

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

Методические указания
по дисциплине
**«Современные тенденции развития информационных
систем»**

для студентов направления подготовки:
09.04.02 «Информационные системы и технологии»
Управление данными

Ставрополь

Оглавление

Введение.....	4
Лабораторная работа 1.....	5
Особенности интерфейсов для смартфонов. Принципы юзабилити	5
Лабораторная работа 2.....	45
Основы тестирования и отладки приложений на смартфоне	45
Лабораторная работа 3.....	57
Портирование приложений с использованием Intel XDK	57
Лабораторная работа 4.....	63
Использование мобильной связи в приложениях для смартфона.....	63
Лабораторная работа 5.....	74
Управление воспроизведением аудио	74
Лабораторная работа 6.....	97
Установка и настройка среды программирования ADT Bundle.....	97
Лабораторная работа 7	118
Основные этапы разработки приложения с использованием Android IDE	118
Лабораторная работа 8.....	143
Основы разработки интерфейсов мобильных приложений.....	143
Лабораторная работа 9.....	183
Создание многоэкранного приложения.....	183
Лабораторная работа 10	203
Демонстрации распознавания стандартных жестов.....	203
Лабораторная работа 11	212
Принципы работы с жестами вводимыми пользователями.....	212
Лабораторная работа 12	222
Многооконное приложение	222
Лабораторная работа 13.....	250
Геолокационные возможности.....	250
Лабораторная работа 14	261
Использование сторонних библиотек.....	261
Лабораторная работа 15.....	278
Работа с базами данных в Android	278

Лабораторная работа 16	288
Установка и настройка среды программирования Intel XDK. Создание приложений.....	288
Рекомендуемая литература.....	308

Введение

Целью изучения дисциплины «Современные тенденции развития информационных систем» является формирование представлений о современных технологиях программирования приложений для мобильных устройств, формирование набора профессиональных компетенций будущего магистра.

Задачи дисциплины:

- изучение базового устройства популярных мобильных платформ;
- изучение основных этапов жизненного цикла информационной системы для мобильных устройств;
- изучение технологии выбора современных операционных сред и информационно-коммуникационных технологий при проектировании, конструировании и отладке программных средств для мобильных устройств;
- овладение практическими навыками анализа рынка программно-технических средств, информационных продуктов и услуг для решения прикладных задач и создания информационных систем для мобильных устройств;
- получение практических навыков программирования, внедрения, адаптации и настройке мобильных гаджетов, пользовательских интерфейсов и сервисов под OS Android и Windows Phone.

ЛАБОРАТОРНАЯ РАБОТА 1.

Особенности интерфейсов для смартфонов. Принципы юзабилити

Цель работы: освоить основные принципы проектирования пользовательских интерфейсов и научиться создавать качественный интерфейс, отвечающий целям и потребностям пользователей.

Теоретическое обоснование

Создание качественного интерфейса: принципы и шаблоны

1 Принципы проектирования взаимодействия

Принципы проектирования взаимодействия – это рекомендации, касающиеся *поведения, формы и содержания* продукта. Они поддерживают проектирование такого поведения продуктов, которое служит потребностям и целям пользователей. Эти принципы представляют собой набор правил, которые основаны на ценностях проектировщиков, и на их опыте, связанном с воплощением этих ценностей в жизнь. В основе таких ценностей – мысль о том, что технология должна служить человеку (а не наоборот) и что опыт общения человека с технологией должен складываться согласно возможностям человеческого восприятия и познания.

Принципы применяются на всем протяжении процесса проектирования. Они помогают преобразовывать задачи и требования, возникающие в ходе разработки сценариев, в поведенческие реакции интерфейса.

Принципы проектирования действуют на нескольких уровнях детализации – от общего проектирования взаимодействия до конкретики интерфейса. Можно выделить следующие категории:

1. Ценности проектирования служат отправной точкой для принципов более низкого уровня, перечисленных ниже.
2. Концептуальные принципы помогают определять сущность продукта.
3. Поведенческие принципы описывают, как продукт должен себя вести – в целом и в конкретных ситуациях.
4. Интерфейсные принципы описывают эффективные стратегии визуального взаимодействия информационных аспектов интерфейса.

Большинство принципов проектирования взаимодействия и визуального

дизайна не привязаны к конкретной платформе. Однако мобильные устройства требуют особых соображений, связанных с различными ограничениями (размер экрана, способы ввода и т.д.). Одно из главных назначений принципов – оптимизировать опыт пользователя. В целях оптимизации необходимо сокращать следующие виды работ:

Когнитивная работа – понимание поведения продукта.

1. Мнемоническая работа – запоминание поведения продукта, паролей, названий и расположения элементов управления.

2. Работа зрения – поиск стартовой точки на экране, поиск одного объекта среди многих, расшифровка визуальной планировки, выявление различий между элементами интерфейса.

3. Физическая работа – использование жестов (нажатие, перемещение по экрану, прокрутка, смахивание и т.д.), переключение между режимами ввода, количество кликов для осуществления навигации.

2 Ценности проектирования

Принципы – это правила, ведущие к действиям и обычно опирающиеся на ряд ценностей и убеждений. Приведенный ниже набор ценностей создали Роберт Рейман, Хью Дабберли (Hugh Dubberly), Ким Гудвин, Дэвид Фор (David Fore) и Джонатан Корман (Jonathan Korman). Он применим к любой дисциплине проектирования, которая служит потребностям человека.

Задача проектировщиков взаимодействия – создавать такие проектные решения, которые:

1. Этичны [тактичны, заботливы]: не причиняют вреда и улучшают положение человека.

2. Прагматичны [жизнеспособны, осуществимы]: помогают организации, внедряющей ваши проектные решения, достигать своих целей; учитывают требования бизнеса и технические требования.

3. Целенаправленны [полезны, применимы]: помогают пользователям решать их задачи и достигать целей, учитывают контексты и возможности пользователей.

4. Элегантны [эффективны, искусны, вызывают эмоции]: представляют

собой простые, но полноценные решения, обладают внутренней (самоочевидной, понятной) целостностью, учитывают и пробуждают эмоции и познавательные процессы.

3 Проектирование этичного взаимодействия

С этическими вопросами проектировщики взаимодействия сталкиваются, когда их просят спроектировать систему, оказывающую серьезное влияние на жизнь людей. Это может быть непосредственное воздействие на пользователей системы или косвенное влияние на людей, жизнь которых каким-либо образом оказывается затронута системой.

Для проектировщиков взаимодействия этот вопрос может быть особенно животрепещущим, поскольку результаты их работы не просто убедительная передача определенной стратегии продвижения продукта. От их работы в действительности зависит реализация этой стратегии, или собственно создание продукта. Интерактивные продукты обладают возможностью изменять мир, и проектировщики должны стремиться к тому, чтобы это были изменения к лучшему. Относительно легко спроектировать систему, которая хорошо относится к пользователям, однако вычислить, какое косвенное влияние система оказывает на других людей, гораздо сложнее.

Не навреди – один из важнейших этических принципов проектирования взаимодействия. Продукты не должны никому причинять вреда – или, принимая во внимание сложность реального мира, должны минимизировать ущерб.

Продукт не должен причинять вред личности. Пользователь даже случайно не должен испытывать чувство раздражения, унижения, замешательства, страха. Если это произойдет, пользователь скорее всего откажется от использования программы. Предупреждение этой разновидности вреда требует глубокого понимания аудитории, особое внимание следует обратить в случае локализации продукта для стран с сильно отличающейся культурой. Продукт не должен причинять физический вред. В контексте мобильной разработки в первую очередь это означает, что пользователь не должен быстро утомляться при использовании приложения. Иными словами, следует очень хорошо продумать

эргономику интерфейса с целью минимизации работы пользователя.

Второй принцип этичного взаимодействия состоит в улучшении положения человека.

Чтобы результатом проектирования стало действительно этичное взаимодействие, мало просто не вредить – требуется способствовать. Приведем примеры улучшений, которым в широком смысле могут содействовать интерактивные системы:

- улучшение понимания (индивидуального, социального, культурного);
- повышение эффективности/действенности отдельных личностей и групп;
- совершенствование коммуникаций – как между отдельными личностями, так и между группами людей;
- снижение социокультурной напряженности между личностями и группами;
- умножение справедливости (финансовой, социальной, правовой);
- сглаживание культурных противоречий путем стимулирования общественной толерантности.

Проектировщикам, начинающим работу над новым проектом, необходимо в глубине души осознавать эти глобальные вопросы.

4 Проектирование целенаправленного взаимодействия

Целенаправленность – это не только понимание целей пользователей, но и осознание их ограничений. В этом смысле персонажи служат качественной меркой, поскольку шаблоны поведения, которые вы сможете наблюдать в ходе исследований и при создании персонажей, дадут вам хорошее представление о сильных и слабых сторонах пользователей. Целеориентированное проектирование помогает проектировщикам создавать продукты, которые поддерживают пользователей в том, в чем они слабы, и делают их более производительными в том, в чем они сильны.

5 Проектирование прагматичного взаимодействия

Чтобы иметь ценность, проектирование должно воплощаться в продукте. Будучи созданным, продукт должен увидеть свет. А увидев свет, он должен

приносить прибыль создателям.

Крайне важно, чтобы цели бизнеса и технические требования учитывались в ходе проектирования. Это не означает, что проектировщики должны принимать как должное все, что говорят заинтересованные лица и разработчики. Необходим живой диалог, в котором участвуют представители бизнеса, инженерная часть команды, проектировщики, позволяющий договориться о том, какие области определения продукта заданы жестко, а какие являются гибкими. Проектирование оказывается наиболее эффективным, когда существует взаимное доверие и уважение между проектировщиками, бизнесменами и инженерами.

6 Проектирование элегантного взаимодействия

Элегантность в словаре определяется как "грациозность и сдержанная красота стиля" или "научная точность, аккуратность и простота". Элегантность в проектировании – или, по меньшей мере, в проектировании взаимодействия – включает оба этих идеала.

Создавай простые, но полноценные решения.

Один из классических элементов качественного дизайна – это экономия формы, достижение большего меньшими затратами. В проектировании мобильных интерфейсов это означает, что интерфейс должен содержать лишь те экраны и элементы, которые требуются для решения поставленной задачи. Это касается и поведения: следует дать пользователю простой набор инструментов, позволяющий ему добиваться великолепных результатов. Такой подход хорошо знаком программистам, которые признают, что лучшие алгоритмы коротки и понятны.

Добивайся внутренней целостности.

Качественно спроектированный продукт оставляет ощущение единого целого, в котором все составные части сбалансированы и гармоничны. Некачественно спроектированный продукт или продукт, вообще не проектировавшийся, обычно создает впечатление, будто его наспех склотили из выбранных случайным образом разрозненных кусков. Часто такой продукт – результат конструирования на основе модели реализации, когда различные

команды разработчиков трудятся над различными модулями интерфейса, не общаясь между собой, или когда продукт проектируется без учета особенностей мобильной платформы. Целеориентированный процесс проектирования, в котором концепция продукта зарождается сразу и полностью на высшем уровне с последующей поэтапной детализацией, дает идеальную среду для создания внутренне целостных проектов.

Учитывай и пробуждай эмоции и познавательные процессы.

Многие проектировщики традиционной школы часто говорят о желаниях и важности желаний при проектировании коммуникаций и продуктов. Однако делая такой сильный упор на одну эмоцию, они временами теряют из виду полную картину. Желание – ограниченная эмоция, если речь идет о продукте, у которого есть определенное назначение, особенно узкопрофессиональное. Пользователь должен получать стимулы и поддержку при работе с продуктом, поэтому вместо желания лучше использовать элегантность.

Шаблоны проектирования взаимодействия

Шаблоны проектирования решают целые классы проблем проектирования, возникающие путем выявления и обобщения ценных проектных находок. Деятельность по формализации знания и фиксации наилучших решений в области проектирования служит многим важным целям:

- сократить время и усилия, затрачиваемые на проектирование в новых проектах;
- повысить качество проектных решений;
- способствовать улучшению коммуникации между проектировщиками и программистами;
- повысить профессиональный уровень проектировщиков.

Шаблоны всегда применяются в рамках некоторого контекста и конструируются так, чтобы быть применимыми в типичных ситуациях, которые имеют схожий контекст использования, схожие ограничения и условия. Описывая шаблон, важно четко задать ситуацию, в которой применимо решение, дать один или несколько конкретных примеров, перечислить абстрактные признаки, характерные для всех примеров, а также рассуждения,

объясняющие, почему решение является хорошим.

Шаблоны проектирования не являются рецептами или готовыми решениями. В своей книге «Designing Interfaces» – объемистом и полезном собрании шаблонов проектирования взаимодействия – Дженифер Тидвелл (Jenifer Tidwell) предостерегает нас: «[Шаблоны] – это не готовые к употреблению компоненты; каждая реализация шаблона немного отличается от всех других».

Типы шаблонов проектирования взаимодействия

Шаблоны проектирования взаимодействия можно выстроить в иерархию. Их можно применять на различных уровнях инфраструктуры интерфейса:

1. Шаблоны позиционирования могут применяться на концептуальном уровне и помогают определить тип продукта в отношении к пользователю.
2. Структурные шаблоны решают проблемы, связанные с управлением отображением информации и функциональных элементов на экране.
3. Поведенческие шаблоны решают широкий спектр проблем, относящихся к конкретным взаимодействиям с теми или иными элементами интерфейса.

Структурные шаблоны являются, по всей видимости, наименее документированными, однако при этом они распространены повсеместно. На рис. 1 представлен один из наиболее широко применяемых высокоуровневых структурных: навигационная панель слева, обзорная панель справа вверху, панель подробностей справа внизу.

Данный шаблон (рис. 1.1) является оптимальным для приложений, в которых пользователю требуется работать с разнообразными объектами, объединять объекты в группы, а также просматривать содержимое или свойства отдельных объектов или документов, не переходя на другой экран.



Рисунок 1.1 – Навигационная схема приложения

Многие программы для электронной почты (в том числе и мобильные) используют этот шаблон (рис. 1.2), а его вариации встречаются в программах для создания и управления информацией, где часто требуется быстрый доступ и управление объектами многих типов.

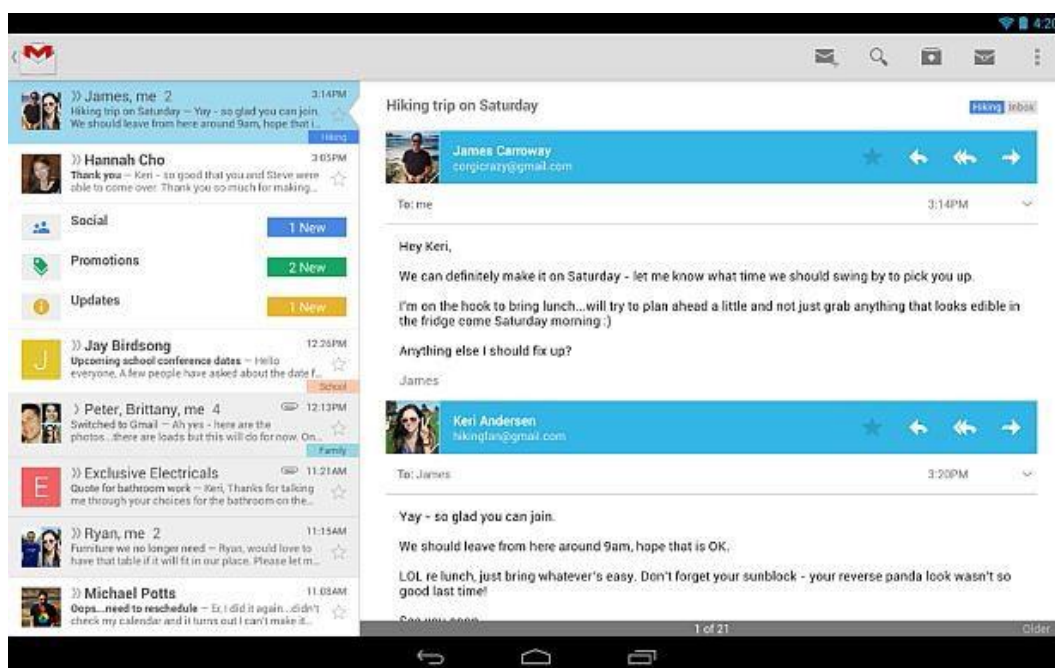


Рисунок 1.2 – Почтовый клиент gmail ориентирован на планшеты

Однако все не так просто. Этот шаблон в чистом виде, безусловно, удобен в случае относительно большого экрана (монитор или планшет), однако на мобильном телефоне он малоприменим. Решением проблемы является отказ от единого экрана: выделив каждому разделу собственную экранную сущность и организовав связь между ними, можно получить высокофункциональное приложение (рис. 1.3).

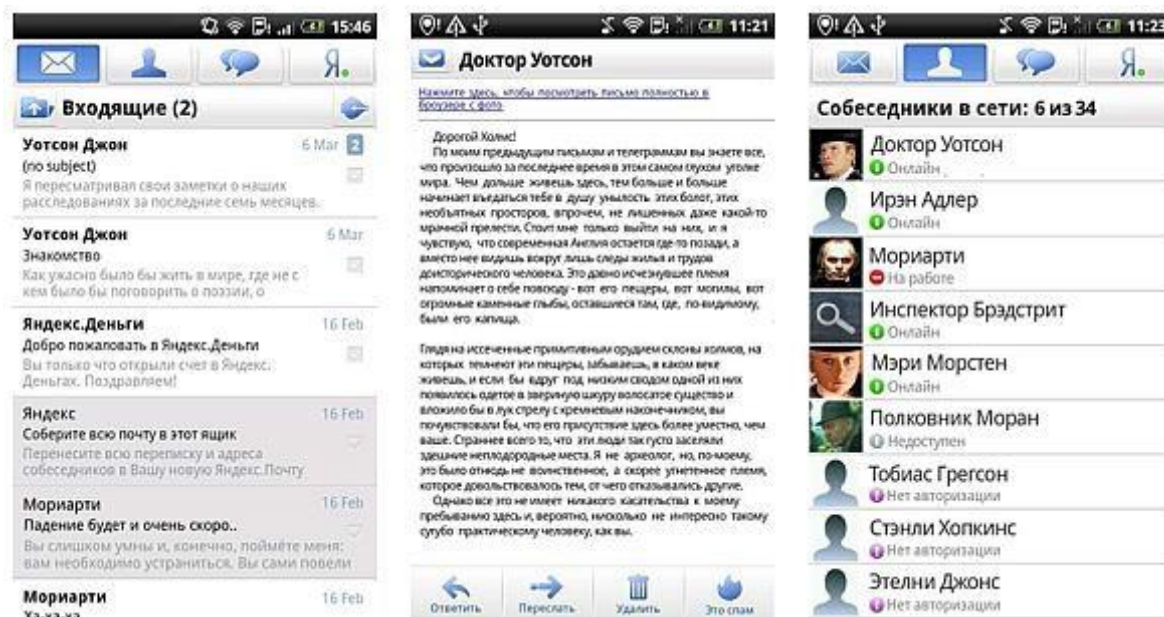


Рисунок 1.3 – Почтовый клиент Яндекс хорошо подходит для телефона

Визуальный дизайн интерфейсов

Силы, вложенные в разработку модели поведения программного продукта, будут потрачены впустую, если вы не сумеете должным образом донести до пользователей принципы этого поведения. В случае мобильных продуктов это делается визуальными средствами – путем отображения объектов на дисплее (в некоторых случаях целесообразно использовать тактильные ощущения от нажатия).

