 19:1
Task 18:98
Task 19:2
Task 18:99
Task 19:3
Task 19:4
Task 18:100
Task 18:101
Task 19:5
Task 19:6
Task 18:102
Task 19:7
Task 18:103
Task 19:8
Task 18:104

```

Рисунок 12.5 – Вывод программы.

Таким образом, можно обобщить алгоритм использования потоков:

1. Создание кода выполняемой задачи в теле переопределяемого метода `run()` класса, реализующего интерфейс `Runnable`. Этот интерфейс очень прост и содержит единственный метод:

```

public interface Runnable {
    void run();
}

```

2. Реализация класса, реализующего интерфейс `Runnable` (рис. 12.3).

3. Конструирование объекта для задачи (рис. 12.4, строка 26):

```
Runnable r = new MyTask();
```

4. Создание объекта типа `Thread` из объекта типа `Runnable` (рис. 12.4, строка 27).

5. Запуск потока на выполнение (рис. 12.4, строка 28).

Можно вызвать метод `run()` из объекта класса `Thread`, но в таком случае отдельный поток создан не будет, а задача выполнится в вызывающем потоке.

В таблице 12.1 представлены основные методы класса `Thread`.

Таблица 12.1 – Методы класса Thread.

Метод	Описание
Thread(Runnable target)	Конструирует новый исполняемый поток, вызывающий метод run() для указанного целевого объекта.
void start()	Запускает поток, иницируя вызов метода run(). Этот метод немедленно возвращает управление. Новый поток выполняется параллельно.
void run()	Вызывает метод run() для связанного с ним объекта типа Runnable.

12.3.2 Управление потоками. Прерывание потоков

Поток прерывается, когда его метод run() возвращает управление, выполнив оператор return вслед за последним оператором в своем теле, или в том случае, если возникает исключение, которое не перехватывается в данном методе. В первоначальной версии Java также присутствовал метод stop(), который мог быть вызван из другого потока, чтобы прервать выполнение данного потока. Но теперь этот метод не рекомендуется к применению.

Поток можно прервать принудительно. Но метод interrupt() может вызываться только для запроса прерывания потока. Когда метод interrupt() вызывается для потока, устанавливается состояние прерывания данного потока. Это признак типа boolean, имеющийся у каждого потока. В каждом потоке следует периодически проверять значение этого признака, чтобы знать, когда он должен быть прерван. Для того чтобы выяснить, было ли установлено состояние прерывания потока, вызывается статический метод Thread.currentThread(), получающий текущий поток и вызывающий далее метод isInterrupted().

Но если поток заблокирован, то проверить состояние его прерывания нельзя. В таком случае применяется исключение типа InterruptedException. Когда метод interrupt() вызывается для потока, который заблокирован, например, вызовом метода sleep() или wait(), блокирующий вызов прерывается исключением типа InterruptedException.

В языке Java не существует требования, чтобы прерванный поток прекратил выполнение. Прерывание лишь привлекает внимание потока. А прерванный поток может сам решить, как реагировать на прерывание. Некоторые потоки настолько важны, что должны обработать исключение и продолжить свое выполнение. Но зачастую поток должен просто интерпретировать прерывание как запрос на прекращение своего выполнения.

На рис. 12.6 показан исходный код модернизированного класса MyTask, который может быть прерван.

```

1 package lesson12_theory;
2
3 public class MyTask implements Runnable {
4     private long millis;
5     private long id;
6
7     MyTask(long m){
8         millis = m;
9     }
10
11     @Override
12     public void run() {
13         id = Thread.currentThread().getId();
14         System.out.println("Start task!");
15         for(int i = 1; i <= 1000; i++)
16         {
17             System.out.println("Task " + id + ":" + i);
18             if(Thread.currentThread().isInterrupted()){
19                 System.out.println("Interrupted!!!");
20                 break;
21             }
22             try { Thread.sleep(millis); }
23             catch (InterruptedException e) {}
24         }
25         System.out.println("End task!");
26     }
27 }

```

Рисунок 12.6 – Задача, учитывающая возможность прерывания.

Для демонстрации возможности прерывания на форму добавлена кнопка «Stop», обработчик которой осуществляет посылку запроса на прерывание (рис. 12.7).

```

17 public TaskExample() {
18     super("Состояние: ");
19     setDefaultCloseOperation(EXIT_ON_CLOSE);
20     setSize(400, 150);
21     Container cont = getContentPane();
22     BoxLayout box = new BoxLayout(cont, BoxLayout.X_AXIS);
23     cont.setLayout(box);
24     // Создание кнопки Start
25     JButton btnStart = new JButton("Start Task");
26     btnStart.setPreferredSize(new Dimension(100, 100));
27     btnStart.addActionListener(new ActionListener() {
28         @Override
29         public void actionPerformed(ActionEvent e) {
30             Runnable task = new MyTask(10);
31             parallelThread = new Thread(task);
32             parallelThread.start();
33         }
34     });
35     // Создание кнопки Stop
36     JButton btnStop = new JButton("Stop Task");
37     btnStop.setPreferredSize(new Dimension(100, 100));
38     btnStop.addActionListener(new ActionListener() {
39         @Override
40         public void actionPerformed(ActionEvent e) {
41             parallelThread.interrupt();
42         }
43     });
44
45     cont.add(Box.createHorizontalGlue());
46     cont.add(btnStart);
47     cont.add(Box.createHorizontalGlue());
48     cont.add(btnStop);
49     cont.add(Box.createHorizontalGlue());
50     setLocationRelativeTo(null);
51     setVisible(true);
52 }

```

Рисунок 12.7 – Модификация формы для предоставления пользователю возможности прервать поток.

После таких модификаций не удастся достигнуть желаемого результата, так как метод `isInterrupted()` не вернет `true` для заблокированного потока. Поток блокируется методами `wait()` и `sleep()`. На рис. 12.8 представлен вариант метода `run()` класса `MyTask`, который выполняет задачу прерывания потока.

```

11  @Override
12  public void run() {
13      id = Thread.currentThread().getId();
14      System.out.println("Start task!");
15      for(int i = 1; i <= 1000; i++)
16      {
17          System.out.println("Task " + id + ":" + i);
18          try { Thread.sleep(millis); }
19          catch (InterruptedException e) {
20              System.out.println("Interopted!!!");
21              break;
22          }
23      }
24      System.out.println("End task!");
25  }

```

Рисунок 12.8 – Модификация класса MyTask.

Проверка состояния прерывания потока вызовом метода `isInterrupted()` не обязательна и не удобна, если можно вызвать метод `sleep()` (или другой прерываемый метод) после каждого рабочего шага цикла. Если же метод `sleep()` вызывается, когда установлено состояние прерывания, поток не переходит в состояние ожидания. Вместо этого он очищает состояние и генерирует исключение типа `InterruptedException`. Таким образом, если метод `sleep()` вызывается в цикле, то проверять состояние прерывания не следует. Вместо этого лучше организовать перехват исключения типа `InterruptedException`, как показано на рис. 12.8.