Визуальный дизайн интерфейсов – очень нужная и уникальная дисциплина, которую следует применять в сочетании с проектированием взаимодействия и промышленным дизайном. Она способна серьезно повлиять на эффективность и привлекательность продукта, но для полной реализации этого потенциала нужно не откладывать визуальный дизайн на потом, а сделать его одним из основных инструментов удовлетворения потребностей пользователей

и бизнеса.

Изобразительное искусство, визуальный дизайн интерфейсов и прочие дисциплины дизайна

Художники и визуальные дизайнеры работают с одними и теми же изобразительными средствами, однако их деятельность служит различным целям. Цель художника – создать объект, взгляд на который вызывает эстетический отклик. Изобразительное искусство – способ самовыражения художника. Художник не связан почти никакими ограничениями.

Чем необычнее и своеобразнее продукт его усилий, тем выше он ценится.

Дизайнеры создают объекты, которыми будут пользоваться другие люди. Если говорить о дизайнерах визуальных интерфейсов, то они ищут наилучшее представление, доносящее информацию о поведении программы, в проектировании которой они принимают участие. Придерживаясь целеориентированного подхода, они должны стремиться представлять поведение и информацию в понятном и полезном виде, который поддерживает маркетинговые цели организации и эмоциональные цели персонажей. Разумеется, визуальный дизайн пользовательских интерфейсов не исключает эстетических соображений, но такие соображения не должны выходить за рамки функционального каркаса.

Графический дизайн и пользовательские интерфейсы

Графические дизайнеры обычно очень хорошо разбираются в визуальных аспектах и хуже представляют себе понятия, лежащие в основе поведения программного продукта и взаимодействия с ним. Они способны создавать красивую и адекватную внешность интерфейсов, а кроме того – привносить фирменный стиль во внешний вид и поведение программного продукта. Для таких специалистов дизайн или проектирование интерфейса есть в первую очередь тон, стиль, композиция, которые являются атрибутами бренда, во вторую очередь – прозрачность и понятность информации и лишь затем – передача информации о поведении посредством ожидаемого назначения.

Дизайнерам визуальной части интерфейса необходимы некоторые навыки, которые присущи графическим дизайнерам, но они должны еще

обладать глубоким пониманием и правильным восприятием роли поведения. Их усилия в значительной степени сосредоточены на организационных аспектах проектирования. В центре их внимания находится соответствие между визуальной структурой интерфейса с одной стороны и логической структурой пользовательской ментальной модели и поведения программы – с другой. Кроме того, их заботит вопрос о том, как сообщать пользователю о состояниях программы и что делать с когнитивными аспектами пользовательского восприятия функций.

Визуальный информационный дизайн

Информационные дизайнеры работают над визуализацией данных, содержимого и средств навигации. Усилия информационного дизайнера направлены на то, чтобы представить данные в форме, способствующей их верному истолкованию. Результат достигается через управление визуальной иерархией при помощи таких средств, как цвет, форма, расположение и масштаб. Распространенными объектами информационного дизайна являются всевозможные графики, диаграммы и прочие способы отображения количественной информации.

Чтобы создавать привлекательные и удобные пользовательские интерфейсы, дизайнер интерфейсов должен владеть базовыми визуальными навыками – пониманием цвета, типографики, формы и композиции – и знать, как их можно эффективно применять для передачи поведения и представления информации, для создания настроения и стимулирования физиологических реакций. Дизайнеру интерфейса также требуется глубокое понимание принципов взаимодействия и идиом интерфейса, определяющих поведение продукта.

Строительные блоки визуального дизайна интерфейсов

Дизайн интерфейсов сводится к вопросу о том, как оформить и расположить визуальные элементы таким образом, чтобы внятно отразить поведение и представить информацию.

Каждый элемент визуальной композиции имеет ряд свойств, и сочетание этих свойств придает элементу смысл. Пользователь получает возможность

разобраться в интерфейсе благодаря различным способам приложения этих свойств к каждому из элементов интерфейса. В тех случаях, когда два объекта обладают общими свойствами, пользователь предположит, что эти объекты связаны или похожи. Когда пользователи видят, что свойства отличаются, они предполагают, что объекты не связаны.

Создавая пользовательский интерфейс, проанализируйте перечисленные ниже визуальные свойства каждого элемента или группы элементов. Чтобы создать полезный и привлекательный пользовательский интерфейс, следует тщательно поработать с каждым из этих свойств.

Форма

Форма – главный признак сущности объекта для человека. Мы узнаем объекты по контурам. Если мы увидим на картинке синий ананас, мы его сразу опознаем, потому что мы помним его форму. И лишь потом мы удивимся странному цвету. При этом различение форм требует большей концентрации внимания, чем анализ цвета или размера. Поэтому форма – не лучшее свойство для создания контраста, если требуется привлечь внимание пользователя.

Размер

Более крупные элементы привлекают больше внимания, особенно если они значительно превосходят размерами окружающие элементы. Люди автоматически упорядочивают объекты по размеру и склонны оценивать их по размеру; если у нас есть текст в четырех размерах, предполагается, что относительная важность текста растет вместе с размером и что полужирный текст более важен, чем текст с нормальным начертанием. Таким образом, размер – полезное свойство для обозначения информационных иерархий.

Цвет

Цветовые различия быстро привлекают внимание. В некоторых профессиональных областях цвета имеют конкретные значения, и этим можно пользоваться. Так, для бухгалтера красный цвет – отрицательные результаты, а черный – положительные.

Цвета приобретают смыслы и благодаря социальным контекстам, в которых проходит наше взросление. Например, белый цвет на Западе

ассоциируется с чистотой и миром, а в Азии и арабских странах – с похоронами и смертью. При этом цвет изначально не обладает свойством упорядоченности и не выражается количественно, поэтому далеко не идеален для передачи информации такого рода. Кроме того, не следует делать цвет единственным способом передачи информации, поскольку цветовая слепота встречается довольно часто.

Применяйте цвет с умом. Чтобы создать эффективную визуальную систему, позволяющую пользователю выявлять сходства и различия объектов, используйте ограниченный набор цветов – эффект радуги перегружает восприятие пользователя и ограничивает возможности по передаче ему информации.

Яркость

Понятия темного и светлого обретают смысл преимущественно в контексте яркости фона. На темном фоне темный текст почти не виден, тогда как на светлом он будет резко выделяться. Контрастность люди воспринимают легко и быстро, так что значение яркости может стать хорошим инструментом привлечения внимания к тем элементам, которые требуется подчеркнуть. Значение яркости – также упорядоченная переменная, например, более темные (с более низкой яркостью) цвета на карте легко интерпретируются: они обозначают большие глубины или большие значения других параметров.

Направление

Направление полезно, когда требуется передавать информацию об ориентации (вверх или вниз, вперед или назад). Помните, что восприятие направления может быть затруднено в случае некоторых форм и при малых размерах объектов, поэтому ее лучше использовать в качестве вторичного признака. Так, если требуется показать, что рынок акций пошел вниз, можно использовать направленную вниз стрелку красного цвета.

Текстура

Разумеется, изображенные на экране элементы не обладают настоящей текстурой, но способны создавать ее видимость. Текстура редко бывает полезна для передачи различий или привлечения внимания, поскольку требует

значительной концентрации на деталях. И тем не менее текстура может быть важной подсказкой. Засечки и выпуклости на элементах пользовательского интерфейса обычно указывают, что элемент можно перетаскивать, а фаски или тени у кнопки усиливают ощущение, что ее можно нажать.

Расположение

Расположение – это переменная упорядоченная и выражаемая количественно, а значит, полезная для передачи иерархии. Расположение также может служить средством создания пространственных отношений между объектами на экране и объектами реального мира (например, небо в верхней половине, земля в нижней).

Расположение элементов мобильного приложения очень сильно влияет на удобство использования и зависит от того, как пользователь будет держать устройство.

Принципы визуального дизайна интерфейсов

Человеческий мозг – великолепное устройство распознавания образов. Он справляется со шквалом входных данных, выявляя визуальные закономерности и создавая для наблюдаемых нами объектов систему приоритетов. Именно способность зрительной системы человеческого мозга к сборке частей визуального поля в образы на основании визуальных якорей (подсказок) позволяет нам обрабатывать зрительную информацию столь быстро и эффективно. Процесс создания визуального дизайна интерфейса должен опираться на наши природные способности к обработке визуальной информации, чтобы обеспечить передачу пользователям информации и отражение возможностей и функций программы.

При создании графических интерфейсов следует:

1. Использовать визуальные свойства для группировки элементов и создания четкой иерархии.
2. Создавать визуальную структуру и прокладывать логический маршрут на каждом уровне организации.
3. Использовать целостные, непротиворечивые и соответствующие контексту образы.

4. Интегрировать визуальный стиль с функциональностью осмысленно и последовательно.

5. Избегать визуального «шума» и беспорядка.

Рассмотрим их подробнее.

Использование визуальных свойств для группировки элементов и создания четкой иерархии

Как правило, имеет смысл группировать логические наборы функциональных или информационных элементов посредством визуальных свойств, например цвета или пространственных характеристик. Последовательно применяя эти визуальные свойства в интерфейсе, вы можете создавать шаблонные образы, которые ваши пользователи быстро научатся распознавать. Согласно инструкциям по проектированию дизайна андроидприложений, обычные кнопки должны быть выпуклыми, со скругленными углами, а текстовые поля прямоугольными, обычно подчеркнутые и плоские, при этом активные элементы выделяются цветом (рис. 1.4). Благодаря систематическому применению этого образа невозможно перепутать кнопку и поле ввода, несмотря на некоторые сходства.

Глядя на любой набор визуальных элементов, пользователь бессознательно задается вопросом: «Что здесь представляет интерес?» – и почти сразу же: «Какая связь между этими объектами?» Мы должны стремиться к тому, чтобы интерфейс содержал в себе ответ на оба вопроса.

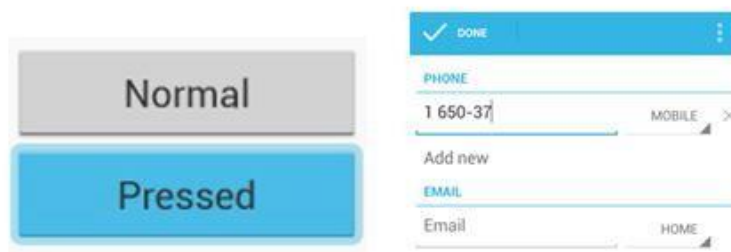


Рисунок 1.4 – Визуализация кнопок и текстовых полей ввода в Android

При создании иерархии необходимо определить, исходя из сценариев, какие функциональные и информационные элементы должны восприниматься пользователями сходу, какие являются вторичными, а какие нужны лишь в

исключительных ситуациях. Такое ранжирование и служит основой для визуальной иерархии.

Чтобы создать видимые различия между уровнями иерархии, используйте цвет, насыщенность, контрастность, размер и положение. Самые важные элементы должны быть более крупными, более ярких цветов, более насыщенными и более контрастными. Их следует располагать над прочими элементами или делать выступающими. Менее важные элементы должны быть менее насыщенными, менее контрастными, более мелкими и плоскими. Нейтральные светлые цвета уводят их на второй план.

Разумеется, настройку этих свойств следует выполнять осторожно. Не следует делать самый важный элемент огромным, красным и выпуклым. Часто бывает достаточно изменить лишь одно из свойств. Если обнаружится, что два элемента различной важности состязаются за внимание пользователя, сделать менее важный менее заметным будет лучшим решением.

Создание четкой визуальной иерархии – одна из сложнейших задач в визуальном дизайне интерфейсов, ее решение требует навыков и таланта. Качественную визуальную иерархию пользователи практически не замечают – а вот ее отсутствие и проистекающая из этого путаница сразу бросаются в глаза.

Чтобы передать связь элементов, вновь обратитесь к сценариям. Необходимо определить не только элементы со сходными функциями, но и элементы, наиболее часто используемые совместно. Совместно используемые элементы обычно следует сгруппировать в пространстве (поместить на отдельный экран), чтобы минимизировать перемещения между ними.

Пространственная группировка объясняет пользователям, каким образом одни задачи, данные и инструменты связаны с другими, и может намекать на правильную последовательность действий. Хорошая группировка посредством расположения принимает во внимание порядок задач и подзадач и движение взгляда по экрану.

Элементы, расположенные рядом, как правило, связаны друг с другом. Если необходимо создать группировку, удобно реализовывать ее посредством расстояний. Элементы, разделенные большими расстояниями, можно

группировать посредством общих визуальных свойств.

Определившись с группами и визуальными особенностями этих групп, начинайте подстраивать контраст между группами – подчеркивая или, наоборот, затеняя группы сообразно их важности в текущем контексте.

Есть хороший способ убедиться, что визуальный дизайн эффективно задействует иерархию и отношения, – дизайнеры называют этот прием тестом с прищуриванием (squint test). Закройте один глаз и посмотрите на экран прищуренным вторым глазом. Обратите внимание на то, какие элементы слишком выпирают, какие стали нечеткими, а какие объединились в группы. Эта процедура часто вскрывает не замеченные ранее проблемы в композиции интерфейса.

Визуальная структура и логические маршруты

Интерфейсы удобно представлять себе состоящими из визуальных и интерактивных элементов, объединяемых в группы с помощью панелей, которые, в свою очередь, можно группировать в экраны. Такая группировка может проводиться посредством распределения в пространстве или при помощи общих визуальных свойств. Крайне важно сохранять прозрачную визуальную структуру, чтобы пользователь мог легко переходить от одной части интерфейса к другой в соответствии со своим рабочим процессом.

Опишем ряд важных свойств, помогающих задать четкую визуальную структуру.

Выравнивание

Выравнивание визуальных элементов – один из главных приемов, позволяющих дизайнеру представить продукт пользователям в систематизированном и упорядоченном виде.

Сгруппированные элементы следует выравнивать как по горизонтали, так и по вертикали (рис. 1.5).

В общем случае каждый элемент на экране следует выровнять по максимально возможному числу других элементов. Отказ от выравнивания двух элементов или двух групп элементов должен быть осознанным: это допустимо только для достижения конкретного разделяющего эффекта. В числе прочего

дизайнерам следует обращать внимание на:

1. Выравнивание подписей. Подписи для элементов управления, расположенные друг над другом, должны быть выровнены по общей границе.

2. Выравнивание внутри группы функциональных элементов. Группа связанных флажков, вариантов выбора или текстовых полей должна подчиняться выравниванию стандартной сетки.

3. Выравнивание элементов, разнесенных по группам и панелям. Группы элементов управления и прочие объекты на экране везде, где это возможно, должны быть привязаны всё к той же сетке (рис. 1.5).

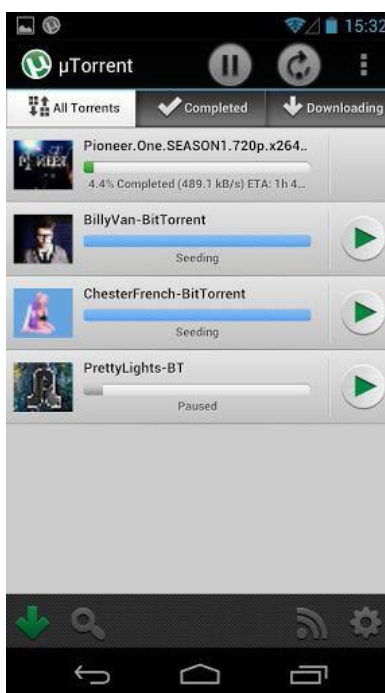


Рисунок 1.5 – Приложение uTorrent эффективно использует выравнивание по композиционной сетке. Текст и функциональные элементы четко выравниваются по сетке с фиксированным шагом

Сетка

Сетка – один из самых мощных инструментов визуального дизайнера. Сетка обеспечивает однородность и последовательность структуры композиции. После того как проектировщики взаимодействия определили общую инфраструктуру приложения и элементов его пользовательского интерфейса, дизайнеры интерфейса должны организовать композицию в структуру в виде сетки, которая будет должным образом подчеркивать важные элементы и

структуры и оставлять жизненное пространство для менее важных элементов и элементов более низкого уровня.

Как правило, сетка делит экран на несколько крупных горизонтальных и вертикальных областей. Качественно спроектированная сетка задействует понятие шага, то есть минимального расстояния между элементами. К примеру, если шаг сетки составляет четыре пиксела, все расстояния между элементами и группами должны быть кратны четырем.

В идеальном случае сетка должна задавать и пропорции различных областей экрана. Такие отношения обычно выражаются дробями. Среди распространенных дробей – прославленное «золотое сечение» (равное примерно 1,62), которое часто встречается в природе и считается особенно приятным для человеческого глаза; величина, обратная квадратному корню из двух (примерно 1:1,41), которая является основой международного стандарта размера бумаги (например, листа А4). В программировании для мобильных устройств не следует полагаться на соотношение сторон дисплеев, так как для устройств на Android не существует единого стандарта размера экранов.

Использование сетки в визуальном дизайне интерфейсов дает ряд преимуществ:

1. Удобство применения. Поскольку сетка делает расположение элементов единообразным, пользователи быстро приобретают навыки поиска нужных элементов в интерфейсе. Последовательность в расположении элементов и выборе расстояний между ними облегчает работу механизмов визуальной обработки в мозгу человека. Качественно спроектированная сетка упрощает восприятие экрана.

2. Эстетическая привлекательность. Аккуратно применяя сетку и выбирая подходящие соотношения между различными областями экрана, дизайнер может создать ощущение порядка, который удобен пользователям и стимулирует их работу с продуктом.

3. Эффективность. Создание сетки и включение ее в процесс на ранних этапах детализации проектных решений сокращает число итераций и действий по «доводке» интерфейса. Качественная и явно обозначенная сетка закладывает

основу для легко модифицируемого и расширяемого дизайна, позволяя разработчикам находить хорошие композиционные решения.

Логические маршруты

Композиция должна не только в точности следовать сетке, но и структурировать эффективный логический маршрут через интерфейс для пользователей, принимая во внимание тот факт, что (в случае западных языков) взгляд движется сверху вниз и слева направо (рис. 1.6).

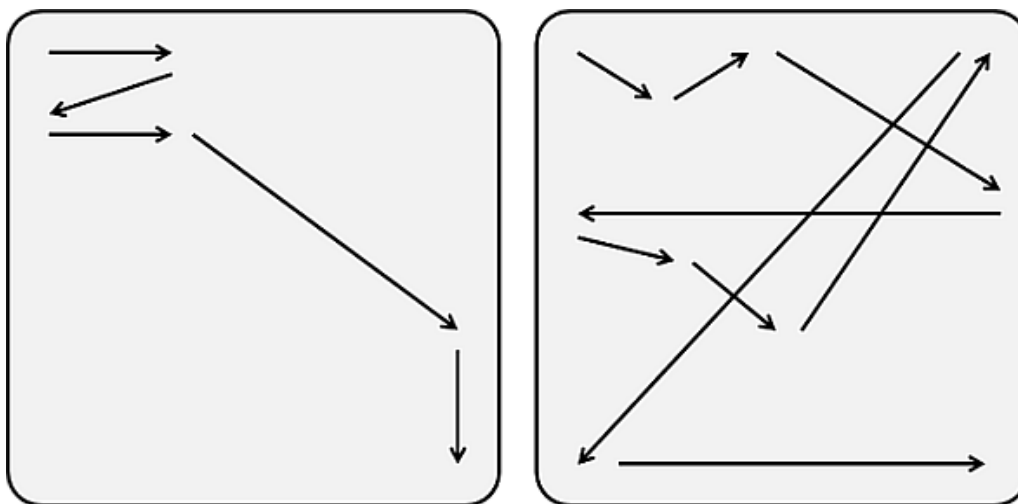


Рисунок 1.6 – Слева представлен хороший логический маршрут: движение взгляда и маршрут в интерфейсе совпадают. Справа – неудобный логический маршрут: все разбросано по экрану.

Симметрия – полезное средство организации интерфейса с точки зрения достижения визуального равновесия. Несимметричные интерфейсы обычно выглядят так, словно вот-вот завалятся на один бок. Опытные дизайнеры способны достигать асимметричного равновесия, управляя визуальным весом отдельных элементов. Тест с прищуриванием позволяет проверить сбалансированность интерфейса.

В интерфейсах чаще всего применяют два типа симметрии: вертикальная осевая симметрия (симметрия относительно вертикальной линии, проведенной через центр группы элементов) и диагональная осевая симметрия (симметрия относительно диагонали). В большинстве приложений присутствует симметрия одного из этих типов.

Целостные, непротиворечивые и соответствующие контексту образы

Применение пиктограмм и других наглядных элементов способно помочь пользователю разобраться в интерфейсе или, наоборот, вызвать раздражение, запутать или даже оскорбить, если пиктограммы выбраны неудачно. Очень важно, чтобы дизайнеры понимали, какое послание должна передать пользователю программа и какое послание он рассчитывает получить. Хорошее понимание персонажей и их ментальных моделей, как правило, дает прочный фундамент для текстового и визуального языков интерфейса. Играть роль и культурные аспекты.

Дизайнеры должны отдавать себе отчет в том, что цвет в различных культурах имеет различные значения (в Китае красный цвет не считается предупреждающим), равно как жесты и символы. Кроме того, следует знать цветовые коды, принятые в разных областях человеческой деятельности. Например, в больницах желтый цвет обозначает радиацию, а красный, как правило, используется в ситуациях, угрожающих жизни. На терминале фондового рынка красный – сигнал продавать. Прежде чем вы возьметесь за дело, убедитесь, что понимаете визуальный язык, принятый в сфере деятельности и культурном окружении ваших пользователей.

Визуальные элементы должны быть частью глобального визуального языка связующего компоненты интерфейса. Это означает, что логически сходные элементы должны иметь общие визуальные свойства, такие как положение, размер, толщина линий и общий стиль, различаясь лишь в том, что подчеркивает их значение. Идея состоит в создании связной системы элементов. Интерфейс, в котором эта цель достигнута, кажется безупречным: ни один элемент не выглядит так, словно его добавили в последнюю минуту.

Помимо функциональной ценности пиктограммы способны играть значительную роль в передаче атрибутов бренда. Яркие «мультяшные» пиктограммы могут попасть в точку, если вы проектируете игру для детей, а выверенные сдержанные пиктограммы больше подойдут для бизнес-приложения. Независимо от того, какой стиль выбран, соблюдайте преемственность – если на одних пиктограммах линии – жирные и черные со скругленными углами, а на других – тонкие ломаные, визуальный стиль

«развалится».

Пиктограммы

Дизайн и рисование пиктограмм – совершенно самостоятельная область. Создание понятных изображений при низком разрешении требует значительного времени и практики – лучше делегировать эту задачу опытным дизайнерам. Основная трудность заключается в представлении абстрактных понятий на пиктографическом, визуальном языке. В таких случаях вместо образов, в которых никто не разберется, лучше опираться на идиомы и снабжать их всплывающими подсказками или подписями.

В случае очевидных и конкретных функций придерживайтесь следующих правил:

1. Помещайте на пиктограмму как функцию, так и объект, чтобы облегчить пользователю понимание пиктограммы. Глагол в сочетании с существительным понять легче, чем глагол сам по себе.

2. Остерегайтесь метафор и представлений, которые могут не обладать требуемыми значениями в глазах целевой аудитории. К примеру, поднятый вверх большой палец может показаться подходящей пиктограммой для обозначения положительной реакции, однако на Среднем Востоке и в других культурах этот жест является оскорбительным в любом приложении, предназначенном для глобального рынка, его следует избегать.

1. Визуально группируйте взаимосвязанные функции – либо пространственно, либо, если это невозможно, с помощью цвета и иных изобразительных средств.

2. Создавайте простые пиктограммы; избегайте лишних деталей.

3. Используйте элементы повторно везде, где это возможно, чтобы пользователь изучил их однажды и навсегда.

Создание уникальных символов для различных типов объектов в интерфейсе также помогает пользователю лучше понимать интерфейс. Такие визуальные маркеры позволяют пользователю ориентироваться среди объектов быстрее, чем просто подписи. Чтобы установить связь между символом и объектом, используйте символ всякий раз, когда объект появляется на экране.

Дизайнер интерфейсов должен позаботиться также о том, чтобы символы, представляющие объекты различных типов, зрительно различались. Вычленив конкретную пиктограмму на экране, переполненном почти одинаковыми пиктограммами, столь же трудно, как и найти конкретное слово на экране, заполненном текстом.

Предварительный просмотр

Вместо того чтобы описывать результаты работы функций в интерфейсе исключительно словами (или, что еще хуже, не давать никаких описаний), показывайте пользователю, какими будут эти результаты. Для этой цели применяйте визуальные элементы. В дополнение к тексту, описывающему параметр или состояние, поместите картинку или диаграмму, описывающую поведение. Хотя визуализация, как правило, требует дополнительного места на экране, ее способность ясно передавать смысл стоит потраченных на это пикселей. Например, функция меню «Эффекты» в Android-приложении Adobe Photoshop показывает, как будет выглядеть документ, если к нему применить один из эффектов (рис. 1.7).

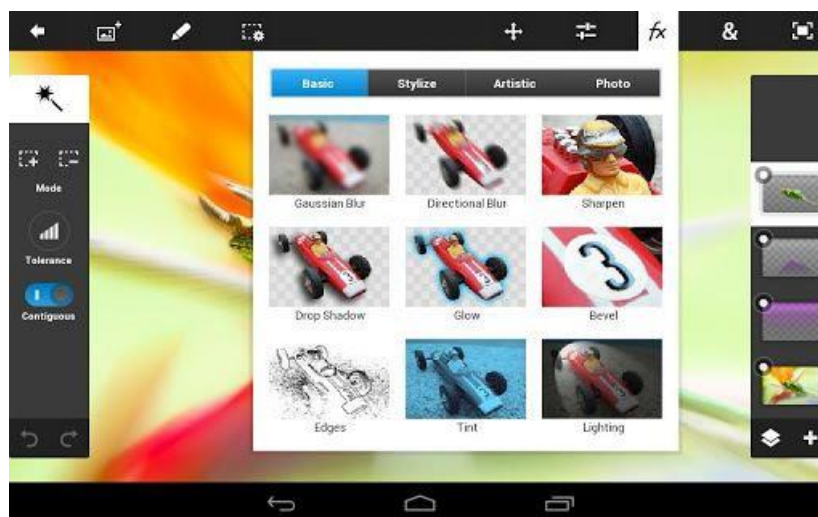


Рисунок 1.7 – Эффекты реализованы таким образом, чтобы пользователь легко мог понять, к каким результатам приводит тот или иной эффект

Интеграция визуального стиля с функциональностью

Если дизайнеры интерфейсов решили придерживаться в интерфейсе

определенного стиля, это следует делать глобально. Каждый аспект интерфейса должен быть проанализирован с точки зрения соответствия стилю; нельзя ограничиваться лишь некоторыми визуальными элементами. Необходимо убедиться, что функциональные аспекты графического интерфейса программы находятся в полной гармонии с общим визуальным стилем бренда вашей продукции. Поведение программы – часть бренда, и опыт взаимодействия пользователя с продуктом должен отражать баланс формы, содержания и поведения.

Для многих заинтересованных лиц (владельцев проекта) визуальный стиль – очень притягательная область, однако стилизованные элементы интерфейса следует держать под неусыпным контролем. Движущими силами в разработке визуального стиля должны быть базовые формы, поведение и ожидаемое назначение, тогда как соображения эстетического плана не должны мешать передаче смысла в интерфейсе и взаимодействию пользователя с продуктом.

При этом образовательные и развлекательные приложения оставляют больше возможностей для экспериментов со стилем. В таких случаях и визуальное впечатление, создаваемое интерфейсом, и содержимое приложения оказывают влияние на то удовольствие, которое пользователь получает от взаимодействия с приложением; это служит весомым аргументом в пользу более тесной связи оформления управляющих элементов с содержимым. Но и здесь следует помнить об ожидаемом назначении, позволяющем пользователям быстро получать доступ к информации.

Избегайте визуального шума и беспорядка

Визуальный шум в интерфейсе возникает из-за излишних графических элементов, отвлекающих внимание от элементов, непосредственно связанных с функциональностью и поведением программы. Визуальный шум возникает благодаря необязательным декоративным и излишне «объемным» элементам, злоупотреблению линиями и иными разделителями между элементами управления, неуместному или излишне интенсивному использованию цвета, текстур и контраста. Причудливые, запутанные или перегруженные интерфейсы повышают информационную нагрузку на пользователя и

уменьшают скорость и точность его попыток разобраться в содержимом экрана.

В интерфейсах следует применять простые геометрические формы, минималистичные контуры и ограниченные наборы не очень ярких или нейтральных цветов, уравновешенные небольшим числом высококонтрастных и ярких цветов, позволяющих подчеркивать важную информацию.

Типографика не должна быть слишком разнообразной: одного или двух шрифтов в нескольких размерах вполне достаточно. Когда несколько элементов дизайна служат родственным или взаимосвязанным логическим целям, эти элементы должны визуальнo оформляться таким образом, чтобы работал принцип наследования. Наследование дает возможность переносить понимание одного элемента на другие сходные элементы. Если элемент требуется выделить, создайте контраст с прочими элементами через настройку одного или нескольких визуальных свойств, таких как размер, цвет и положение. Бессмысленные вариации визуальных свойств мешают создавать целостные и удобные интерфейсы.

За любым визуальным элементом, любым отличием цвета, размера или другого визуального свойства должны стоять определенные причины. Если вы не можете сформулировать причину для отличий, избавьтесь от них.

Хороший графический интерфейс, как любой хороший дизайн, визуальнo эффективен. Он максимально полно использует минимальный набор визуальных и функциональных элементов. Распространенный прием – эксперимент с удалением отдельных элементов с целью проверки их вклада в ясность задуманного послания пользователю. Удаляйте элементы, пока продукт не сломается, а затем верните последний удаленный элемент на место.

Текст в графических интерфейсах

Текст – неотъемлемая составляющая практически всех пользовательских интерфейсов. Следует крайне внимательно относиться к применению текста, поскольку он обладает способностью запутывать и усложнять простые вещи.

Человек распознает буквы исходя из их форм. Чем более узнаваема форма, тем проще распознать букву, поэтому СЛОВА, СОСТОЯЩИЕ СПЛОШЬ ИЗ ЗАГЛАВНЫХ БУКВ, ЧИТАТЬ ТРУДНЕЕ, чем написанные

обычным образом: в заглавных буквах отсутствуют привычные глазу начертания, и поэтому чтение требует большего внимания. Избегайте в своих интерфейсах слов, состоящих из заглавных букв.

Распознавание отдельных слов – не то же самое, что чтение, при котором мы осознанно сканируем отдельные слова и интерпретируем их значения в контексте. Следует стараться минимизировать объем текста, который пользователю необходимо прочитать, чтобы сориентироваться в интерфейсе. Применение коротких простых слов облегчает навигацию при минимальной необходимости в чтении.

Что касается читаемого текста в интерфейсе, рекомендуется придерживаться некоторых принципов:

1. Используйте контрастный текст. Убедитесь, что текст хорошо контрастирует с фоном и что не используются дополнительные цвета, способные повлиять на удобочитаемость текста.

2. Используйте подходящий шрифт и кегль (размер). Как правило, шрифт без засечек и с резкими контурами, такой как Verdana или Tahoma, читается лучше всего. Текст мельче 10 пикселей в большинстве ситуаций трудно читать.

3. Четко формулируйте мысли. Пользуйтесь минимальным количеством слов, необходимым для ясной передачи смысла. Избегайте сокращений. Если они все-таки необходимы, используйте общепринятые.

Цвет в графических интерфейсах

Цвет является составной частью визуального языка интерфейса, и пользователи будут пытаться усмотреть смысл в том, как он применяется. В большинстве приложений цвет должен использоваться сдержанно и хорошо сочетаться с прочими элементами визуального языка – символами, пиктограммами, текстом – и пространственными отношениями между ними. Если не проявлять осторожность, цвет очень легко применить неправильно. Самые распространенные ошибки:

1. Слишком много цветов. Добавление одного цвета, выделяющего важные элементы в наборе, значительно сокращает время поиска. Добавление новых цветов приводит к дополнительному ускорению работы пользователя,

но при семи и более цветах скорость поиска значительно падает. Разумно предположить, что сходные результаты будут получены при любом типе навигации по интерфейсу, поскольку число семь отражает количество элементов информации, единовременно сохраняемой в кратковременной памяти человека.

2. Использование насыщенных дополнительных цветов. Дополнительными являются цвета, противоположные друг другу в числовом представлении. Если такие цвета обладают достаточно высокой насыщенностью и расположены рядом, то порождают зрительные эффекты, препятствующие легкому восприятию и мешающие сосредоточить внимание.

3. Чрезмерная насыщенность. Сильно насыщенные цвета выглядят кричаще и привлекают слишком много внимания. Умеренное насыщение цвета допустимо для небольших областей, привлекающих внимание пользователей, но такие области всегда следует создавать с осторожностью.

4. Недостаточный контраст. Когда цвет объекта отличается от цветов фона лишь оттенком, но не насыщенностью или яркостью, объект становится трудно воспринимать. Фигура и фон должны различаться по яркости и насыщенности, а не только по оттенку. Кроме того, необходимо избегать использования цветного текста на цветном фоне везде, где только возможно.

5. Недостаточная забота о людях с нарушениями цветового восприятия. Примерно десять процентов мужского населения страдает цветовой слепотой той или иной степени. Это означает, что при использовании (в частности) оттенков красного и зеленого для передачи важной информации следует проявлять внимательность. Любые цвета, применяемые в интерфейсе, должны заметно различаться по насыщенности или яркости. Если интерфейс остается читаемым после преобразования в черно-белый вариант, люди с нарушениями цветового восприятия смогут работать и с цветным вариантом интерфейса.

Принципы визуального информационного дизайна

Разработка визуального представления информации имеет свои принципы, и дизайнер может применять их для достижения наилучшего результата. В сфере визуального представления информации существуют две

важных проблемы:

1. Трудно визуализировать многомерную информацию (информацию с более чем двумя переменными) на двухмерной поверхности.

2. Разрешение конечного носителя не всегда достаточно велико для вывода плотной информации.

Существуют универсальные принципы информационного дизайна, которые помогают повысить эффективность любого представления информации.

Информация, представленная визуально, должна:

1. Способствовать визуальному сравнению. Предоставьте пользователям возможность сравнивать взаимосвязанные переменные и тенденции либо сопоставлять варианты «до» и «после». Сравнение создает контекст, делающий информацию более ценной и понятной пользователям (пример см. на рис. 1.7).

2. Показывать причинно-следственную связь. Представляя информацию в графическом виде, четко обозначайте причину и следствие. В интерактивных интерфейсах необходимо обеспечить визуальную обратную связь, чтобы информировать пользователей о последствиях их действий или предоставить им подсказку о том, как следует действовать.