Существуют два похожих метода `interrupted()` и `isInterrupted()`. Статический метод `interrupted()` проверяет, был ли прерван текущий поток. Более того, вызов этого метода приводит к очистке состояния прерывания потока. Метод экземпляра `isInterrupted()` можно использовать для проверки, был ли прерван любой поток. Его вызов не приводит к изменению состояния прерывания.

Используемые для конструирования управляемых потоков методы рассмотрены в таблице 12.2.

Таблица 12.2 – Методы класса Thread, участвующие в реализации механизма прерывания.

Метод	Описание
<code>void interrupt()</code>	Посылает потоку запрос на прерывание. Признак состояния прерывания потока устанавливается равным логическому значению <code>true</code> . Если поток в данный момент блокирован вызовом метода <code>sleep()</code> , генерируется исключение типа <code>InterruptedException</code> .
<code>static boolean interrupted()</code>	Проверяет, был ли прерван текущий поток. Следует иметь в виду, что это статический метод. Его вызов имеет побочный эффект: признак состояния прерывания текущего потока устанавливается равным логическому значению <code>false</code> .
<code>boolean isInterrupted()</code>	Проверяет, был ли прерван поток. В отличие от статического метода <code>interrupted()</code> , вызов этого метода не изменяет состояние прерывания потока.
<code>static Thread currentThread()</code>	Возвращает объект типа <code>Thread</code> , представляющий текущий выполняющийся поток.

12.3.3 Состояния потоков

Потоки могут находиться в одном из шести состояний:

- новый;
- исполняемый;
- блокированный;
- ожидающий;
- временно ожидающий;
- заверченный.

Чтобы определить текущее состояние потока, достаточно вызвать метод `getState()`.

Если поток создан в результате операции `new`, например `new MyThread()`, то он еще не запущен на выполнение. Это означает, что он находится в новом состоянии и что программа еще не запустила на выполнение код в данном

потоке. Прежде чем поток будет запущен, необходимо выполнить определенные подготовительные операции.

Как только вызывается метод `start()`, поток оказывается в исполняемом состоянии. Исполняемый поток может выполняться или не выполняться в данный момент, поскольку от операционной системы зависит, будет ли выделено потоку время на выполнение. В спецификации Java это отдельное состояние не указывается. Выполняющийся поток по-прежнему находится в исполняемом состоянии.

Если поток запущен, он не обязательно продолжает выполняться. На самом деле даже желательно, чтобы выполняющийся поток периодически приостанавливался, давая возможность выполниться другим потокам. Особенности планирования потоков зависят от конкретных служб, предоставляемых операционной системой. Системы приоритетного планирования выделяют каждому исполняемому потоку по кванту времени для выполнения его задачи. Когда этот квант времени истекает, операционная система выгружает поток и дает возможность выполниться другому потоку. Выбирая следующий поток, операционная система принимает во внимание приоритеты потоков.

Все современные настольные и серверные операционные системы используют приоритетное (вытесняющее) планирование. Но на переносных устройствах вроде мобильных телефонов может применяться кооперативное планирование. В таких устройствах поток теряет управление только в том случае, если он вызывает метод `yield()`, заблокирован или находится в состоянии ожидания.

На машинах с несколькими процессорами каждый процессор может выполнять поток, что позволяет иметь несколько потоков, работающих одновременно. Очевидно, если потоков больше, чем процессоров, планировщик вынужден заниматься разделением времени для их выполнения. Не следует забывать, что в любой момент времени исполняемый поток может выполняться

или не выполняться. Именно поэтому рассматриваемое здесь состояние потока называется исполняемым, а не исполняющимся.

Когда поток находится в состоянии блокировки или ожидания, он временно не активен. Он не выполняет никакого кода и потребляет минимум ресурсов. На планировщике потоков лежит обязанность повторно активизировать его. В зависимости от метода, которым было установлено неактивное состояние, возможны следующие пути:

1. Когда поток пытается получить встроенную блокировку объектов (но не объект типа `Lock` из библиотеки `java.util.concurrent`), которая в настоящий момент захвачена другим потоком, он становится заблокированным. Поток разблокируется, когда все остальные потоки освобождают блокировку и планировщик потоков позволяет захватить ее данному потоку.

2. Когда поток ожидает от другого потока уведомления планировщика о наступлении некоторого условия, он входит в состояние ожидания. Переход в состояние ожидания происходит при вызове метода `Object.wait()` или `Thread.join()` либо в ожидании объекта типа `Lock` или `Condition` из библиотеки `java.util.concurrent`. Но на практике отличия состояний блокировки и ожидания не существенны.

3. Несколько методов принимают в качестве параметра время ожидания. Их вызов вводит поток в состояние временного ожидания. Это состояние сохраняется до тех пор, пока не истечет заданное время ожидания или не будет получено соответствующее уведомление. К числу методов со временем ожидания относятся `Object.wait()`, `Thread.join()`, `Lock.tryLock()` и `Condition.await()`.

На рис. 12.9 показаны состояния, в которых может находиться поток, а также возможные переходы между ними. Когда поток находится в состоянии блокировки или ожидания (и, конечно, когда он завершается), к запуску планируется другой поток. А когда поток повторно активизируется (например, по истечении времени ожидания или в том случае, если ему удастся захватить блокировку), планировщик потоков сравнивает его приоритет с приоритетом

выполняющихся в данный момент потоков. Если приоритет данного потока ниже, он приостанавливается и запускается новый поток.

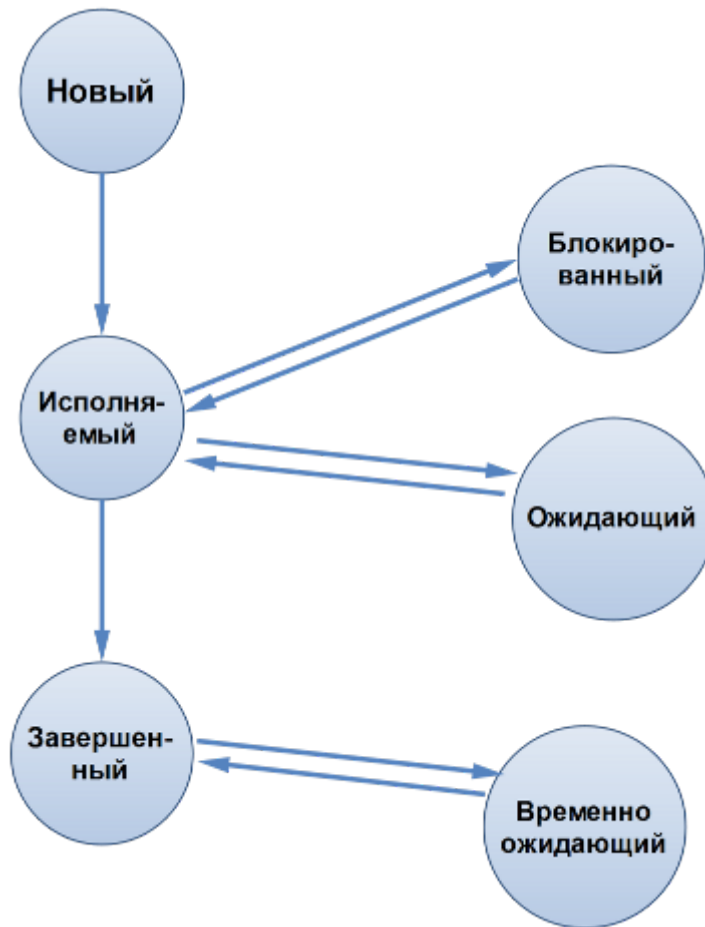


Рисунок 12.9 – Состояния потоков и возможные переходы.