3. Отображать сразу несколько величин. Окна, содержащие информацию о нескольких взаимосвязанных переменных, должны в случае необходимости отображать все эти переменные одновременно без ущерба для ясности. При этом у пользователя должна быть возможность избирательно включать и отключать вывод величин этих переменных в интерактивном режиме, чтобы облегчить их сравнение и поиски корреляции.

4. Объединять текст, графику и данные в одном изображении. Диаграммы, требующие дополнительных подписей, легенд или расшифровок, нагружают пользователя добавочной когнитивной обработкой. Пользователю приходится переключать внимание с диаграммы на подпись и обратно, а затем соотносить их в уме. Гораздо лучше поместить необходимые надписи прямо на диаграмме (рис. 1.8).

5. Гарантировать качество, релевантность и целостность данных. Не

следует выводить информацию на экран только потому, что это технически возможно. Убедитесь, что любая отображаемая информация помогает пользователю достигать конкретных целей и уместна в данном контексте.

6. Группировать объекты в пространстве, а не во времени. Когда вы отображаете данные, меняющиеся во времени, пользователю легче воспринимать их динамику, если значения выводятся рядом, а не перекрываются.

7. Представлять числовые данные в числовом виде. Даже если вы предпочитаете пользоваться графиками и диаграммами, чтобы облегчать восприятие тенденций и иной количественной информации, не следует полностью отказываться от вывода собственно числовых данных (рис. 1.8).

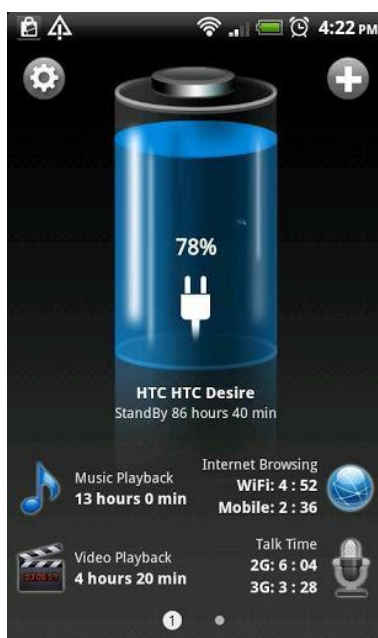


Рисунок 1.8 – Гистограмма Android-приложения Battery HD дает примерное представление об уровне заряда батареи, но здесь же выводится точное значение

Поведение окон и определение компоновки

Основными составляющими интерфейса приложений являются окна двух видов – главные и подчиненные. Выбор способа применения окон в программе – важный аспект общей инфраструктуры пользовательского интерфейса.

Исторически окна появились из аналогии с бумагами, которыми завален письменный стол. Когда необходимо использовать какой-то документ, человек

кладет его поверх других. Эта концепция вполне логична, когда речь идет о большом экране. Однако для смартфонов следует учитывать, что пользователь в момент времени может увидеть максимум один документ, причем довольно в редких случаях целиком.

Лишние комнаты

Если мы представим приложение в виде дома, то каждое окно – это отдельная комната.

Сам дом будет представлен главным окном программы, а каждая комната – это окно документа, диалоговое окно или панель. Мы не пристраиваем к нашему дому комнату, если у нее нет уникальной функции, которую не способны выполнять другие комнаты. Точно так же мы не должны добавлять окно в программу, если у него нет предназначения, которое не могут или не должны выполнять существующие окна.

Важно подходить к вопросам предназначения с позиции будущих целей и пользовательских ментальных моделей. Думая о том, каково предназначение комнаты, мы подразумеваем определенную цель – не обязательно конкретную задачу или функцию.

Диалоговое окно – это еще одна комната; не ходите в комнату без веской причины.

Это один из наиболее часто нарушаемых принципов в проектировании пользовательских интерфейсов. Программисты выполняют свою работу, разбивая приложение на дискретные функции, при этом пользовательский интерфейс часто разрабатывается параллельно. В получается закономерный результат: по диалоговому окну на каждую функцию. Это сильно усложняет интерфейс.

Старайтесь, чтобы разрабатываемое приложение как можно лучше отвечало правилу трех кликов. Иными словами, до любого элемента программы можно добраться не более чем за три действия.

Важные окна

Когда пользователи совершают действия, выходящие за пределы обычного хода событий, программа должна отвести специальное место для их

выполнения. Например, если пользователь приехал в другой город и хочет узнать погоду с помощью установленного у него приложения, он должен указать свое новое местоположение. Для программы совершенно естественно перевести пользователя для выполнения этой функции в отдельную «комнату» – либо окно, либо диалоговое окно.

Руководствуясь принципами целеориентированного мышления, можно определить полезность каждой функции. Если человек рисует в графическом редакторе, его цель – создать привлекательное и эффектное изображение. Все инструменты рисования непосредственно связаны с этой целью, но самые нужные инструменты – это карандаши, кисти и ластик.

Эти инструменты следует глубоко интегрировать в рабочее пространство, ведь так поступают художники в реальном мире, располагая свои инструменты у холста, под рукой. Инструменты готовы к моментальному применению, за ними не нужно даже тянуться, не говоря уже о том, чтобы ходить за ними в другую комнату.

Инструменты программы рисования следует располагать у границы холста и делать доступными в одно действие. Пользователя не следует отправлять за ними в меню или диалоговое окно (рис. 1.9).

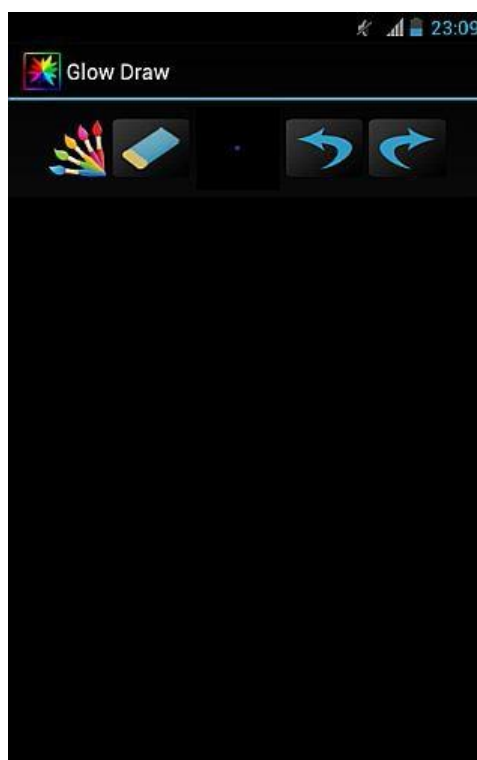


Рисунок 1.9 – Главный экран приложения Glow Draw не содержит ничего

Изучая цели пользователей, мы естественным образом приходим к уместной форме приложения. Вместо того чтобы отвести по диалоговому окну на каждую функцию, мы делаем вывод, что некоторым функциям вообще не место в диалоговых окнах, а есть и такие функции, которые нужно попросту удалить из программы.

Замусоривание окнами

Некоторые проектировщики считают, что каждое диалоговое окно должно вмещать единственную функцию. Результатом такого подхода становится замусоривание интерфейса окнами.

Решение многих пользовательских задач требует выполнения последовательности действий. Если для каждого действия открывается отдельное диалоговое окно, то экран вскоре визуальнo перегружается, а навигация усложняется. Каждое новое окно взваливает на плечи пользователя дополнительный интерфейсный налог, связанный с управлением окнами. При ежедневном использовании программы совокупный размер этих налогов может увеличиться до невероятных размеров.

Проектирование для различных потребностей

Персонажи и сценарии помогают нам сосредотачивать усилия проектирования на целях, поведении, потребностях и ментальных моделях реальных пользователей. Существуют также последовательные и доступные для обобщения шаблоны пользовательских потребностей, которые следует учитывать при проектировании продуктов. Рассмотрим некоторые стратегии удовлетворения этих распространенных потребностей.

Командные векторы, рабочие наборы и персонажи

При классификации потребностей пользователей с различным уровнем подготовки весьма полезны понятия командных векторов и рабочих наборов. Командные векторы – это самостоятельные методы, которыми пользователи могут давать команды программе. Визуальные элементы для непосредственного манипулирования, раскрывающиеся и контекстные меню, элементы управления

на панели инструментов – все это примеры командных векторов.

Хороший пользовательский интерфейс предоставляет пользователю множественные командные векторы, в которых важные функции приложения представлены в разных формах, чтобы обеспечить пользователя параллельными возможностями управлять приложением. Например, многие приложения работы с картами позволяют изменять масштаб как с помощью специальных кнопок, так и с помощью специальных жестов (рис. 1.10).

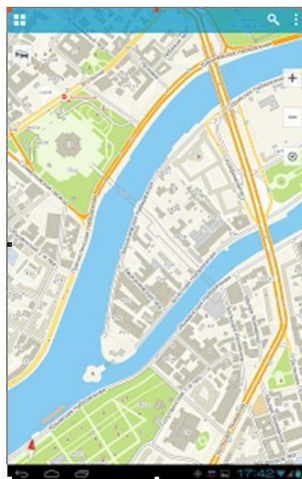


Рисунок 1.10 – Масштаб изображения можно изменять разными способами

На экране есть специальные кнопки; в то же время опытные пользователи сенсорных экранов ими практически не пользуются, предпочитая одновременное сведение/разведение двух пальцев (один из жестов технологии мультитач)

Среди командных векторов есть более удобные для новичков. Как правило, максимальную поддержку оказывают меню и диалоговые окна, и поэтому мы назовем их обучающими векторами. Начинающие пользователи выигрывают от педагогических свойств меню, когда им необходимо сориентироваться в незнакомой программе, а вечные середняки стремятся отказаться от меню и ищут более непосредственные эффективные векторы. Поскольку в памяти каждого пользователя невольно откладываются часто используемые команды, многие запоминают некоторое подмножество команд, называемое рабочим набором. У каждого пользователя свой рабочий набор, хотя весьма вероятно, что он значительно пересекается с рабочими наборами других пользователей. Строго говоря, не существует стандартного рабочего

набора, покрывающего потребности всех пользователей, исследование и моделирование пользователей и способов использования продукта позволяют получить минимальное подмножество команд, необходимых большинству. Такой минимальный рабочий набор можно определить методами целеориентированного проектирования – посредством сценариев, раскрывающих функциональные потребности персонажей.

Команды, составляющие рабочий набор любого пользователя, – это наиболее часто используемые команды. Пользователь хочет, чтобы доступ к ним был особенно быстрым и простым. Проектировщик обязан предоставить незамедлительные командные векторы для минимального рабочего набора наиболее вероятных пользователей данного приложения. Кроме того, это облегчает освоение новичками программного продукта.

Существует исключение из правила множественных векторов. Опасные команды (такие как Стереть все) не должны иметь легко доступных множественных командных векторов. Их следует защищать посредством меню и диалоговых окон.

Персонализация и настройка

Проектировщикам взаимодействия часто приходится принимать решение относительно того, давать ли пользователю настраивать продукт согласно своим предпочтениям.

Люди любят изменять окружение по своему вкусу. Даже новички любят ставить на программу свое клеймо, настраивая ее так, чтобы она выглядела и вела себя в соответствии с их предпочтениями и личным вкусом. Люди делают это по той причине, что это придает им индивидуальность.

Персонализация – украшение стабильных объектов. Персонализация делает программные продукты более симпатичными и узнаваемыми, более человечными и приятными для работы, становится полезным средством навигации.

С другой стороны, от перемещения стабильных объектов навигация может пострадать. Украшение стабильных объектов способствует навигации, а

перемещение их – препятствует.

Настройкой называется перемещение, добавление и удаление стабильных объектов.

Настройка желательна для более опытных пользователей, которые, определившись со своим рабочим набором функций, захотят настроить интерфейс таким образом, чтобы эти функции было легче находить и выполнять.

Инструменты персонализации должны быть простыми и удобными в обращении, давая пользователю возможность предварительного просмотра результатов его действий. Кроме того, действия персонализации должны легко отменяться.

Большинство конечных пользователей не будут громко протестовать, если им не предоставят возможность настраивать программу. Некоторые действительно опытные пользователи, возможно, почувствуют себя обделенными, но они, тем не менее, будут работать с программой и ценить ее по достоинству, если она работает так, как они того ожидают.

Однако в некоторых случаях гибкость абсолютно необходима. Если вы проектируете продукт для быстро эволюционирующих рабочих процессов, становится чрезвычайно важным, чтобы этот продукт мог эволюционировать с такой же скоростью, как среда его применения.

Пользовательское тестирование неоднократно демонстрировало, как аудитория расходится во взглядах на эффективность той или иной идиомы, разделяясь на две примерно равные части. Половина пользователей явно предпочитает одну идиому, а вторая половина – другую. Такое четкое разделение аудитории по предпочтениям на две или несколько крупных групп указывает, что эти предпочтения идиосинкратически модальны.

В таких случаях не нужно устраивать дебаты, чтобы понять, какой из способов следует применять, потому что они легко превращаются в холивар (священную войну). Здесь ответ очевиден: применяйте оба! Когда пользовательская аудитория разделена предпочтениями относительно идиом, проектировщики обязаны предложить обе идиомы. Необходимо удовлетворить обе группы. Нельзя удовлетворить одну половину аудитории, оставив в

негодовании другую, – и при этом совершенно неважно, к какой группе относите себя вы или ваши разработчики.

Локализация и глобализация

Создание приложений для различных языков и культур ставит перед проектировщиками ряд специфических проблем. И здесь снова хорошим ориентиром могут служить командные векторы. Незамедлительные векторы, такие как непосредственное манипулирование и кнопки-значки на панели инструментов, являются идиоматическими и визуальными, поэтому они без особых сложностей поддаются глобализации. Конечно, проектировщикам следует проделать подготовительную работу, чтобы убедиться, что выбранные для этих идиом цвета и символы не имеют в других культурах какого-то особого смысла, не заложенного разработчиком. Однако в целом неметафорические идиомы в глобализованных интерфейсах вполне безопасны.

Обучающие векторы, такие как пункты меню, подписи к полям ввода, всплывающие подсказки и инструкции, зависят от языка и, таким образом, являются объектом локализации, осуществляемой посредством перевода. При создании интерфейсов, подлежащих локализации, следует принять во внимание следующие моменты:

1. В некоторых языках слова и фразы длиннее, чем в других.
2. В азиатских языках может вызвать сложности алфавитная сортировка.
3. Формат вывода даты и соглашения о применении 12и 24-часовых систем счисления времени в разных странах различны.
4. Символы, отделяющие десятичную часть от целой в числах и триады в денежных суммах, могут быть разными.
5. В ряде стран принято нумеровать недели в году. Кроме того, не во всех странах придерживаются григорианского календаря.

К переводу пунктов меню и сообщений в диалоговых окнах следует подходить целостно. Важно убедиться, что в результате перевода интерфейсы не утратили связности. Переведенные в отрыве от контекста пункты меню и метки могут вызывать путаницу, когда окажутся рядом с другими независимо переведенными элементами интерфейса. Семантику интерфейса следует

сохранять на всех уровнях.

Коллекции и шаблоны

Не каждый пользователь приложения, имеющего дело с документами, способен создавать документы с нуля. Однако большинство программ предлагают пользователям лишь простейшие инструменты. Некоторым пользователям этого достаточно, но другим нужно больше. В любом приложении необходимо позволить пользователям выбирать дизайн или исходную структуру документа из коллекции шаблонов, когда у них нет необходимости или желания создавать документ с нуля. Пустые листы пугают большинство людей, а пользователи не должны вынужденно терпеть то, что им не нравится. Коллекции базовых шаблонов являются удачным решением.

Справка

Хорошая оперативная справка крайне важна, но она не должна становиться костылем для продукта. Качественно спроектированный интерфейс должен значительно сократить потребность пользователей в справке.

Сложную программу с большим количеством функций и возможностей необходимо сопровождать справочным документом, с помощью которого пользователи, желающие расширить свое понимание продукта, смогут найти конкретные ответы на свои вопросы. Однако для простой программы (например, казуальной игры) часто лучше вообще отказаться от справочной системы, заменив ее небольшим руководством по «быстрому старту» (рис. 1.11).



Рисунок 1.11 – В игре Loops! правила игры занимают всего пять экранов, использован очень простой язык, каждое действие снабжено понятной иллюстрацией

Всплывающие подсказки относятся к немодальной оперативной справке и являются невероятно эффективным средством помощи пользователю.

Задания к лабораторной работе

Задание 1. Создание макета интерфейса и эскиза навигации на бумаге

Используя сформулированные в результате выполнения предыдущей лабораторной работы сценарии поведения персонажей, выполните следующие действия:

1. Нарисуйте макет вашего интерфейса.
2. Изложите кому-нибудь свой замысел и попросите его представить, что он пользуется вашим приложением. Пусть «пользователь» при этом рассказывает о том, что ожидает увидеть, щелкая по разным кнопкам и ссылкам.
3. Процесс «использования» можно заснять на видео с тем чтобы проанализировать его
4. впоследствии.
5. Усовершенствуйте дизайн навигации с учетом замечаний.
6. Продумайте детали: комбинации кнопок, подсказки, рабочие

инструменты. Попробуйте усовершенствовать взаимодействие с пользователем на каждом шаге проектирования.

Задание 2. Разработка прототипа интерфейса

На этом этапе необходимо перейти от проектирования на бумаге к использованию компьютера с установленным специализированным программным обеспечением.

1. Детализируйте макет интерфейса, подготовленный ранее.
2. Выберите базовые цвета и фигуры для приложения.
3. Выберите шрифты для своего приложения.

Подумайте, как нужно изменить ваше приложение, чтобы оно подошло пользователям других стран.

Может быть, имеет смысл развить идею и сделать из нее также программу для компьютера или планшета?

Содержание отчета и его форма

Отчет к лабораторной работе оформляется в виде текстового документа по форме простого реферата и должен включать:

1. Название лабораторной работы.
2. Цель лабораторной работы.
3. Краткие выводы по результатам выполнения лабораторной работы согласно пунктам раздела «Методики выполнения работы».
4. Ответы на вопросы к лабораторной работе.
5. Формулировку задания для самостоятельной работы и результат его выполнения.

Задания для самостоятельной работы

1. Создать макет интерфейса и эскиз навигации на бумаге
2. Протестировать подготовленный макет на предполагаемых пользователях
3. Внести необходимые корректировки в макет с учетом результатов тестирования
4. Разработать прототип интерфейса с использованием реальных элементов управления

Вопросы к лабораторной работе

1. Принципы проектирования взаимодействия.
2. Задачи проектировщиков взаимодействия.
3. Назначение шаблонов проектирования взаимодействия.
4. Типы шаблонов проектирования взаимодействия
5. Понятия «графический дизайн» и «пользовательские интерфейсы».
6. Строительные блоки визуального дизайна интерфейсов
7. Использование визуальных свойств для группировки элементов и создания четкой иерархии
8. Интеграция визуального стиля с функциональностью
9. Принципы визуального информационного дизайна

Лабораторная работа 2

Основы тестирования и отладки приложений на смартфоне

Цель работы: рассмотреть инструменты для тестирования мобильных приложений, входящие в состав Android SDK.

Теоретическое обоснование

К вопросам тестирования мобильных приложений, разработанных под Android, имеет непосредственное отношение работа с эмулятором, который позволяет тестировать приложения без привлечения реальных устройств. Android SDK позволяет создавать эмулятор и запускать его на компьютере разработчика, существует набор ключевых клавиш для управления эмулятором, таблица ключевых клавиш:

<http://developer.android.com/tools/help/emulator.html>

Часто необходимо тестировать поведение приложения при поступлении входящего вызова, при получении SMS, а также работу приложений использующих дополнительные возможности смартфона: камеру, приемник GPS и другие возможности. Описание работы с эмулятором:

<http://developer.android.com/tools/devices/emulator.html>

Среда Android тестирования основана на использовании библиотеки JUnit, поэтому необходимо иметь представление о том, как устроена эта библиотека и как с ней работать, для первичного ознакомления можно порекомендовать видео-урок:

<https://www.youtube.com/watch?v=QJZb1fNYh9c&list=PLB0276A0A62BDEF06&index=17> из целого набора видео-уроков «Java для тестировщиков»:

<https://www.youtube.com/playlist?list=PLB0276A0A62BDEF06>. И еще информация по JUnit: <http://habrahabr.ru/post/120101/>.

Инструменты тестирования в Android SDK

Android SDK включает мощные инструменты для тестирования приложений. Инструменты расширяют JUnit дополнительными возможностями; предоставляют готовые к использованию классы для объектов, имитирующих Android систему; дают контроль над главным приложением во время его

тестирования.

В работе используем простое Android приложение из комплекта Android SDK, для которого создадим тестирующее приложение, тем самым продемонстрируем инструменты Android тестирования, включенные в Android IDE.

Ключевые моменты тестирования в Android IDE:

1. Android тест сам является приложением, которое связано с тестируемым приложением, что отражено в его файле AndroidManifest.xml;
2. вместо Android-компонентов приложение тестов содержит один или более тестовых сценариев, каждый из которых является отдельным классом;
3. классы Android тестов расширяют класс TestCase библиотеки JUnit;
4. классы Android тестов для активностей расширяют JUnit и связываются с тестируемым

приложением, можно передавать события: нажатие кнопки или касания экрана напрямую в пользовательский интерфейс;

5. класс теста выбирается в соответствии с типом тестируемого компонента (приложение, активность, контент-провайдер или сервис);

6. в Eclipse/ADT существуют дополнительные инструменты, обеспечивающие интегрированную поддержку создания тест-приложения, запуска его и просмотра результатов.

Тестирующее приложение содержит методы, которые выполняют следующие тесты:

7. **Тест начальных условий** проверяет, что тестируемое приложение корректно инициализировано, тестируется метод onCreate(). Данный тест обеспечивает степень достоверности для последующих тестов.

Тест пользовательского интерфейса (UI) проверяет, что главный UI процесс работает корректно. Данный тест демонстрирует аппаратные особенности доступные в тестировании активностей, показывает, что можно автоматизировать тестирование пользовательских интерфейсов, передавая события кнопок из тест-приложения в главное приложение.

8. **Тест управления состояниями** демонстрирует возможности

инструмента для запуска тестов, которые доступны при тестировании любого компонента.

Задания к лабораторной работе

Задание 1. Создание тестирующего приложения

а. В качестве приложения для тестирования будем использовать уже готовое приложение, входящее в набор созданных разработчиками Android SDK примеров, приложение называется Spinner и содержится в наборе примеров (Samples). Мы взяли приложение из набора примеров для API 8 (рис. 2.1).

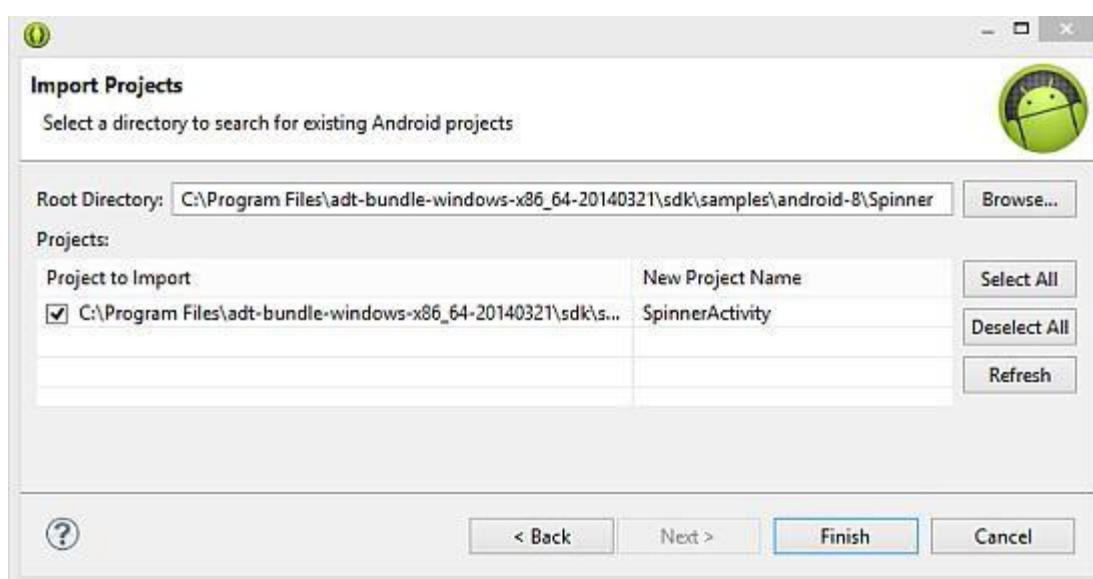


Рисунок 2.1 – Импорт существующего Android-приложения

Запустите Eclipse, создайте новое Android приложение, используя уже существующее:

New->Project...->Android->Android Project from Existing Code

в открывшемся диалоге выберите расположение проекта, это может выглядеть примерно, как на рис. 1. Убедитесь, что приложение запускается и что-то выполняет. Разумеется, не обязательно тестировать именно это приложение, можно и лучше использовать для этого Ваше Android приложение.

б. Создадим тестирующее приложение.

В Eclipse New->Project...->Android->Android Test Project

Задайте имя проекта, обычно имя тестирующего проекта собирается

следующим образом: имя тестируемого проекта + слово Test, в нашем случае получится SpinnerActivityTest. Далее необходимо указать тестируемый проект (рис. 2.2).

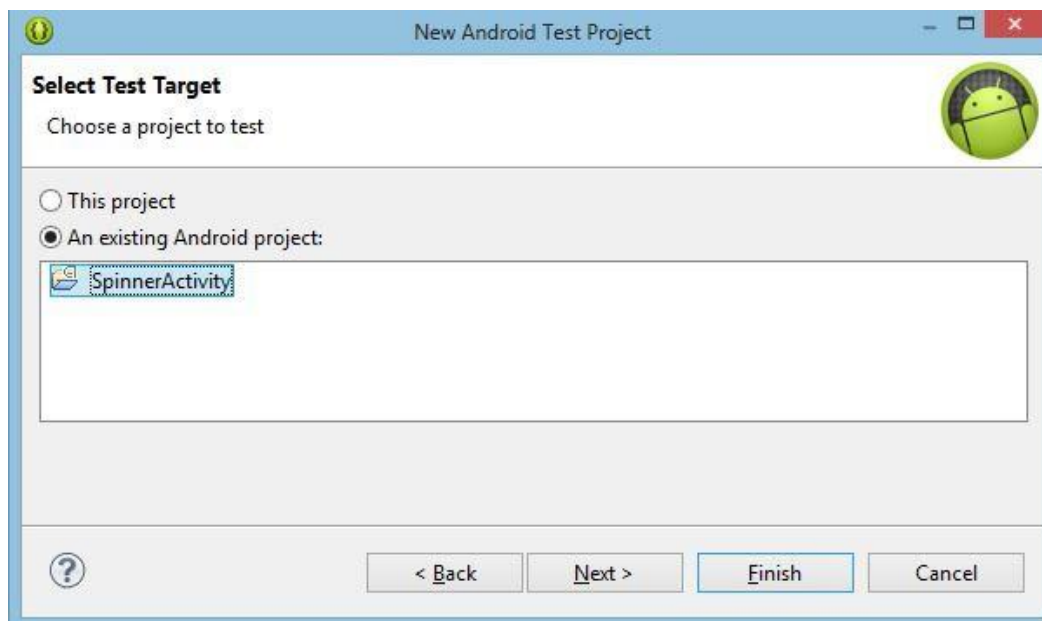


Рисунок 2.2 – Выбор тестируемого проекта

Теперь у нас есть простейшее Android приложение (SpinnerActivity) и пустое тестирующее приложение (SpinnerActivityTest).

Файл AndroidManifest.xml тестирующего приложения выглядит, как изображено на рисунке 2.3. В разделе <instrumentation> указано где располагается тестируемый проект (рис. 2.4).



Рисунок 2.3 – Файл манифеста приложения SpinnerActivityTest

с. Создадим класс тестов (рис. 2.4).

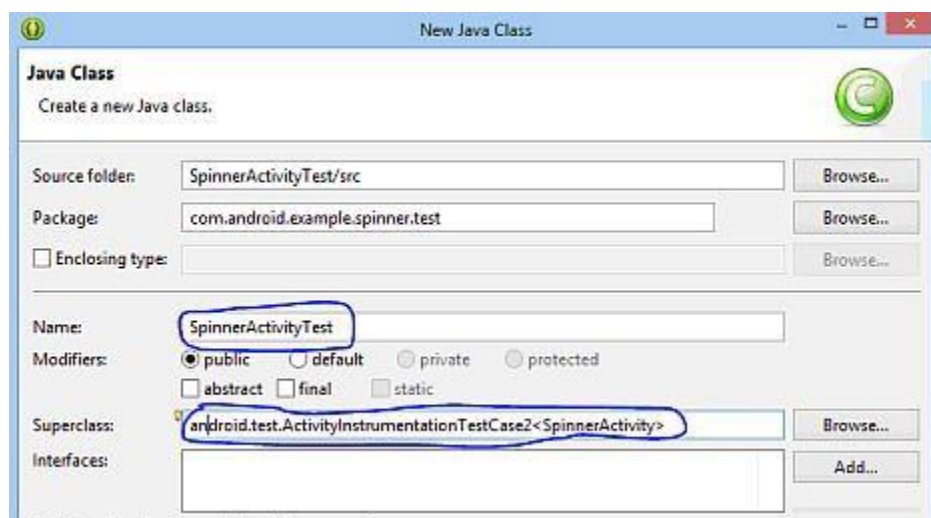


Рисунок 2.4 – Создание класса тестовых сценариев

В пакете, расположенном в папке src/ проекта SpinnerActivityTest, создадим новый класс. Имя класса: SpinnerActivityTest, суперкласс:

`android.test.ActivityInstrumentationTestCase2<SpinnerActivity>`

Класс ActivityInstrumentationTestCase2 спроектирован для выполнения функционального тестирования одной или нескольких активностей приложения.

Чтобы была возможность обращаться к SpinnerActivity необходимо в файл SpinnerActivityTest.java добавить следующую строчку:

```
import com.android.example.spinner.SpinnerActivity;
```

Добавим конструктор класса тестов:

```
public SpinnerActivityTest() {  
    super(«com.android.example.spinner»,  
        SpinnerActivity.class);  
}
```

Добавим метод начальных установок

Метод setUp() вызывается перед каждым тестом, используется для инициализации переменных и очистки значений после предыдущих тестов. Также можно использовать метод tearDown(), который вызывается после каждого теста, в этой лабораторной работе мы его рассматривать не будем.

Код метода:

```

@Override
protected void setUp() throws Exception {
    super.setUp();
    setActivityInitialTouchMode(false);    mActivity =
getActivity();
    mSpinner = (Spinner)mActivity.findViewById
(com.android.example.spinner.R.id.Spinner01);
    mPlanetData = mSpinner.getAdapter();
}

```

Рассмотрим метод:

1) `super.setUp()`— вызывает конструктор суперкласса для `setUp()`, как этого требует JUnit;

2) `setActivityInitialTouchMode(false)` – выключает режим касаний на эмуляторе и устройстве, если какой-то из ваших тестов передает события нажатия кнопок в приложение, необходимо отключать режим касаний перед запуском любой активности, иначе вызовы будут игнорироваться;

`getActivity()` – получает ссылку на тестируемую активность, этот вызов также запускает активность, если это до сих пор не сделано;

3) `findViewById(int)` – получает ссылку на виджет `Spinner` в тестируемом приложении;

4) `getAdapter()` – получает ссылку на адаптер (массив строк) соответствующий `Spinner`.

В класс тестов необходимо добавить следующие элементы: `private SpinnerActivity mActivity;`

```
private Spinner mSpinner;
```

```
private SpinnerAdapter mPlanetData;
```

И импортировать следующие пакеты: `import android.widget.Spinner;`

```
import android.widget.SpinnerAdapter;
```

Добавим тест начальных условий

Этот тест проверяет, что тестируемое приложение инициализировано корректно. Метод будет проверять следующее:

5) инициализирован слушатель события выбора элемента из списка, этот слушатель вызывается когда в спинере выбирается какой-то элемент;

6) инициализирован адаптер, предоставляющий значения для спинера;

7) адаптер содержит правильное количество записей. Код метода:

```
public void testPreConditions() {
    assertTrue(mSpinner.getItemSelectedListener() != null);
    assertTrue(mPlanetData != null);
    assertEquals(mPlanetData.getCount(), ADAPTER_COUNT);
}
```

В класс тестов необходимо добавить элемент: `public static final int ADAPTER_COUNT = 9;` **Добавим тест интерфейса пользователя**

Создадим UI тест, который выбирает элементы из виджета спинер. Тест посылает события нажатия клавиш и проверяет, что выбор соответствует ожидаемым результатам.

Для работы со спинером тест должен запросить фокус и затем установить его в известную позицию, для этого используются методы `requestFocus()` и `setSelection()`. Оба эти метода взаимодействуют с представлениями тестируемого приложения, поэтому должны вызываться специальным образом.

Код для получения фокуса и установки выделения выглядит следующим образом:

```
mActivity.runOnUiThread(
    new Runnable() {
        public void run() { mSpinner.requestFocus();
mSpinner.setSelection(INITIAL_POSITION);
        }
    }
);
```

Необходимо добавить его в метод:

```
public void testSpinnerUI() {
    ...
```

```
}
```

Необходимо добавить в класс тестов элемент: `public static final int INITIAL_POSITION = 0;`

Далее необходимо сделать выбор элемента спинера для этого передадим событие нажатия кнопки в спинер:

```
this.sendKeys(KeyEvent.KEYCODE_DPAD_CENTER); for (int
i      =      1;      i      <=      TEST_POSITION;      i++)      {
this.sendKeys(KeyEvent.KEYCODE_DPAD_DOWN);
}
this.sendKeys(KeyEvent.KEYCODE_DPAD_CENTER);
```

Необходимо добавить в класс тестов элемент: `public static final int TEST_POSITION = 5;` Необходимо импортировать:

```
import android.view.KeyEvent;
```

Проверка результатов. Запросить текущее состояние спинера и сравнить его с ожидаемым значением. Вызов метода `getSelectedItemPosition()` позволит получить текущую выбранную позицию, а метод `getItemAtPosition()` возвращает элемент, соответствующий этой позиции (представленный в виде строки). Метод `assertEquals()`; проверяет совпадает ли полученное значение с ожидаемым «Saturn».

Добавьте в метод `public void testSpinnerUI()` следующий код:

```
mPos = mSpinner.getSelectedItemPosition();
mSelection
=
(String)mSpinner.getItemAtPosition(mPos);
TextView
resultView      =      (TextView)mActivity.findViewById
(com.android.example.spinner.R.id.SpinnerResult);
String
resultText      =      (String)      resultView.getText();
assertEquals(resultText,mSelection);
```

Необходимо добавить в класс тестов следующие элементы: `private String mSelection;`

```
private int mPos;
```


И импортировать:

```
import android.widget.TextView;
```

Задание 2. Запуск теста и просмотр результатов.

Перед запуском теста необходимо удостовериться, что запущен эмулятор или подключено устройство, на котором будет производиться тестирование.

Для запуска теста в Package Explorer щелкните правой кнопкой на названии проекта тестов (SpinnerActivityTest), а затем выберите Run As→Android JUnit Test

После запуска теста появится новая вкладка JUnit рядом с вкладкой Package Explorer (рис. 2.5).

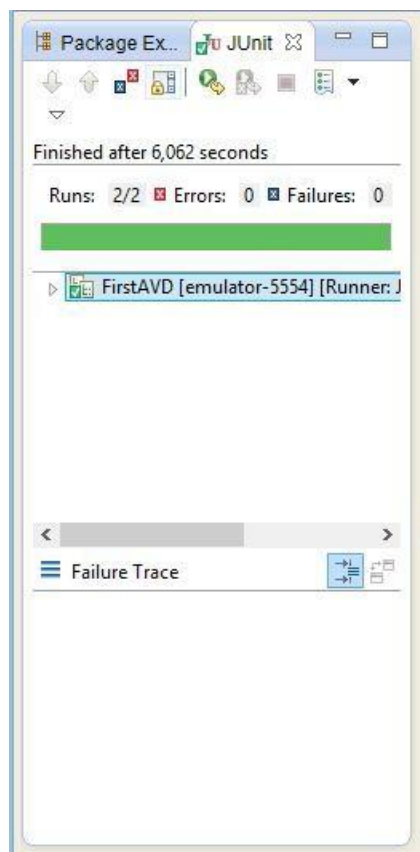


Рисунок 2.5 – Вкладка JUnit

Эта вкладка содержит две панели. Верхняя панель содержит информацию о тестах, которые были запущены, нижняя панель содержит запись ошибок.