Поток завершается по одной из следующих причин.

1. Прекращает свое существование естественным образом при нормальном завершении метода `run()`.
2. Прекращает свое существование внезапно, поскольку неперехваченное исключение прерывает выполнение метода `run()`.

В частности, поток можно уничтожить, вызвав его метод `stop()`. Этот метод генерирует объект ошибки типа `ThreadDeath`, который уничтожает поток. Но метод `stop()` не рекомендован к применению, и поэтому следует избегать его применения в прикладном коде.

В таблице 12.3 указаны методы, связанные с состоянием потоков и их изменениями.

Таблица 12.3 – Методы класса Thread для управления состоянием потоков.

Метод	Описание
<code>void join()</code>	Ожидает завершения указанного потока.
<code>void join(long millis)</code>	Ожидает завершения указанного потока или истечения заданного периода времени.
<code>Thread.State getState()</code>	Получает состояние данного потока. Может принимать значения NEW, RUNNABLE, BLOCKED, WAITING, TIMED_WAITING или TERMINATED.
<code>void stop()</code>	Останавливает поток. Этот метод не рекомендован к применению.
<code>void suspend()</code>	Временно приостанавливает поток. Этот метод не рекомендован к применению.
<code>void resume()</code>	Возобновляет поток. Вызывается только после вызова метода <code>suspend()</code> . Этот метод не рекомендован к применению.

12.3.4 Свойства потоков.

В языке программирования Java каждый поток имеет свой приоритет. По умолчанию поток наследует приоритет того потока, который его создал. Повысить или понизить приоритет любого потока можно, вызвав метод `setPriority()`. А установить приоритет можно, указав любое значение в пределах от `MIN_PRIORITY` (определено в классе `Thread` равным 1) до `MAX_PRIORITY` (равно 10). Обычному приоритету соответствует значение `NORM_PRIORITY`, равное 5.

Всякий раз, когда планировщик потоков выбирает новый поток для выполнения, он предпочитает потоки с более высоким приоритетом. Но приоритеты потоков сильно зависят от системы. Когда виртуальная машина полагается на реализацию потоков средствами главной платформы, на которой она выполняется, приоритеты потоков Java привязываются к уровням приоритетов этой платформы, где может быть больше или меньше уровней приоритетов.

Например, в Windows предусмотрено семь уровней приоритетов. Некоторые приоритеты Java привязываются к тому же самому уровню приоритета операционной системы. В виртуальной машине Oracle для Linux приоритеты потоков вообще игнорируются. Все потоки имеют одинаковый приоритет.

Не рекомендуется строить свои программы таким образом, чтобы их правильное функционирование зависело от уровней приоритетов.

С приоритетами потоков связан метод `yield()`, который заставляет текущий выполняющийся поток уступить управление. Если есть другие исполняемые потоки с приоритетом не ниже приоритета данного потока, они будут запланированы для выполнения следующими.

В Java существует механизм потоковых демонов. Превратить поток в потоковый демон можно, сделав следующий вызов:

```
t.setDaemon(true);
```

Демон – это поток, у которого нет других целей, кроме обслуживания других потоков. В качестве примера можно привести потоки таймера, посылающие регулярные «такты» другим потокам, или же потоки, очищающие устаревшие записи в кеше. Когда остаются только потоковые демоны, виртуальная машина завершает работу. Нет смысла продолжать выполнение программы, когда все оставшиеся потоки являются демонами.

Неоправданное использование потоковых демонов может быть небезопасно. Потоковый демон вообще не должен обращаться к такому постоянному ресурсу, как файл или база данных, поскольку он может быть прерван в любой момент, даже посередине операции.

12.4. Оборудование и материалы

Для выполнения лабораторной работы рекомендуется использовать персональный компьютер со следующими характеристиками: 64-разрядный (x64) процессор с тактовой частотой 1 ГГц и выше, оперативная память – 1 Гб и

выше, свободное дисковое пространство – не менее 1 Гб, графическое устройство DirectX 9. Программное обеспечение: операционная система Windows 7 и выше, Java Development Kit 8 и выше; IDE Eclipse for Java SE (or Java EE) developers.

12.5. Указания по технике безопасности

Студенты должны следовать общепринятой технике безопасности для пользователей персональных компьютеров. Не следует самостоятельно производить ремонт технических средств, установку и удаление программного обеспечения. В случае обнаружения неисправностей необходимо сообщить об этом администратору компьютерного класса (обслуживающему персоналу лаборатории).

12.6. Методика и порядок выполнения работы

Модифицируйте приложение, полученное в предыдущей лабораторной работе №11 таким образом, чтобы операции чтения и записи в файлы осуществлялись в отдельных потоках. Реализуйте визуализацию процесса чтения и записи: пользователю должен показываться процент выполнения.

12.7. Содержание отчета и его форма

Отчет по лабораторной работе должен содержать:

1. Номер и название лабораторной работы; задачи лабораторной работы.
3. Ответы на контрольные вопросы.
4. Экранные формы (консольный вывод) и листинг программного кода с комментариями, показывающие порядок выполнения лабораторной работы, и результаты, полученные в ходе её выполнения.

Отчет о выполнении лабораторной работы, подписанный студентом, сдается преподавателю.

12.8. Контрольные вопросы

8. Какие классы для организации многопоточного приложения вы знаете?
9. Как установить и получить приоритет потока?
10. Чем отличаются методы `run()` и `start()` класса `Thread`?
11. Как выполнить принудительное прерывание выполнения потока?
12. Почему не рекомендуется совместное использование `isInterrupted()` и `sleep()`?
13. Опишите основные шаги для создания параллельной ветки программного кода.
14. Опишите состояния потоков в Java и условия перехода между ними.

12.9. Список литературы

Для выполнения лабораторной работы, при подготовке к защите, а также для ответа на контрольные вопросы рекомендуется использовать следующие источники: [4], [7], [8].

ЗАКЛЮЧЕНИЕ

Методические указания по дисциплине «Современные технологии программирования» предоставляют студентам необходимый теоретический материал, а также позволяют получить практические навыки для дальнейшего самообучения, развития своих навыков в области программирования на языке Java.

Материал лабораторных работ представлен на базе современной среды разработки программных средств – интегрированной среды разработки Eclipse. Технологии, доступные разработчику с использованием данной IDE, непрерывно совершенствуются и требуют от программиста непрерывного повышения профессионального уровня. Полный спектр возможностей IDE Eclipse выходит далеко за рамки данного лабораторного практикума, но выполнение заданий лабораторного практикума обеспечивает студентов знаниями, востребованными на современном рынке труда.

В качестве основы построения лабораторного практикума предложена технология Java SE. Данная технология предлагает широкий спектр алгоритмов и технологий разработки программного обеспечения: широкий набор библиотечных типов; специализированные синтаксические конструкции; современные примитивы программирования; компоненты разработки графических приложений; классы для работы с объектами файловой системы. Все представленные технологии в рамках одного языка позволяют студентам получить практический опыт использования современных инструментов разработки объектно-ориентированных приложений.