Информация о запущенных тестах содержит следующие пункты:

- 8) общее время, затраченное на исполнение теста;
- 9) количество запусков (Runs) – количество тестов в классе;

10) количество ошибок (Errors) – количество ошибок и исключений, обнаруженных в приложении за время работы теста;

11) количество провалов (failures) – количество неудачных испытаний во время работы теста;

12) полоса состояния, если тесты прошли успешно, полоса остается зеленой, если тест провален, полоса становится красной;

13) под полосой состояния можно увидеть описание каждого класса в тестирующем приложении, можно просмотреть результаты каждого метода тест-класса.

Спровоцируем провал некоторых тестов

Посмотрим, что произойдет в Android IDE, когда тест будет провален. Можно быстро увидеть какой тест-класс был провален, найти метод или методы, которые были провалены, используя запись ошибки точно найти проблему.

Испортим в файле SpinnerActivity.java метод onCreate(), в этом файле есть закомментированная строка:

```
// spinner.setOnItemSelectedListener(null);
```

Снимите комментарий, тем самым проверим работу метода testPreCondition() класса

SpinnerActivityTest, в этом методе в строке:
assertTrue(mSpinner.getOnItemSelectedListener() != null);

утверждается, что слушатель событий визуального компонента Spinner не равен null. После удаления символов комментария и запуска тестирующего проекта, это утверждение будет выдавать ошибку, т.е. тест будет провален. Результаты таких изменений на рис. 2.6. Теперь полоса состояния стала красной, количество провалов равно 2. Очевидно желание разработчика посмотреть на эти провалы и увидеть точно, где они появились.

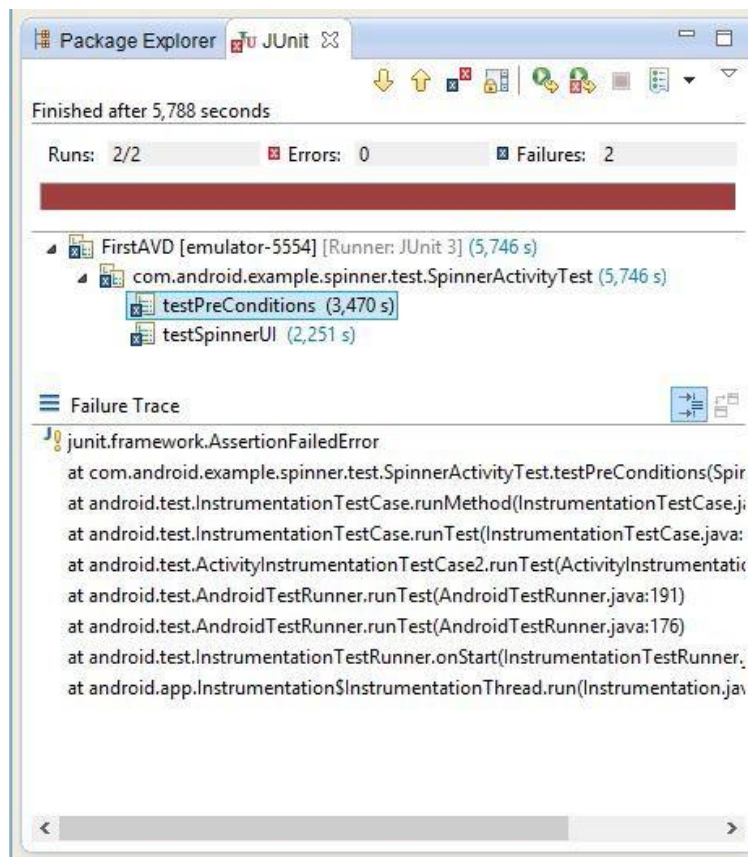


Рисунок 2.6 – Вкладка JUnit, пример проваленных тестов

Чтобы найти место в коде, на котором был провален тест, достаточно щелкнуть мышью по строке `testPreCondition`, выделенной на рис.4-5.6 в нижней панели `Failure Trace` появится список вызовов, который привел к провалу теста. Первая строка в этой панели сообщает об ошибке, чтобы увидеть место в коде, где произошел провал теста достаточно дважды щелкнуть по строке с ошибкой.

Выполненная работа продемонстрировала путь создания тестирующего проекта и способ связывания его с приложением, которое необходимо тестировать. Был выбран и добавлен класс тестовых сценариев, созданы тесты и запущен процесс тестирования: в первом случае без провалов, во втором случае созданы условия для провала тестов, рассмотрена вкладка JUnit с результатами тестирования.

Для дальнейшего развития имеет смысл ознакомиться с основными классами тестовых сценариев:

`ActivityInstrumentationTestCase2` – класс функционального тестирования активности; `ActivityUnitTestCase` – класс, обеспечивающий изолированное

тестирование активности;

ProviderTestCase2 – класс, обеспечивающий тестирование контент-провайдера; ServiceTestCase – класс, обеспечивающий тестирование сервиса;

Assert – множество методов утверждений; MoreAsserts – дополнительные методы утверждений;

ViewAsserts – полезные методы для тестирования визуальных компонентов; TouchUtils – методы, симулирующие сенсорные события активности;

Instrumentation – базовый инструментальный класс; InstrumentationTestCase – базовый класс сценариев тестов;

InstrumentationTestRunner – стандартный класс запуска Android тестов.

В конце лабораторной работы хочется обратить внимание на статьи, посвященные рассмотрению автоматического тестирования приложений для Android: <http://software.intel.com/ru-ru/articles/automated-android-application-testing>; <http://habrahabr.ru/post/219655/>.

Содержание отчета и его форма

Отчет к лабораторной работе оформляется в виде текстового документа по форме простого реферата и должен включать:

1. Название лабораторной работы.
2. Цель лабораторной работы.
3. Краткие выводы по результатам выполнения лабораторной работы согласно пунктам раздела «Методики выполнения работы».
4. Ответы на Вопросы к лабораторной работе.
5. Формулировку задания для самостоятельной работы и результат его выполнения.

Задания для самостоятельной работы

1. Создать тестирующее приложение на основе JUnit
2. Выполнить анализ создания тестирующего приложения на основе JUnit.

Вопросы к лабораторной работе

1. Характеристика инструментов тестирования в Android SDK.
2. Методика создания тестирующего приложения.
3. Методика запуска теста и просмотр результатов.

Лабораторная работа 3

Портирование приложений с использованием Intel XDK

Цель работы: освоить основные принципы переноса приложений на смартфон при помощи среды разработки Intel XDK.

Теоретическое обоснование

Исследование переносимого приложения и переработка интерфейса

Перед началом переноса существующего приложения на мобильную платформу Android необходимо очень хорошо продумать, как именно оно будет выглядеть в новых условиях.

Следует учитывать особенности как самой платформы, так и устройств, которые на ней базируются. Необходимо обязательно обратить внимание на следующие факторы:

1. Изменение размера окна приложения (как правило, в меньшую сторону).
2. Наличие новых возможностей управления приложением. В первую очередь речь идет о том, что у сенсорного экрана любая область является кликабельной, а, значит, в некоторых случаях можно отказаться от выделенных кнопок.

Конечно, если вы портируете небольшое приложение наподобие классического калькулятора, то его вид на мобильном устройстве практически не будет отличаться от десктопных аналогов (рис. 3.1):



Рисунок 3.1 – Приложение «Калькулятор» для разных операционных систем. Слева Калькулятор Windows 7, справа бесплатное приложение Shake Calc (скриншот из Google Play)

Однако при внимательном рассмотрении можно заметить существенное отличие: в мобильной версии отсутствует строка меню. Возможность доступа к дополнительным настройкам реализована с использованием возможностей смартфона. Так, для доступа к дополнительным опциям можно переключиться в режим функций, просто проведя пальцем по экрану. Настройки приложения вызываются нажатием на специальную клавишу (аппаратную во многих моделях телефонов).

Если главное окно оригинального приложения больше калькулятора, то при переносе его на мобильное устройство придется основательно переработать интерфейс. Проще всего разработать интерфейс мобильного приложения «с нуля» и уже потом, сравнивая интерфейс исходного и нового приложений, добавить в последнее возможности для наиболее полного соответствия обеих версий.

Задания к лабораторной работе

Задание 1. Перенос интерфейса в среду Intel XDK с учетом особенностей мобильного устройства

Прежде всего необходимо создать новое приложение в Intel XDK. Для этого необходимо при создании нового проекта выбрать пункт меню Start with a Template (рис. 3.2).

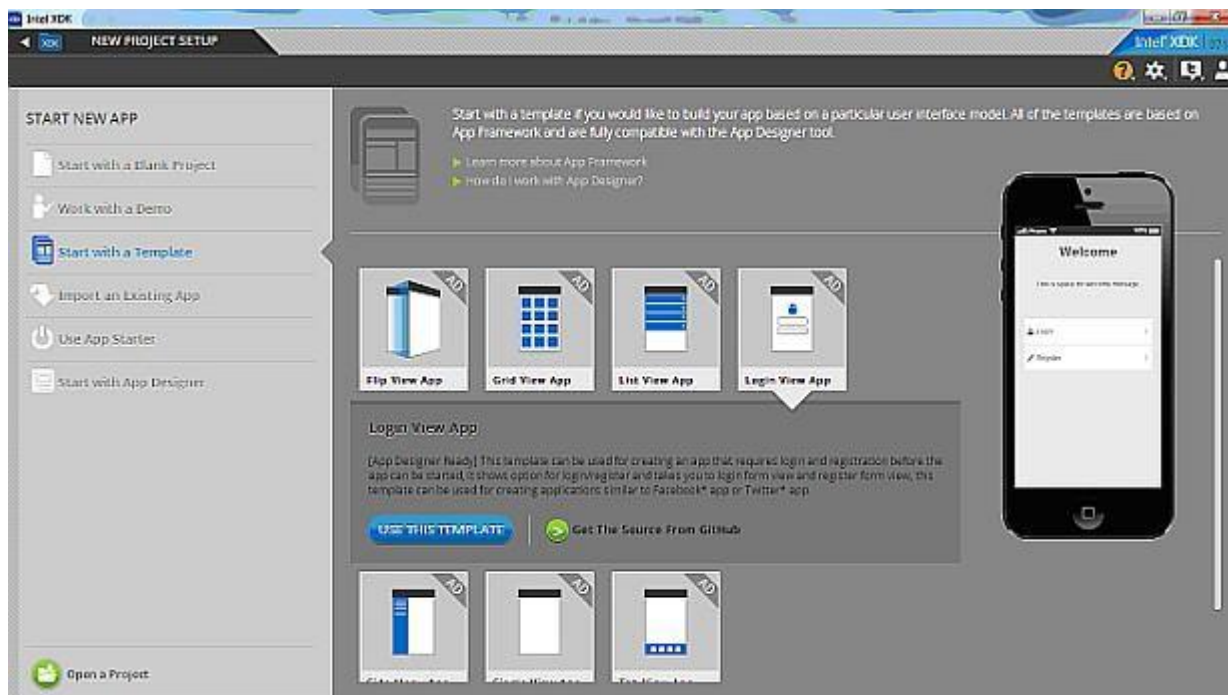


Рисунок 3.2 – Создание проекта из шаблона

С развитием среды список шаблонов должен увеличиться. На современном этапе доступны следующие семь шаблонов:

3. Flip View App – предназначено для небольших приложений, содержащих только две экранные сущности (главную и окно настроек).

4. Grid View App подходит для разработки приложений, позволяющих просматривать фотографии и другие объекты.

List View App – для приложений, содержащих списки с возможностью просмотра подробностей для каждого конкретного элемента (например, почтовые приложения, ленты новостей и т.д.).

Login View App – для приложений, требующих обязательной предварительной регистрации.

5. Side Menu App позволяет разрабатывать приложения, содержащие несколько экранов, при этом переключение между ними осуществляется посредством меню, которое вызывается при помощи жеста вытягивания его из одной из сторон экрана.

6. Single View App – для простых одноэкранных приложений.

Tab View App – приложения, позволяющие переключаться между

вкладками как с помощью нажатий на кнопки, так и посредством перелистывания.

Подумайте, какой из перечисленных выше способов наиболее удобен для портируемого приложения, и создайте соответствующий проект.

Если же вам повезло и приложение, которое вы собираетесь перенести на мобильную платформу, уже является HTML5-приложением, мы можете сразу использовать инструмент Import an Existing App (рис. 3.3).

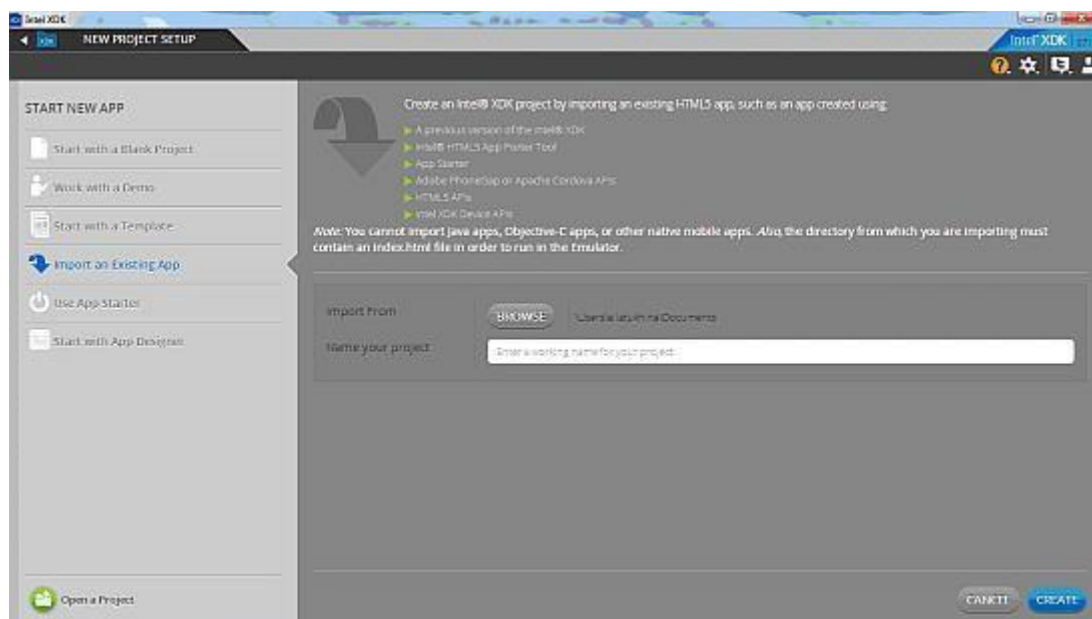


Рисунок 3.3 – Создание проекта из существующего HTML5-приложения

Задание 2. Организация функционирования приложения

После разработки нового интерфейса необходимо исследовать код исходного приложения. Если оно было разработано с соблюдением принципов объектно-ориентированного программирования, то отделение интерфейсных элементов от логики его работы не должно вызвать особых проблем. Если же эти принципы не слишком соблюдались, то могут возникнуть большие сложности с выделением логики работы приложения.

Для использования имеющегося кода в Intel XDK необходимо перевести его на подходящий язык. Поскольку в HTML5-приложениях для описания интерфейса используются HTML5 и CSS, а для обеспечения логики работы скриптовый язык JavaScript, разделенный код необходимо переносить следующим образом. Все, что имеет отношение к интерфейсу, описывается на

HTML5 с использованием CSS. Для ускорения процесса портирования можно использовать встроенный редактор дизайна, который имеется в Intel XDK. Для работы в этом режиме необходимо на вкладке Developer перейти из режима CODE в режим DESIGN. В этом режиме можно добавлять управляющие элементы и редактировать их свойства. В разделе PAGES здесь можно переключаться между экранными сущностями, если их несколько, и редактировать каждую из них (рис. 3.4).

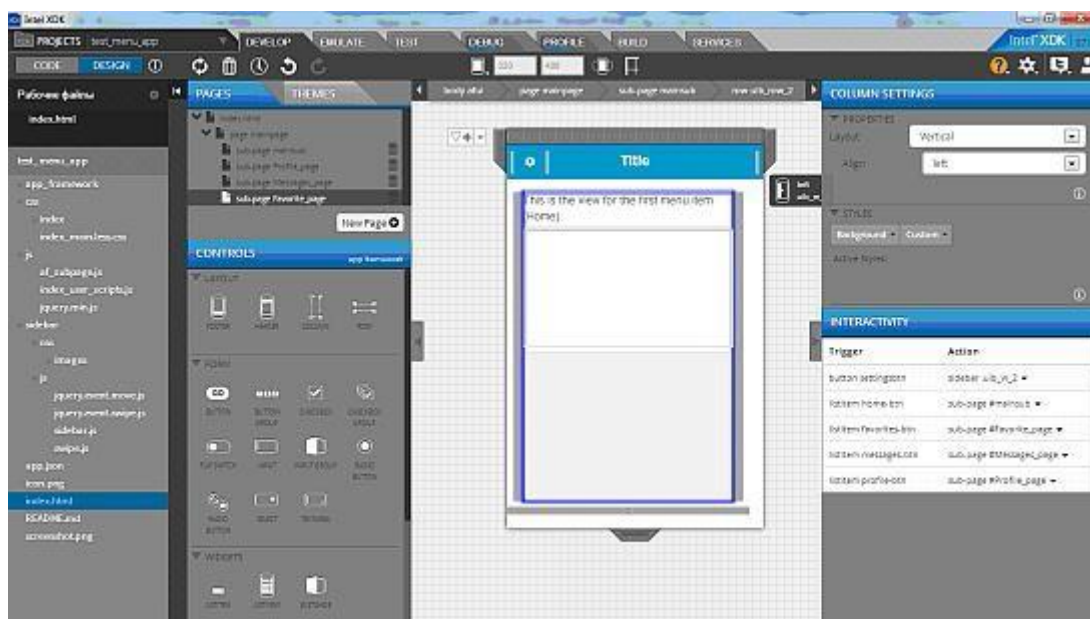


Рисунок 3.4 – Редактирование проекта в режиме дизайнера

Для переноса логики работы приложения необходимо перевести существующий код на JavaScript. Основные конструкции JavaScript произошли из языка C, поэтому, если исходный код портируемого приложения был написан на языке из C-семейства, перевод его на JavaScript будет гораздо проще. Можно попробовать использовать готовые трансляторы, которые позволяют переводить код с одного языка программирования на другой. Конечно, после автоматической трансляции все равно придется править код, но, по крайней мере, количество трудоемкой рутинной работы значительно сокращается.

В качестве транслятора с языка Java можно использовать проект GWT. Компилятор GWT переведёт код Java-приложения в соответствующий браузеру JavaScript, HTML и CSS. Для перевода кода с других языков можно использовать аналогичные решения.

Содержание отчета и его форма

Отчет к лабораторной работе оформляется в виде текстового документа по форме простого реферата и должен включать:

1. Название лабораторной работы.
2. Цель лабораторной работы.
3. Краткие выводы по результатам выполнения лабораторной работы согласно пунктам раздела «Методики выполнения работы».
4. Ответы на Вопросы к лабораторной работе.
5. Формулировку задания для самостоятельной работы и результат его выполнения.

Задания для самостоятельной работы

1. Подумайте, какие особенности мобильного устройства могут повлиять на появление дополнительных возможностей в работе вашего приложения.
2. Попробуйте реализовать эти возможности.

Вопросы к лабораторной работе

1. Понятия «переносимого приложения» и «переработка интерфейса».
2. Методика переноса интерфейса в среду Intel XDK с учетом особенностей мобильного устройства.
3. Методика организации функционирования приложения.

Лабораторная работа 4

Использование мобильной связи в приложениях для смартфона

Цель работы: научиться работать с мобильной связью в приложениях для смартфона.

Теоретическое обоснование

Приложение, описанное ниже, позволяет выполнять телефонные звонки через встроенное приложение для звонков, используя при этом собственный список контактов. Конечно, то же самое можно сделать вручную или добавить эти номера в адресную книгу телефона. Однако описанный способ может пригодиться в некоторых ситуациях, например, в качестве части внутреннего приложения для хранения заранее определенного списка номеров сотрудников компании. В этом случае пользователь, используя корпоративное приложение, может вызывать нужные номера со своего личного смартфона, не добавляя контакты в адресную книгу. В некоторых случаях таких номеров может быть очень много, и специальное приложение позволит использовать рабочий справочник, не смешивая служебную информацию с личной.

Задания к лабораторной работе

Мы создадим приложение, содержащее несколько кнопок. Нажатие на кнопку вызывает конкретный телефонный номер, проассоциированный с ней. Для простоты понимания в программе не используются базы данных, все нужные сведения «защиты» в код приложения. При желании несложно доработать программу и вынести эту информацию в отдельный файл.

Создайте проект SimpleDialer.

Рекомендуемые настройки представлены на рисунке 4.1.

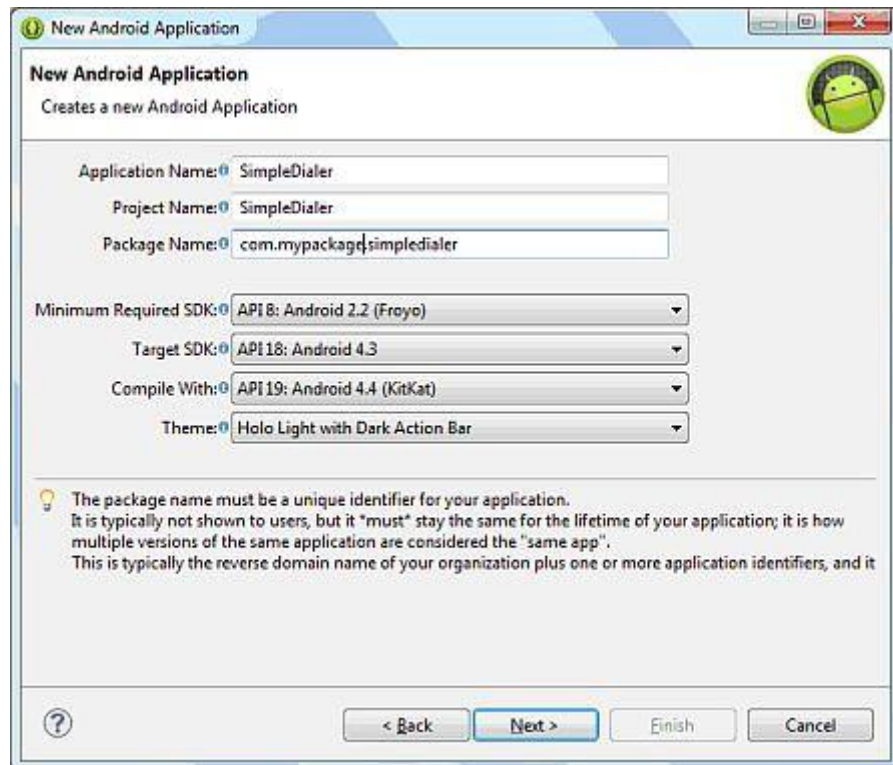


Рисунок 4.1 – Создание нового приложения

Добавьте следующую строчку в файл strings.xml:

```
<string name=«main_label»>My Friends</string>
```

В файле layout.xml создайте объекты TextView и шесть кнопок:

```
<LinearLayout
xmlns:android=«http://schemas.android.com/apk/res/android»
    android:layout_width=«fill_parent»
    android:layout_height=«fill_parent»
    android:orientation=«vertical»
    android:padding=«15dip» >
    <TextView
        android:layout_width=«wrap_content»
        android:layout_height=«wrap_content»
        android:layout_gravity=«center»
        android:layout_marginBottom=«25dip»
        android:text=«@string/main_label» android:textSize=«22sp»
    />
    <Button
```

```
        android:id=«@+id/button1»  
        android:layout_width=«fill_parent»  
        android:layout_height=«wrap_content»  
        android:textSize=«18sp» />
```

```
        ...//здесь добавить описания остальных кнопок  
</LinearLayout>
```

Перейдите в файл MainActivity.java. Создайте переменную для хранения числа записей и два массива строк:

```
        private int entries = 6; private String phoneNum[];  
        private String buttonLabels[];
```

Так как кнопки будут реагировать на нажатие, добавьте в заголовок класса implements OnClickListener (нужно также добавить import android.view.View.OnClickListener;).

В методе onCreate() пропишите:

```
        phoneNum = new String[entries]; buttonLabels = new  
        String[entries];
```

и создайте шесть кнопок:

```
        Button button1 = (Button)findViewById(R.id.button1);  
        button1.setText(buttonLabels[0]);  
        button1.setOnClickListener(this);  
  
        Button button2 = (Button)findViewById(R.id.button2);  
        button2.setText(buttonLabels[1]);  
        button2.setOnClickListener(this)
```

...

Создайте новый метод lunchDialer():

```
        public void launchDialer(String number){  
            String numberToDial = «tel:»+number;  
            startActivity(new Intent(Intent.ACTION_DIAL,  
            Uri.parse(numberToDial)));  
        }
```

Данный метод отвечает за вызов встроенного телефонного приложения, которое позволяет совершать звонки. Он получает номер телефона в качестве параметра и создает специально отформатированную строку `numberToDial`, которая является универсальным идентификатором (Uniform Resource Identifier). Далее создается новый интент со стандартным действием `ACTION_DIAL` (звонок), который используется для старта активности, отвечающей за звонки.

Заполните массивы через метод `populateArrays()` (не забудьте вызвать его в начале):

```
public void populateArrays() {
    phoneNum[0] = «123-456-78-90»;
    phoneNum[1] = «234-567-89-01»;
    phoneNum[2] = «345-678-90-12»;
    phoneNum[3] = «456-789-01-23»;
    phoneNum[4] = «567-890-12-34»;
    phoneNum[5] = «678-901-23-45»;

    buttonLabels[0] = «Иванов Ваня»; buttonLabels[1] =
«Петров Петя»; buttonLabels[2] = «Семеныч Сеня»;
buttonLabels[3] = «Кузнецова Катя»; buttonLabels[4] =
«Смирнова Саша»; buttonLabels[5] = «Попова Полина»;
}
```

Далее опишите метод `onClick()`:

```
@Override
public void onClick(View v) {
    switch (v.getId()) {
        case R.id.button1:
            launchDialer(phoneNum[0]); break;
        case R.id.button2:
            launchDialer(phoneNum[1]);
            break;
        case R.id.button3:
            launchDialer(phoneNum[2]);
```

```
break;
case R.id.button4:
    launchDialer(phoneNum[3]);
break; case R.id.button5:
    launchDialer(phoneNum[4]); break;
case R.id.button6:
    launchDialer(phoneNum[5]); break;
}
```

Приложение можно запускать (рис. 4.2), с его помощью можно совершать звонки (рис. 4.3).

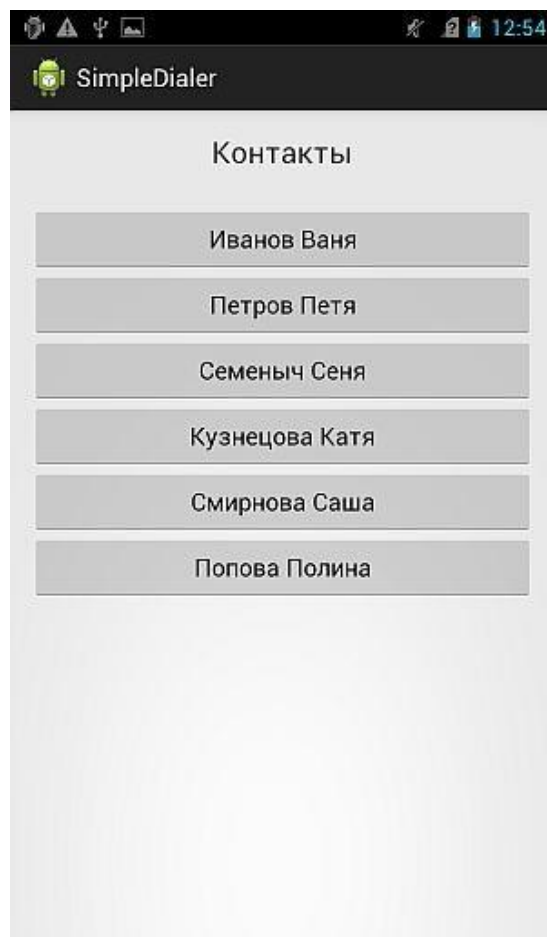


Рисунок 4.2 – Приложение, запущенное на устройстве

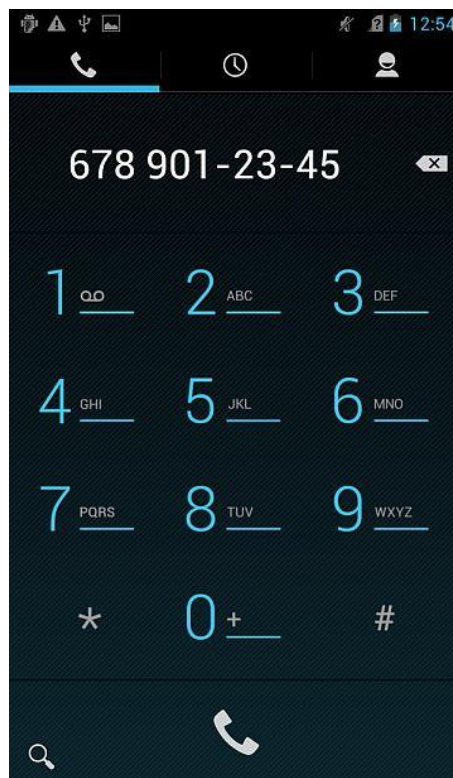


Рисунок 4.3 – С помощью приложения можно совершать звонки

Задания для самостоятельной работы

Подумайте, как можно переработать приложение, чтобы можно было хранить произвольное (не заданное заранее) количество контактов, добавлять и удалять контакты и сопутствующую информацию. Попробуйте реализовать появившиеся идеи. В качестве хранилища данных можно использовать базу данных SQLite.

Листинг 1. Файл strings.xml

```
<?xml version=«1.0» encoding=«utf-8»?>
<resources>
<string name=«app_name»>SimpleDialer</string>
<string name=«action_settings»>Settings</string>
<string name=«hello_world»>Hello world!</string>
<string name=«hello»>SimpleDialer</string>
<string name=«main_label»>My Friends</string>
</resources>
```


Листинг 2. Файл activity_main.xml

```
<LinearLayout
xmlns:android=«http://schemas.android.com/apk/res/android
« android:layout_width=«fill_parent»
android:layout_height=«fill_parent»
    android:orientation=«vertical»
    android:padding=«15dip» >
    <TextView
        android:layout_width=«wrap_content»
        android:layout_height=«wrap_content»
        android:layout_gravity=«center»
        android:layout_marginBottom=«25dip»
        android:text=«@string/main_label»    android:textSize=«22sp»
    />

    <Button
        android:id=«@+id/button1»
        android:layout_width=«fill_parent»
        android:layout_height=«wrap_content»
        android:textSize=«18sp» />

    <Button
        android:id=«@+id/button2»
        android:layout_width=«fill_parent»
        android:layout_height=«wrap_content»
        android:textSize=«18sp» />

    <Button
        android:id=«@+id/button3»
        android:layout_width=«fill_parent»
        android:layout_height=«wrap_content»
        android:textSize=«18sp» />

    <Button
        android:id=«@+id/button4»
```

```

        android:layout_width=«fill_parent»
android:layout_height=«wrap_content»
android:textSize=«18sp» />
        <Button
        android:id=«@+id/button5»
android:layout_width=«fill_parent»
android:layout_height=«wrap_content»
android:textSize=«18sp» />
        <Button
        android:id=«@+id/button6»
        android:layout_width=«fill_parent»
        android:layout_height=«wrap_content»
        android:textSize=«18sp» />
</LinearLayout>

```

Листинг 3. Файл MainActivity.java

```

package mypackage.simplifiedialer;

import android.os.Bundle; import android.app.Activity;
import android.view.Menu; import android.content.Intent;
import android.net.Uri;

import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class MainActivity extends Activity implements
OnClickListener { private int entries = 6;
    private    String    phoneNum[];    private    String
buttonLabels[];

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}

```

```

        phoneNum = new String[entries]; buttonLabels = new
String[entries];
        populateArrays();
        Button button1 = (Button)findViewById(R.id.button1);
button1.setText(buttonLabels[0]);
button1.setOnClickListener(this);
        Button button2 = (Button)findViewById(R.id.button2);
button2.setText(buttonLabels[1]);
button2.setOnClickListener(this);
        Button button3 = (Button)findViewById(R.id.button3);
button3.setText(buttonLabels[2]);
button3.setOnClickListener(this);
        Button button4 = (Button)findViewById(R.id.button4);
button4.setText(buttonLabels[3]);
button4.setOnClickListener(this);
        Button button5 = (Button)findViewById(R.id.button5);
button5.setText(buttonLabels[4]);
button5.setOnClickListener(this);
        Button button6 = (Button)findViewById(R.id.button6);
button6.setText(buttonLabels[5]);
button6.setOnClickListener(this);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.main, menu); return true;
    }

    public void launchDialer(String number){ String
numberToDial = «tel:»+number;
        startActivity(new Intent(Intent.ACTION_DIAL,
Uri.parse(numberToDial)));
    }

```

```

    phoneNum[0] = «123-456-78-90»;
    phoneNum[1] = «234-567-89-01»;
    phoneNum[2] = «345-678-90-12»;
    phoneNum[3] = «456-789-01-23»;
    phoneNum[4] = «567-890-12-34»;
    phoneNum[5] = «678-901-23-45»;
    buttonLabels[0] = «Иванов Ваня»;
    buttonLabels[1] = «Петров Петя»;
    buttonLabels[2] = «Семеныч Сеня»;
    buttonLabels[3] = «Кузнецова Катя»; buttonLabels[4] =
«Смирнова Саша»; buttonLabels[5] = «Попова Полина»; }

    @Override
    public void onClick(View v) {
        switch (v.getId()) {
            case R.id.button1:
                launchDialer(phoneNum[0]); break;
            case R.id.button2:
                launchDialer(phoneNum[1]);
                break;
            case R.id.button3:
                launchDialer(phoneNum[2]);
                break;
            case R.id.button4:
                launchDialer(phoneNum[3]);
                break;
            case R.id.button5:
                launchDialer(phoneNum[4]);
                break;

            case R.id.button6:
                launchDialer(phoneNum[5]); break;
        }
    }
}

```

}
}
}

Содержание отчета и его форма

Отчет к лабораторной работе оформляется в виде текстового документа по форме простого реферата и должен включать:

1. Название лабораторной работы.
2. Цель лабораторной работы.
3. Краткие выводы по результатам выполнения лабораторной работы согласно пунктам раздела «Методики выполнения работы».
4. Ответы на Вопросы к лабораторной работе.
5. Формулировку задания для самостоятельной работы и результат его выполнения.

Задания для самостоятельной работы

1. Подумайте, какие особенности мобильного устройства могут повлиять на появление дополнительных возможностей в работе вашего приложения.
2. Попробуйте реализовать эти возможности.

Вопросы к лабораторной работе

1. Технология создания объекта TextView.
2. Технология создания переменной для хранения числа записей.
3. Характеристика метода populateArrays().
4. Характеристика метода onClick().

Лабораторная работа 5

Управление воспроизведением аудио

Цель работы: научиться добавлять в приложения, поддерживающие воспроизведения аудио, возможностей:

1. управления громкостью и воспроизведением звука с помощью соответствующих кнопок устройства;
2. управления аудио фокусом и адекватного реагирования на изменения аудио фокуса, вызванные системой или другими приложениями.

Теоретическое обоснование

Одной из особенностей смартфонов является тот факт, что для большинства их владельцев не последнюю роль играет возможность использования этого «умного телефона» в качестве аудио или видеоплеера, поэтому современные устройства становятся все более и более мультимедийными. В состав платформы Android входит набор библиотек для обработки мультимедиа Media Framework, в котором реализована поддержка большинства общих медиа-форматов. В связи с чем, в приложения, разрабатываемые для смартфонов под управлением Android, можно интегрировать запись и воспроизведение аудио и видео, а также возможность распознавания речи.