СПИСОК ЛИТЕРАТУРЫ

Список основной литературы

1. Шилдт, Г. Java. Полное руководство. 8-е изд. / Г. Шилдт; пер. с англ. В.А. Коваленко. – М.: ООО «И.Д. Вильямс», 2012. – 1104 с.
2. Корнелл, Г. Java. Библиотека профессионала, том 1. Основы. 9-е изд. / Г. Корнелл, К.С. Хорстманн; пер. с англ. И.В. Берштейн. – М.: ООО «И.Д. Вильямс», 2014. – 864 с.
3. Корнелл, Г. Java. Библиотека профессионала, том 2. Расширенные средства. 9-е изд. / Г. Корнелл, К.С. Хорстманн; пер. с англ. И.В. Берштейн. – М.: ООО «И.Д. Вильямс», 2014. – 864 с.
4. Язык программирования C# 5.0 и платформа .NET 4.5. 6-е изд. Пер. с англ. / Э. Троелсен. – М.: Изд. «Издательский дом Вильямс». 2013. – 1312 с.
5. Албахари Дж. C# 5.0. Справочник. Полное описание языка / Дж. Албахари, Б. Албахари. – М.: Изд. «Издательский дом Вильямс». 2013. – 1008 с.

Список дополнительной литературы

6. Стиллмен Э. Изучаем C#. 3-е изд. / Э. Стиллмен, Дж. Грин. – СПб.: Питер. 2014. – 816 с.
7. Флентов, М. Библия C#. 2-е изд. / М. Флентов. – СПб.: БХВ-Петербург, 2011. – 560 с.
8. Ватсон, Б. C# 4.0 на примерах: пер. с англ. / Б. Ватсон. – СПб.: БХВ-Петербург, 2011. – 608 с.
9. Буч, Г. Объектно-ориентированный анализ и проектирование с примерами приложений, 3-е изд.: Пер. с англ. – М.: ООО «И.Д. Вильямс», 2008. – 720 с.: ил. – Парал. тит. англ.
10. Петцольд Ч. Программирование для Microsoft Windows 8. 6-е изд. / Ч. Петцольд. – СПб.: Питер. 2013 – 1008 с.

11. Гранд, М. Шаблоны проектирования в Java / М. Гранд; пер. с. англ. С. Беликовой. – М.: Новое знание, 2004. – 559 с.
12. Монахов, В. Язык программирования Java и среда NetBeans. 3-е изд., перераб. и доп. / В.В. Монахова. – СПб.: БХВ-Петербург, 2011. – 704 с.
13. Блох, Дж. Java. Эффективное программирование. 2-е изд. / Дж. Блох; пер. с. англ. Е. Коротылева. – М.: Лори, 2014. – 461 с.
14. Блинов, И. Java: методы программирования: уч.-мет. Пособие / И.И. Блинов, В.С. Романчик. – Минск: «Четыре четверти», 2013. – 896 с.

ПРИЛОЖЕНИЕ 1. ИСХОДНЫЙ КОД КЛАССА «БИБЛИОТЕКА» НА ЯЗЫКЕ C#

```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4
5  namespace LR2_metod
6  {
7      5 references
8      public class Библиотека
9      {
10         private readonly Dictionary<Книга, List<Читатель>> книги =
11             new Dictionary<Книга, List<Читатель>>();
12
13         private Библиотека()
14         {
15         }
16
17         1 reference
18         public static Библиотека Инициализация()
19         {
20             var res = new Библиотека();
21             var новый = new Книга
22             {
23                 автор = "Коротков А.Е.",
24                 жанр = Жанр.Образование,
25                 код = "001",
26                 название = "Программирование C#",
27                 состояние = Состояние.Доступна
28             };
29
30             res.книги.Add(
31                 новый,
32                 new List<Читатель>());
33             новый = new Книга
34             {
35                 автор = "Тодеев А.А.",
36                 жанр = Жанр.Образование,
37                 код = "002",
38                 название = "Java. Учебник",
39                 состояние = Состояние.Доступна,
40             };
41
42             res.книги.Add(
43                 новый,
44                 new List<Читатель>());
45
46             return res;
47         }
48     }
  
```

```

47 public void ВыдачаКниги(String кодКниги, Читатель читатель)
48 {
49     // Ищем книгу с переданным кодом
50     var res = (from ob in книги where ob.Key.код
51                 == кодКниги select ob).ToList();
52
53     if (res.Count > 0)
54     {
55         if (res[0].Key.состояние == Состояние.Доступна)
56         {
57             res[0].Key.состояние = Состояние.НаРуках;
58             res[0].Value.Add(читатель);
59             Console.WriteLine("Выдача книги зафиксирована.");
60         }
61         else
62             Console.WriteLine("Книга на руках!!!");
63     }
64     else
65         Console.WriteLine("Выдача книги зафиксирована.");
66 }
67
68
69 0 references
70 public void ВозвратКниги(Читатель читатель)
71 {
72     Console.WriteLine("Функция не реализована!!!");
73 }
74
75 1 reference
76 public void Каталог()
77 {
78     foreach (var book in книги.Keys)
79     {
80         Console.WriteLine(book.автор + "\n" +
81                             book.название + "\n" + book.код);
82     }
83 }

```

Современные технологии программирования

Методические указания к лабораторным работам

для студентов специальности

09.03.02 «Информационные системы и технологии»

Автор: Е.И. Николаев

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Методические указания
по организации и проведению самостоятельной работы
по дисциплине
«Современные технологии программирования»
для студентов направления подготовки
09.03.02 «Информационные системы и технологии»
направленность (профиль)
**«Прикладное программирование в интеллектуальных информационных
системах»**

Ставрополь, 2026

1. Общие положения

Самостоятельная работа – планируемая учебная, учебно-исследовательская, научно-исследовательская работа студентов, выполняемая во внеаудиторное (аудиторное) время по заданию и при методическом руководстве преподавателя, но без его непосредственного участия (при частичном непосредственном участии преподавателя, оставляющем ведущую роль за работой студентов).

Самостоятельная работа студентов (СРС) в ВУЗе является важным видом учебной и научной деятельности студента. Самостоятельная работа студентов играет значительную роль в рейтинговой технологии обучения.

К основным видам самостоятельной работы студентов относятся:

- формирование и усвоение содержания конспекта лекций на базе рекомендованной лектором учебной литературы, включая информационные образовательные ресурсы (электронные учебники, электронные библиотеки и др.);
- написание докладов;
- подготовка к семинарам, практическим и лабораторным работам, их оформление;
- составление аннотированного списка статей из соответствующих журналов по отраслям знаний (педагогических, психологических, методических и др.);
- выполнение учебно-исследовательских работ, проектная деятельность;
- подготовка практических разработок и рекомендаций по решению проблемной ситуации;
- выполнение домашних заданий в виде решения отдельных задач, проведения типовых расчетов, расчетно-компьютерных и индивидуальных работ по отдельным разделам содержания дисциплин и т.д.;
- компьютерный текущий самоконтроль и контроль успеваемости на базе электронных обучающих и аттестующих тестов;
- выполнение курсовых работ (проектов) в рамках дисциплин;
- выполнение выпускной квалификационной работы и др.

Методика организации самостоятельной работы студентов зависит от структуры, характера и особенностей изучаемой дисциплины, объема часов на ее изучение, вида заданий для самостоятельной работы студентов, индивидуальных качеств студентов и условий учебной деятельности.

Процесс организации самостоятельной работы студентов включает в себя следующие этапы:

- подготовительный (определение целей, составление программы, подготовка методического обеспечения, подготовка оборудования);
- основной (реализация программы, использование приемов поиска информации, усвоения, переработки, применения, передачи знаний, фиксирование результатов, самоорганизация процесса работы);
- заключительный (оценка значимости и анализ результатов, их систематизация, оценка эффективности программы и приемов работы, выводы о направлениях оптимизации труда).

Самостоятельная работа по дисциплине «Современные технологии программирования» направлена на формирование следующих **компетенций**:

Код, формулировка компетенции	Код, формулировка индикатора
-------------------------------	------------------------------

ОПК-4 – способен участвовать в разработке технической документации, связанной с профессиональной деятельностью с использованием стандартов, норм и правил	ОПК-4 _{ид-4.1} Демонстрирует знания основных стандартов оформления технической документации на различных стадиях жизненного цикла информационной системы
	ОПК-4 _{ид-4.2} Владеет основными стандартами оформления технической документации на различных стадиях жизненного цикла информационной системы
	ОПК-4 _{ид-4.3} Применяет стандарты оформления технической документации на различных стадиях жизненного цикла информационной системы
	ОПК-4 _{ид-4.4} Умеет составлять техническую документацию на различных этапах жизненного цикла информационной системы
ОПК-5 – способен устанавливать программное и аппаратное обеспечение для информационных и автоматизированных систем	ОПК-5 _{ид-5.1} Демонстрирует знания основ системного администрирования, администрирования СУБД, современные стандарты информационного взаимодействия систем
	ОПК-5 _{ид-5.2} Выполняет параметрическую настройку информационных и автоматизированных систем
	ОПК-5 _{ид-5.3} Владеет навыками установки программного и аппаратного обеспечения информационных и автоматизированных систем
	ОПК-5 _{ид-5.4} Выполняет работы по настройке аппаратного и программного обеспечения при решении задач профессиональной деятельности
	ОПК-5 _{ид-5.5} Осуществляет администрирование программного обеспечения при решении задач профессиональной деятельности
	ОПК-5 _{ид-5.6} Выполняет работы по проверке работоспособности аппаратного обеспечения при решении задач профессиональной деятельности
ОПК-6 – способен разрабатывать алгоритмы и программы, пригодные для практического применения в области информационных систем и технологий	ОПК-6 _{ид-6.1} Имеются знания об основных языках программирования, базах данных, операционных системах и оболочек, современных программных средах разработки информационных систем и технологий
	ОПК-6 _{ид-6.2} Применяет современные программные среды разработки информационных систем и технологий для автоматизации бизнес-процессов, решения прикладных задач различных классов
	ОПК-6 _{ид-6.3} Имеются навыки программирования, отладки и тестирования прототипов программно-технических комплексов задач
	ОПК-6 _{ид-6.4} Использует методы алгоритмизации и

	языки программирования, пригодные для практического применения в области информационных систем и технологий.
--	--

2. Цель и задачи самостоятельной работы

Ведущая цель организации и осуществления СРС совпадает с целью обучения студента – формирование набора общенаучных, профессиональных и специальных компетенций будущего бакалавра по соответствующему направлению подготовки

При организации СРС важным и необходимым условием становятся формирование умения самостоятельной работы для приобретения знаний, навыков и возможности организации учебной и научной деятельности. Целью самостоятельной работы студентов является овладение фундаментальными знаниями, профессиональными умениями и навыками деятельности по профилю, опытом творческой, исследовательской деятельности. Самостоятельная работа студентов способствует развитию самостоятельности, ответственности и организованности, творческого подхода к решению проблем учебного и профессионального уровня.

Задачами СРС являются:

- систематизация и закрепление полученных теоретических знаний и практических умений студентов;
- углубление и расширение теоретических знаний;
- формирование умений использовать нормативную, правовую, справочную документацию и специальную литературу;
- развитие познавательных способностей и активности студентов: творческой инициативы, самостоятельности, ответственности и организованности;
- формирование самостоятельности мышления, способностей к саморазвитию, самосовершенствованию и самореализации;
- развитие исследовательских умений;
- использование материала, собранного и полученного в ходе самостоятельных занятий на семинарах, на практических и лабораторных занятиях, при написании курсовых и выпускной квалификационной работ, для эффективной подготовки к итоговым зачетам и экзаменам.

3. Технологическая карта самостоятельной работы студента

Коды реализуемых компетенций	Вид деятельности студентов	Средства и технологии оценки	Объем часов, в том числе (акад.)		
			СРС	Контактная работа с преподавателями	Всего
6 семестр					
ОПК-4, ОПК-5, ОПК-6	Самостоятельное изучение литературы и источников	Собеседование	2	2	4
ОПК-4, ОПК-5, ОПК-6	Подготовка лабораторным занятиям	Защита ЛР	38	2	40
Итого за 6 семестр			40	4	44
Итого			40	4	44

4. Порядок выполнения самостоятельной работы студентом

4.1. Методические рекомендации по работе с учебной литературой

При работе с книгой необходимо подобрать литературу, научиться правильно ее читать, вести записи. Для подбора литературы в библиотеке используются алфавитный и систематический каталоги.

Важно помнить, что рациональные навыки работы с книгой - это всегда большая экономия времени и сил.

Правильный подбор учебников рекомендуется преподавателем, читающим лекционный курс. Необходимая литература может быть также указана в методических разработках по данному курсу.

Изучая материал по учебнику, следует переходить к следующему вопросу только после правильного уяснения предыдущего, описывая на бумаге все выкладки и вычисления (в том числе те, которые в учебнике опущены или на лекции даны для самостоятельного вывода).

При изучении любой дисциплины большую и важную роль играет самостоятельная индивидуальная работа.

Особое внимание следует обратить на определение основных понятий курса. Студент должен подробно разбирать примеры, которые поясняют такие определения, и уметь строить аналогичные примеры самостоятельно. Нужно добиваться точного представления о том, что изучаешь. Полезно составлять опорные конспекты. При изучении материала по учебнику полезно в тетради (на специально отведенных полях) дополнять конспект лекций. Там же следует отмечать вопросы, выделенные студентом для консультации с преподавателем.

Выводы, полученные в результате изучения, рекомендуется в конспекте выделять, чтобы они при перечитывании записей лучше запоминались.

Опыт показывает, что многим студентам помогает составление листа опорных сигналов, содержащего важнейшие и наиболее часто употребляемые формулы и понятия. Такой лист помогает запомнить формулы, основные положения лекции, а также может служить постоянным справочником для студента.

Чтение научного текста является частью познавательной деятельности. Ее цель – извлечение из текста необходимой информации. От того на сколько осознанна читающим собственная внутренняя установка при обращении к печатному слову (найти нужные сведения, усвоить информацию полностью или частично, критически проанализировать материал и т.п.) во многом зависит эффективность осуществляемого действия.

Выделяют **четыре основные установки в чтении научного текста:**

информационно-поисковый (задача – найти, выделить искомую информацию)

усваивающая (усилия читателя направлены на то, чтобы как можно полнее осознать и запомнить как сами сведения излагаемые автором, так и всю логику его рассуждений)

аналитико-критическая (читатель стремится критически осмыслить материал, проанализировав его, определив свое отношение к нему)

творческая (создает у читателя готовность в том или ином виде – как отправной пункт для своих рассуждений, как образ для действия по аналогии и т.п. – использовать суждения автора, ход его мыслей, результат наблюдения, разработанную методику, дополнить их, подвергнуть новой проверке).

Основные виды систематизированной записи прочитанного:

Аннотирование – предельно краткое связное описание просмотренной или прочитанной книги (статьи), ее содержания, источников, характера и назначения;

Планирование – краткая логическая организация текста, раскрывающая содержание и структуру изучаемого материала;

Тезирование – лаконичное воспроизведение основных утверждений автора без привлечения фактического материала;

Цитирование – дословное выписывание из текста выдержек, извлечений, наиболее существенно отражающих ту или иную мысль автора;

Конспектирование – краткое и последовательное изложение содержания прочитанного.

Конспект – сложный способ изложения содержания книги или статьи в логической последовательности. Конспект аккумулирует в себе предыдущие виды записи, позволяет всесторонне охватить содержание книги, статьи. Поэтому умение составлять план, тезисы, делать выписки и другие записи определяет и технологию составления конспекта.

Методические рекомендации по составлению конспекта:

1. Внимательно прочитайте текст. Уточните в справочной литературе непонятные слова. При записи не забудьте вынести справочные данные на поля конспекта;

2. Выделите главное, составьте план;

3. Кратко сформулируйте основные положения текста, отметьте аргументацию автора;

4. Законспектируйте материал, четко следуя пунктам плана. При конспектировании старайтесь выразить мысль своими словами. Записи следует вести четко, ясно.

5. Грамотно записывайте цитаты. Цитируя, учитывайте лаконичность, значимость мысли.

В тексте конспекта желательно приводить не только тезисные положения, но и их доказательства. При оформлении конспекта необходимо стремиться к емкости каждого

предложения. Мысли автора книги следует излагать кратко, заботясь о стиле и выразительности написанного. Число дополнительных элементов конспекта должно быть логически обоснованным, записи должны распределяться в определенной последовательности, отвечающей логической структуре произведения. Для уточнения и дополнения необходимо оставлять поля.

Овладение навыками конспектирования требует от студента целеустремленности, повседневной самостоятельной работы.

4.2. Методические рекомендации по подготовке к практическим и лабораторным занятиям

Для того чтобы практические и лабораторные занятия приносили максимальную пользу, необходимо помнить, что упражнение и решение задач проводятся по вычитанному на лекциях материалу и связаны, как правило, с детальным разбором отдельных вопросов лекционного курса. Следует подчеркнуть, что только после усвоения лекционного материала с определенной точки зрения (а именно с той, с которой он излагается на лекциях) он будет закрепляться на практических занятиях как в результате обсуждения и анализа лекционного материала, так и с помощью решения проблемных ситуаций, задач. При этих условиях студент не только хорошо усвоит материал, но и научится применять его на практике, а также получит дополнительный стимул (и это очень важно) для активной проработки лекции.

При самостоятельном решении задач нужно обосновывать каждый этап решения, исходя из теоретических положений курса. Если студент видит несколько путей решения проблемы (задачи), то нужно сравнить их и выбрать самый рациональный. Полезно до начала вычислений составить краткий план решения проблемы (задачи). Решение проблемных задач или примеров следует излагать подробно, вычисления располагать в строгом порядке, отделяя вспомогательные вычисления от основных. Решения при необходимости нужно сопровождать комментариями, схемами, чертежами и рисунками.

Следует помнить, что решение каждой учебной задачи должно доводиться до окончательного логического ответа, которого требует условие, и по возможности с выводом. Полученный ответ следует проверить способами, вытекающими из существа данной задачи. Полезно также (если возможно) решать несколькими способами и сравнить полученные результаты. Решение задач данного типа нужно продолжать до приобретения твердых навыков в их решении.

4.3. Методические рекомендации по самопроверке знаний

После изучения определенной темы по записям в конспекте и учебнику, а также решения достаточного количества соответствующих задач на практических занятиях и самостоятельно студенту рекомендуется, провести самопроверку усвоенных знаний, ответив на контрольные вопросы по изученной теме.

В случае необходимости нужно еще раз внимательно разобраться в материале.

Иногда недостаточность усвоения того или иного вопроса выясняется только при изучении дальнейшего материала. В этом случае надо вернуться назад и повторить плохо усвоенный материал. Важный критерий усвоения теоретического материала - умение решать задачи или пройти тестирование по пройденному материалу. Однако следует помнить, что правильное решение задачи может получиться в результате применения механически заученных формул без понимания сущности теоретических положений.

4.4. Методические рекомендации по написанию научных текстов (докладов, докладов, эссе, научных статей и т.д.)

Перед тем, как приступить к написанию научного текста, важно разобраться, какова истинная цель вашего научного текста - это поможет вам разумно распределить свои силы и время.

Во-первых, сначала нужно определиться с идеей научного текста, а для этого необходимо научиться либо относиться к разным явлениям и фактам несколько критически (своя идея – как иная точка зрения), либо научиться увлекаться какими-то известными идеями, которые нуждаются в доработке (идея – как оптимистическая позиция и направленность на дальнейшее совершенствование уже известного). Во-вторых, научиться организовывать свое время, ведь, как известно, свободное (от всяких глупостей) время – важнейшее условие настоящего творчества, для него наконец-то появляется время. Иногда именно на организацию такого времени уходит немалая часть сил и талантов.

Писать следует ясно и понятно, стараясь основные положения формулировать четко и недвусмысленно (чтобы и самому понятно было), а также стремясь структурировать свой текст. Каждый раз надо представлять, что ваш текст будет кто-то читать и ему захочется сориентироваться в нем, быстро находить ответы на интересующие вопросы (заодно представьте себя на месте такого человека). Понятно, что работа, написанная «сплошным текстом» (без заголовков, без выделения крупным шрифтом наиболее важным мест и т. п.), у культурного читателя должна вызывать брезгливость и даже жалость к автору (исключения составляют некоторые древние тексты, когда и жанр был иной и к текстам относились иначе, да и самих текстов было гораздо меньше – не то, что в эпоху «информационного взрыва» и соответствующего «информационного мусора»).

Объем текста и различные оформительские требования во многом зависят от принятых в конкретном учебном заведении порядков.

Доклад - это самостоятельное исследование студентом определенной проблемы, комплекса взаимосвязанных вопросов.

Доклад не должна составляться из фрагментов статей, монографий, пособий. Кроме простого изложения фактов и цитат, в доклад е должно проявляться авторское видение проблемы и ее решения.

Рассмотрим основные этапы подготовки
а студентом.

Выполнение доклада начинается с выбора темы.

Затем студент приходит на первую консультацию к руководителю, которая предусматривает:

- обсуждение цели и задач работы, основных моментов избранной темы;
- консультирование по вопросам подбора литературы;
- составление предварительного плана.

Следующим этапом является работа с литературой. Необходимая литература подбирается студентом самостоятельно.

После подбора литературы целесообразно сделать рабочий вариант плана работы. В нем нужно выделить основные вопросы темы и параграфы, раскрывающие их содержание.

Составленный список литературы и предварительный вариант плана уточняются, согласуются на очередной консультации с руководителем.

Затем начинается следующий этап работы - изучение литературы. Только внимательно читая и конспектируя литературу, можно разобраться в основных вопросах темы и подготовиться к самостоятельному (авторскому) изложению содержания доклада.

Конспектируя первоисточники, необходимо отразить основную идею автора и его позицию по исследуемому вопросу, выявить проблемы и наметить задачи для дальнейшего изучения данных проблем.

Систематизация и анализ изученной литературы по проблеме исследования позволяют студенту написать работу.

Рабочий вариант текста доклада предоставляется руководителю на проверку. На основе рабочего варианта текста руководитель вместе со студентом обсуждает возможности доработки текста, его оформление. После доработки доклад сдается на кафедру для его оценивания руководителем.

Требования к написанию доклада

Написание 1 доклада является обязательным условием выполнения плана СРС по любой дисциплине профессионального цикла.

Тема доклада может быть выбрана студентом из предложенных в рабочей программе или фонде оценочных средств дисциплины, либо определена самостоятельно, исходя из интересов студента (в рамках изучаемой дисциплины). Выбранную тему необходимо согласовать с преподавателем.

Доклад должен быть написан научным языком.

Объем доклада должен составлять 20-25 стр.

Структура доклада:

- Введение (не более 3-4 страниц). Во введении необходимо обосновать выбор темы, ее актуальность, очертить область исследования, объект исследования, основные цели и задачи исследования.
- Основная часть состоит из 2-3 разделов. В них раскрывается суть исследуемой проблемы, проводится обзор мировой литературы и источников Интернет по предмету исследования, в котором дается характеристика степени разработанности проблемы и авторская аналитическая оценка основных теоретических подходов к ее решению. Изложение материала не должно ограничиваться лишь описательным подходом к раскрытию выбранной темы. Оно также должно содержать собственное видение рассматриваемой проблемы и изложение собственной точки зрения на возможные пути ее решения.
- Заключение (1-2 страницы). В заключении кратко излагаются достигнутые при изучении проблемы цели, перспективы развития исследуемого вопроса
- Список использованной литературы (не меньше 10 источников), в алфавитном порядке, оформленный в соответствии с принятыми правилами. В список использованной литературы рекомендуется включать работы отечественных и зарубежных авторов, в том числе статьи, опубликованные в научных журналах в течение последних 3-х лет и ссылки на ресурсы сети Интернет.
- Приложение (при необходимости).

Требования к оформлению:

- текст с одной стороны листа;
- шрифт Times New Roman;
- кегль шрифта 14;
- межстрочное расстояние 1,5;
- поля: сверху 2,5 см, снизу – 2,5 см, слева - 3 см, справа 1,5 см;
- доклад должен быть представлен в сброшюрованном виде.

Порядок защиты доклада:

Защита доклада проводится на практических занятиях, после окончания работы студента над ним и исправления всех недочетов, выявленных преподавателем в ходе консультаций. На защиту доклада отводится 5-7 минут времени, в ходе которого студент

должен показать свободное владение материалом по заявленной теме. При защите доклада приветствуется использование мультимедиа-презентации.

Оценка доклада

Доклад оценивается по следующим критериям:

- соблюдение требований к его оформлению;
- необходимость и достаточность для раскрытия темы приведенной в тексте доклада информации;
- умение студента свободно излагать основные идеи, отраженные в докладе;
- способность студента понять суть задаваемых преподавателем и сокурсниками вопросов и сформулировать точные ответы на них.

Критерии оценки:

Оценка «отлично» выставляется студенту, если в докладе студент исчерпывающе, последовательно, четко и логически стройно излагает материал; свободно справляется с задачами, вопросами и другими видами применения знаний; использует для написания доклада современные научные материалы; анализирует полученную информацию; проявляет самостоятельность при написании доклада.

Оценка «хорошо» выставляется студенту, если качество выполнения доклада достаточно высокое. Студент твердо знает материал, грамотно и по существу излагает его, не допуская существенных неточностей в ответе на вопросы по теме доклада.

Оценка «удовлетворительно» выставляется студенту, если материал доклада излагается частично, но пробелы не носят существенного характера, студент допускает неточности и ошибки при защите доклада, дает недостаточно правильные формулировки, наблюдаются нарушения логической последовательности в изложении материала.

Оценка «неудовлетворительно» выставляется студенту, если он не подготовил доклад или допустил существенные ошибки. Студент неуверенно излагает материал доклада, не отвечает на вопросы преподавателя.

4.5. Методические рекомендации по выполнению исследовательских проектов

Исследовательская проектная работа – это групповая работа, для выполнения которой необходим выбор и приложение научной методики к поставленной задаче, получение собственного теоретического или экспериментального материала, на основании которого необходимо провести анализ и сделать выводы об исследуемом явлении. Выполнение проекта – это всегда коллективная, творческая практическая работа, предназначенная для получения определенного продукта или научно-технического результата. Такая работа подразумевает четкое, однозначное формирование поставленной задачи, определение сроков выполнения намеченного, определение требований к разрабатываемому объекту.

Выполнение 1 группового проекта является обязательным условием выполнения самостоятельной работы по любой дисциплине профессионального цикла. Тема проектного задания может быть выбрана студентом из предложенных в рабочей программе или фонде оценочных средств дисциплины, либо определена самостоятельно, исходя из интересов студента (в рамках изучаемой дисциплины). Выбранную тему необходимо согласовать с преподавателем.

Требования по выполнению и оформлению проекта

При выполнении проекта приветствуется работа в группе (2-3 человека). Проект – это исследовательская работа, в ходе которой студенты должны продемонстрировать владение навыками научного исследования, умения проводить анализ, обобщать информацию, делать выводы, предлагать свои решения проблемы, рассматриваемой в проекте.

При подготовке материалов проекта студенты должны продемонстрировать владение современными методами компьютерной обработки данных.

Критерии оценки работы участника проекта.

Для каждого из участников проекта оцениваются:

- профессиональные теоретические знания в соответствующей области;
- умение работать со справочной и научной литературой, осуществлять поиск необходимой информации в Интернет;
- умение работать с техническими средствами;
- умение пользоваться соответствующими выполняемому проекту информационными технологиями;
- умение готовить материалы проекта для презентации: составлять и редактировать тексты, формировать презентацию проекта;
- умение работать в команде;
- умение публично представлять результаты собственной деятельности;
- коммуникабельность, инициативность, творческие способности.

Критерии выставления оценки участникам проекта

Оценка	Профессиональные компетенции	Компетенции, связанные с использованием соответствующих выполняемому проекту технических средств и информационных технологий	Иные универсальные компетенции (коммуникабельность, инициативность, умение работать в «команде», управленческие навыки и т.д.)	Отчетность
«Отлично»	Работа выполнена на высоком профессиональном уровне. Представленный материал в основном фактически верен, допускаются негрубые фактические неточности. Студент свободно отвечает на вопросы, связанные с проектом.	Технические средства и информационные технологии освоены и использованы для реализации проекта полностью	Студент проявил инициативу, творческий подход, способность к выполнению сложных заданий, навыки работы в коллективе, организационные способности.	Проект представлен полностью и в срок.
«Хорошо»	Работа выполнена на достаточно высоком профессиональном уровне. Допущено до 4–5 фактических ошибок. Студент отвечает на вопросы, связанные с проектом, но	Обнаруживаются некоторые ошибки в использовании соответствующих технических средств и информационных технологий	Студент достаточно полно, но без инициативы и творческих находок выполнил возложенные на него задачи.	Проект представлен достаточно полно и в срок, но с некоторыми недоработками.

Оценка	Профессиональные компетенции	Компетенции, связанные с использованием соответствующих выполняемому проекту технических средств и информационных технологий	Иные универсальные компетенции (коммуникабельность, инициативность, умение работать в «команде», управленческие навыки и т.д.)	Отчетность
	недостаточно полно.			
«Удовлетворительно»	Уровень недостаточно высок. Допущено до 8 фактических ошибок. Студент может ответить лишь на некоторые из заданных вопросов, связанных с проектом.	Обнаруживает недостаточное владение навыками работы с техническими средствами и соответствующим и информационным и технологиями	Студент выполнил большую часть возложенной на него работы.	Проект сдан со значительным опозданием (более недели) и не полностью
«Неудовлетворительно»	Работа не выполнена или выполнена на низком уровне. Допущено более 8 фактических ошибок. Ответы на связанные с проектом вопросы обнаруживают непонимание предмета и отсутствие ориентации в материале проекта.	Навыков работы с техническими средствами нет, информационные технологии не освоены	Студент практически не работал, не выполнил свои задачи или выполнил лишь отдельные не существенные поручения в групповом проекте.	Проект не сдан.

Студенты должны: защитить проект в режиме презентации, предъявить файлы выполненного проекта, уметь рассказать о технологиях, использованных ими при выполнении проекта, дать оценку работы каждого члена группы (*если проект групповой*).

4.6. Методические рекомендации по подготовке к экзаменам и зачетам

Изучение многих общепрофессиональных и специальных дисциплин завершается экзаменом. Подготовка к экзамену способствует закреплению, углублению и обобщению знаний, получаемых, в процессе обучения, а также применению их к решению практических задач. Готовясь к экзамену, студент ликвидирует имеющиеся пробелы в знаниях, углубляет, систематизирует и упорядочивает свои знания. На экзамене студент демонстрирует то, что он приобрел в процессе обучения по конкретной учебной дисциплине.

Экзаменационная сессия - это серия экзаменов, установленных учебным планом. Между экзаменами интервал 3-4 дня. Не следует думать, что 3-4 дня достаточно для успешной подготовки к экзаменам.

В эти 3-4 дня нужно систематизировать уже имеющиеся знания. На консультации перед экзаменом студентов познакомят с основными требованиями, ответят на возникшие у них вопросы. Поэтому посещение консультаций обязательно.

Требования к организации подготовки к экзаменам те же, что и при занятиях в течение семестра, но соблюдаться они должны более строго. Во-первых, очень важно соблюдение режима дня; сон не менее 8 часов в сутки, занятия заканчиваются не позднее, чем за 2-3 часа до сна. Оптимальное время занятий - утренние и дневные часы. В перерывах между занятиями рекомендуются прогулки на свежем воздухе, неустойчивые занятия спортом. Во-вторых, наличие хороших собственных конспектов лекций. Даже в том случае, если была пропущена какая-либо лекция, необходимо во время ее восстановить (переписать ее на кафедре), обдумать, снять возникшие вопросы для того, чтобы запоминание материала было осознанным. В-третьих, при подготовке к экзаменам у студента должен быть хороший учебник или конспект литературы, прочитанной по указанию преподавателя в течение семестра. Здесь можно эффективно использовать листы опорных сигналов.

Вначале следует просмотреть весь материал по сдаваемой дисциплине, отметить для себя трудные вопросы. Обязательно в них разобраться. В заключение еще раз целесообразно повторить основные положения, используя при этом листы опорных сигналов.

Систематическая подготовка к занятиям в течение семестра позволит использовать время экзаменационной сессии для систематизации знаний.

Контроль самостоятельной работы студентов

Контроль самостоятельной работы проводится преподавателем в аудитории.

Предусмотрены следующие виды контроля: собеседование, оценка доклада, оценка презентации, оценка участия в круглом столе, оценка выполнения проекта.

Подробные критерии оценивания компетенций приведены в Фонде оценочных средств для проведения текущей и промежуточной аттестации.

Список литературы для выполнения СРС

Перечень основной литературы:

1. Павлова, Е. А. Технологии разработки современных информационных систем на платформе Microsoft .NET : учебное пособие / Павлова Е. А. - Москва : Интернет-Университет Информационных Технологий (ИНТУИТ), 2019. - 128 с. - Книга находится в базовой версии ЭБС IPRbooks. - ISBN 978-5-9963-0003-7, экземпляров неограничено.
2. Марченко, А.Л. Основы программирования на C# 2.0 Электронный ресурс : учебное пособие / А.Л. Марченко. - Основы программирования на C# 2.0, 2020-07-28. - Москва, Саратов : Интернет-Университет Информационных Технологий (ИНТУИТ), Вузовское образование, 2019. - 552 с. - Книга находится в базовой версии ЭБС IPRbooks. - ISBN 978-5-4487-0084-2, экземпляров неограничено.

Перечень дополнительной литературы:

1. Кариев, Ч.А. Разработка Windows-приложений на основе Visual C# Электронный ресурс : учебное пособие / Ч.А. Кариев. - Разработка Windows-приложений на основе

Visual C#, 2020-11-14. - Москва, Саратов : Интернет-Университет Информационных Технологий (ИНТУИТ), Вузовское образование, 2019. - 768 с. - Книга находится в базовой версии ЭБС IPRbooks. - ISBN 978-5-4487-0146-7, экземпляров неограничено.

2. Чеповский, А. Common Intermediate Language и системное программирование в Microsoft .NET / А. Чеповский ; А. Макаров ; С. Скоробогатов. - 2-е изд., исправ. - Москва : Национальный Открытый Университет «ИНТУИТ», 2019. - 399 с.

Методическая литература:

1. Методические рекомендации для самостоятельной работы студентов по дисциплине «Современные технологии программирования»
2. Методические указания к лабораторным работам по дисциплине «Современные технологии программирования»

Интернет-ресурсы:

1. <http://el.ncfu.ru/> – система управления обучением ФГАОУ ВО СКФУ.
- Дистанционная поддержка дисциплины «Современные технологии программирования».
2. <http://www.un.org> - Сайт ООН Информационно-коммуникационные технологии.
 3. <http://www.intuit.ru> – Интернет-Университет Компьютерных технологий.