Идеи естественно-интуитивного взаимодействия человека с компьютером (perceptual computing), предполагают способность вычислительных устройств понимать человека через физический контакт, словесные команды и жесты. Разработчикам программного обеспечения использование новых идей позволит создавать приложения, вносящие ощущение присутствия и погружения. Такого эффекта можно добиться реализацией возможностей отслеживания жестов рук и пальцев, анализа лица (мимики), распознавания голоса (речи), отслеживания перемещений 2D/3D объектов. Умение разрабатывать такие приложения в ближайшем будущем будет востребовано. Intel® Perceptual Computing SDK позволяет добавлять элементы естественно-интуитивного взаимодействия в приложения и снизить стоимость разработки таких приложений.

Основы работы с аудио и видеoinформацией 1.Воспроизведение аудио

И ВИДЕО

Первый вопрос, который будет интересовать нас в данной лекции, связан с возможностью создания приложений, воспроизводящих аудио и видеoinформацию. Android API содержит класс MediaPlayer, упрощающий разработку таких приложений. Мультимедиа библиотека Android включает поддержку воспроизведения множества наиболее распространенных форматов, что позволяет легко использовать в приложениях аудио, видео и изображения. Можно проигрывать аудио или видео из медиа файлов сохраненных как ресурсы приложения (raw ресурсы), из файлов, расположенных в файловой системе или из потока данных, получаемого через сетевое соединение, для всего этого используется MediaPlayer API. Актуальная информация о поддерживаемых форматах аудио и видео приводится по ссылке: <http://developer.android.com/guide/appendix/media-formats.html>

Для воспроизведения аудио и видео необходимо создать новый экземпляр класса MediaPlayer, задать источник медиа-информации и подготовить его к воспроизведению. Рассмотрим работу с аудио-данными. Самый простой способ инициализации плеера для воспроизведения аудио-контента заключается в использовании метода create(), в который в качестве параметров передаются контекст приложения и источник аудио-данных. `MediaPlayer mediaPlayer = MediaPlayer.create(context, source);` параметр source задает источник и может иметь одно из следующих значений:

- идентификатор ресурса;
- путь URI к локальному файлу (используя схему file://);
- путь URI к удаленному ресурсу с аудио, предоставленному в виде URL;
- путь URI к записи внутри локального Источника данных

Код в примере 1 демонстрирует разные источники инициализации медиаплеера. Необходимо отметить, что для объекта MediaPlayer, возвращаемого методом create(), уже был вызван метод prepare(), поэтому не надо вызывать его еще раз.

Пример 1.

```
MediaPlayer resPlayer =
MediaPlayer.create(getApplicationContext(),
R.raw.my_audio);

MediaPlayer filePlayer =
MediaPlayer.create(getApplicationContext(),
Uri.parse («file:///sdcard/localfile.mp3»));

MediaPlayer urlPlayer =
MediaPlayer.create(getApplicationContext(),
Uri.parse («http://site.com/audio/audio.mp3»));

MediaPlayer contPlayer =
MediaPlayer.create(getApplicationContext(),
Settings.System.DEFAULT_RINGTONE_URI);
```

Другой способ инициализации плеера для воспроизведения аудио-контента заключается в использовании метода `setDataSource()`, в этом случае сначала необходимо создать экземпляр класса `MediaPlayer`. В качестве параметра этот метод принимает путь к файлу, путь URI к источнику данных, адрес URL мультимедийного потока или файловый дескриптор. Используя данный подход, важно не забыть вызвать метод `prepare()` класса `MediaPlayer`, прежде чем начинать воспроизведение, как показано в примере 9.2 .

Пример 2.

```
MediaPlayer mediaPlayer = new MediaPlayer();
mediaPlayer.setDataSource («/sdcard/test.3gp»);
mediaPlayer.prepare();
```

Если вы передаете адрес URL, который ссылается на файл в Интернете, удаленный сервер должен поддерживать последовательную загрузку с помощью протоколов RTSP или HTTP.

Рассмотрим работу с видео-данными. Первым делом при использовании медиаплеера с целью воспроизведения видео необходимо подготовить поверхность, на которой это видео будет отображаться. Поэтому необходимо, чтобы окно активности приложения содержало элемент `SurfaceView`, а

медиаплеер получил объект класса `SurfaceHolder` с помощью метода `setDisplay()`, как показано в примере 3.

Пример 3.

```
MediaPlayer mediaPlayer = new MediaPlayer();
mediaPlayer.setDisplay(((SurfaceView) findViewById(R.id.su
rfaceView1)).getHolder());
mediaPlayer.setDataSource(«/sdcard/test2.3gp»);
mediaPlayer.prepare();
```

Инициализация видео-контента использует метод `setDataSource()`, как показано в примере

3. В отличие от ресурсов с аудиоданными, Android не поддерживает проигрывание видео, включенного в проект в качестве ресурса. А также нет возможности использовать статический метод `create()` для упрощенного создания объекта `MediaPlayer` и путь URI, ссылающийся на локальный файл с помощью схемы `file://`.

После подготовки медиаплеера можно начать воспроизведение соответствующих мультимедийных данных, для запуска воспроизведения используется метод: `mediaPlayer.start()`;

Процессом воспроизведения можно управлять с помощью методов `stop()` и `pause()`, позволяющих остановить или приостановить воспроизведение, и метода `start()`, позволяющего снова запустить воспроизведение.

Класс `MediaPlayer` также предоставляет методы `getDuration()` — позволяет получить длину воспроизводимого аудио или видео и `getCurrentPosition()` — возвращает текущую позицию. Чтобы перейти к определенной позиции, необходимо использовать метод `seekTo()`. В процессе воспроизведения медиаплеер позволяет регулировать громкость:

```
mediaPlayer.setVolume(1f, 0.5f);
```

метод `setVolume()` позволяет управлять громкостью каждого аудиоканала. В качестве параметров этот метод принимает скалярные значения с плавающей точкой в диапазоне между 0 и 1 для обоих каналов (где 0 — полная тишина, а 1 — максимальная громкость).

В процессе воспроизведения медиаплеер позволяет уберечь подсветку экрана от автоматического отключения с помощью метода `setScreenOnWhilePlaying()`.

```
mediaPlayer.setScreenOnWhilePlaying(true);
```

В случае, если в приложении предполагается сохранение экрана и процессора от засыпания, или использование методов класса `MediaPlayer`:

`setWakeMode()` и `setScreenOnWhilePlaying()`, необходимо добавить в файл манифеста следующую строку: `<uses-permission android:name=«android.permission.WAKE_LOCK» />`

Медиаплеер предоставляет возможность проигрывать файлы в режиме повторения для этого существуют методы `isLooping()` и `setLooping()`:

```
if (!mediaPlayer.isLooping())
mediaPlayer.setLooping(true);
```

В настоящее время нельзя воспроизводить аудиоданные одновременно с разговором по телефону. Медиаплеер всегда использует стандартное устройство для вывода звука – динамик или подключенную по Bluetooth гарнитуру. При воспроизведении видеоресурсов можно использовать метод `getFrame()`, чтобы получить заданный кадр.

При завершении процесса воспроизведения необходимо вызвать метод `release()` класса `MediaPlayer`, чтобы освободить соответствующие ресурсы:

```
mediaPlayer.release();
```

Android поддерживает ограниченное число одновременно работающих объектов `MediaPlayer`. Если не выполнять освобождение ресурсов, можно столкнуться с выбросом исключения, когда система исчерпает ресурсы.

2. Запись аудиои видео-данных

Следующий вопрос, который будет интересовать нас в данной лекции, связан с возможностью разработки приложений, позволяющих записывать аудиои видео-контент.

Самый простой способ записи видео заключается в использовании `intent`-объектов для запуска стандартного приложения, управляющего камерой. Для запуска записи видео необходимо создать новый `intent`-объект, используя

константу ACTION_VIDEO_CAPTURE класса MediaStore:

```
Intent intent = new  
Intent(MediaStore.ACTION_VIDEO_CAPTURE);
```

Можно (но не обязательно) настроить пару свойств intent-объекта:

место сохранения видеозаписи, если есть необходимость сохранить ее в месте, отличном от используемого по умолчанию, применяют константу EXTRA_OUTPUT класса MediaStore:

```
intent.putExtra(MediaStore.EXTRA_OUTPUT, outputpath);
```

outputpath – указывает альтернативный путь URI;

качество видеозаписи, если есть необходимость сохранить картинку в качестве, отличном от используемого по умолчанию, применяют константу EXTRA_VIDEO_QUALITY класса MediaStore:

```
intent.putExtra(MediaStore.EXTRA_VIDEO_QUALITY,  
VIDEO_QUALITY); VIDEO_QUALITY — указывает качество видеозаписи,  
может принимать всего два значения 0 и 1, значение 0 задает низкое качество  
(подходит для MMS), значение 1 задает высокое качество. По умолчанию  
используется 1.
```

После создания и настройки intent-объекта необходимо вызвать метод startActivityForResult() и передать ему полученный Intent в качестве параметра: startActivityForResult(intent, RECORD_VIDEO);

Этот код запускает стандартную активность, управляющую видеокамерой и позволяющую начать, остановить, просмотреть и повторить запись видео. При этом нет необходимости создавать собственное приложение для этих нужд.

Другой способ записи мультимедийных файлов состоит в использовании класса MediaRecorder, который позволяет указать источник для аудио и видео, формат итогового файла, а также кодировщики, используемые для записи. Чтобы иметь возможность записывать мультимедийные файлы в Android, приложение должно получить полномочия RECORD_AUDIO и/или RECORD_VIDEO. Для этого необходимо добавить в манифест приложения следующие строки:

```
<uses-permission  
android:name=«android.permission.RECORD_AUDIO»/>  
<uses-permission  
android:name=«android.permission.RECORD_VIDEO»/>
```

Для управления записью медиа-контента с помощью класса `MediaRecorder` необходимо выполнить следующие действия в заданной последовательности:

создать экземпляр класса `MediaRecorder`:

```
MediaRecorder mediaRecorder = new MediaRecorder();
```

определить источник записи с помощью методов `setAudioSource()` и `setVideoSource()`, которым в качестве параметра передаются статические константы классов `MediaRecorder.AudioSource` и `MediaRecorder.VideoSource`, определяющие источники для аудио и видеоданных соответственно:

```
mediaRecorder.setAudioSource(MediaRecorder.AudioSource.  
MIC);  
mediaRecorder.setVideoSource(MediaRecorder.VideoSource.CA  
MERA);
```

здать итоговый формат с помощью метода `setOutputFormat()`, передав ему одну из констант класса `MediaRecorder.OutputFormat`:

```
mediaRecorder.setOutputFormat(MediaRecorder.OutputFor  
mat.DEFAULT);
```

определить кодировщики для аудио и видео с помощью методов `setAudioEncoder()` и `setVideoEncoder()`, используя константы класса `MediaRecorder.AudioEncoder` и `MediaRecorder.VideoEncoder`, соответственно.

```
mediaRecorder.setAudioEncoder(MediaRecorder.AudioEnco  
der.DEFAULT);  
mediaRecorder.setVideoEncoder(MediaRecorder.VideoEncoder.  
DEFAULT);
```

При желании можно установить частоту кадров и размер итогового видео.

выбрать итоговый файл с помощью метода `setOutputFile()`:

```
mediaRecorder.setOutputFile(«/sdcard/myoutputfile.mp4
```

») ;

После выполнения вышеперечисленных действий, получим экземпляр класса `MediaRecorder`, настроенный для записи аудио и видеоданных с микрофона и камеры, соответственно, при этом использованы итоговый формат и кодировщики по умолчанию, а запись будет сохранена на карту SD.

подготовить к записи с помощью метода `prepare()`:

```
mediaRecorder.prepare();
```

начать запись с помощью метода `start()`:

```
mediaRecorder.start();
```

завершить запись с помощью метода `stop()` – завершение записи, и метода `release()` – освобождение ресурсов объекта `MediaRecorder`:

```
mediaRecorder.stop(); mediaRecorder.release();
```

Метод `setOutputFile()` нужно вызвать до метода `prepare()` и после метода `setOutputFormat()`, иначе произойдет выброс исключения `IllegalStateException`.

В процессе записи видео, как правило, предполагается предварительный просмотр записываемой картинки. Метод `setPreviewDisplay()` позволяет задать объект `Surface` для показа видеопотока во время записи. Работает данный подход так же, как воспроизведение видео с помощью медиаплеера, описанное ранее.

```
mediaRecorder.setPreviewDisplay((SurfaceView) findViewById(R.id.surface).getHolder().getSurface());
```

```
mediaRecorder.prepare();
```

После настройки предварительного просмотра необходимо вызвать метод `prepare()`, после этого на экране в режиме реального времени начнет отображаться записываемый видеопоток.

Добавление мультимедийных файлов в `MediaStore`

По умолчанию мультимедийные файлы, созданные любым приложением, недоступны для других программ, чтобы предоставить доступ к таким файлам необходимо добавить их в

`MediaStore`.

Android предоставляет два способа добавления данных в `MediaStore`:

Первый способ предполагает использование сервиса

MediaScannerConnection, который автоматически распознает и внесет ваш файл в MediaStore.

Для установки связи с сервисом MediaScannerConnection необходимо вызвать метод connect(), чтобы получить уведомление об успешной установке связи необходимо реализовать интерфейс MediaScannerConnectionClient. В реализации необходимо прописать методы onMediaScannerConnected() и onScanCompleted(), первый вызывается системой, когда связь с сервисом установлена, второй — когда завершен процесс сканирования. В первом методе вызываем метод scanFile(), чтобы начать сканирование, во втором — вызываем метод disconnect(), чтобы разорвать связь с сервисом. В пример 4 показан каркас для создания нового класса MediaScannerConnectionClient.

Пример 4.

```
MediaScannerConnectionClient mediaScannerClient = new
MediaScannerConnectionClient() {
    private MediaScannerConnection msc = null;
    msc = new
MediaScannerConnection(getApplicationContext(), this);
    msc.connect();
    public void onMediaScannerConnected() {
msc.scanFile(«/sdcard/test1.jpg», null);
    }
    public void onScanCompleted(String path, Uri uri) {
msc.disconnect();
    }
};
```

Второй способ предполагает создание нового объекта ContentValues и добавление его в соответствующий источник данных. В качестве метаданных для нового медиафайла нужно указывать название, временную отметку (timestamp) и географическую информацию, как показано в пример 5.

Пример 5.

```
ContentValues content = new ContentValues(3);
```

```

        content.put(Audio.AudioColumns.TITLE,
«TheSoundandtheFury»);
        content.put(Audio.AudioColumns.DATE_ADDED,
System.currentTimeMillis()/1000);
        content.put(Audio.Media.MIME_TYPE, «audio/amr»);
        //указание абсолютного пути к добавляемому файлу
        content.put(MediaStore.Audio.Media.DATA,
«/sdcard/myoutputfile.mp4»);
        //доступ к объекту ContentResolver приложения
        ContentResolver resolver = getContentResolver();
        //вставка новой строки в MediaStore
        Uri uri =
resolver.insert(MediaStore.Video.Media.EXTERNAL_CONTENT_U
RI,
        content);
        //объявление о доступности добавленного файла
        sendBroadcast(new
Intent(Intent.ACTION_MEDIA_SCANNER_SCAN_FILE, uri));

```

Работа с несжатым звуком

Android API предоставляет классы `AudioTrack` и `AudioRecord`, позволяющие напрямую записывать аудио и воспроизводить аудиопоток в формате PCM, используя аппаратные возможности устройства. Рассмотрим процесс записи и воспроизведения «сырых» данных в формате PCM.

Для записи аудиоданных непосредственно с аппаратных буферов используется класс `AudioRecord`. Необходимо создать новый объект `AudioRecord`, указав источник, частоту, настройки каналов, кодировщик для аудио и размер буфера. Значения частоты, кодировщика аудио и настроек каналов повлияют на размер и качество записанного аудиофайла. `int frequency = 11025; //частота`

```

        int channelConfiguration =
AudioFormat.CHANNEL_CONFIGURATION_MONO; int audioEncoding

```

```

= AudioFormat.ENCODING_PCM_16BIT; //кодировщик
    //размер буфера
    int bufferSize =
AudioRecord.getMinBufferSize(frequency,
    channelConfiguration, audioEncoding);
    AudioRecord audioRecord = new AudioRecord(
MediaRecorder.AudioSource.MIC, frequency,
    channelConfiguration, audioEncoding, bufferSize);

```

Необходимо в манифест приложения добавить полномочие RECORD_AUDIO.

```

<uses-permission
android:name=«android.permission.RECORD_AUDIO»/>

```

После инициализации объекта AudioRecord необходимо вызвать метод startRecording(), чтобы начать асинхронную запись. Для добавления необработанных аудиоданных в записываемый буфер используется метод read():

```

short[] buffer = new short[bufferSize];
audioRecord.startRecording();
while (isRecording) {
    //заполнение буфера данными
    int bufferReadResult = audioRecord.read(buffer, 0,
bufferSize);
}

```

Для обработки входящего аудиопотока и воспроизведения его практически в режиме реального времени, используется класс AudioTrack. Этот класс дает возможность манипулировать входящим и исходящим звуком, обрабатывая на устройстве несжатые аудиоданные.

Необходимо создать новый объект AudioTrack, указав потоковый режим, частоту, параметры каналов, тип кодировщика и длину аудио.

```

//частота, увеличивает скорость проигрывания вдвое int
frequency = 11025/2;
    int channelConfiguration =

```



```

AudioFormat.CHANNEL_CONFIGURATION_MONO;
    int audioEncoding = AudioFormat.ENCODING_PCM_16BIT;
    AudioTrack          audioTrack          =          new
AudioTrack(AudioManager.STREAM_MUSIC,
    frequency, channelConfiguration, audioEncoding,
    audioLength, AudioTrack.MODE_STREAM);

```

Поскольку аудиоданные необработанные, отсутствует метаданная, связанная с ними. Поэтому важно установить корректные свойства, чтобы они совпадали с теми, которые были использованы при записи файла.

```

    File          file          =          new
File(Environment.getExternalStorageDirectory(),
«raw.pcm»);

    //для хранения аудиоданных массив типа short
    //(звук 16-битный, поэтому выделяем по 2 байта на
значение)

    int audioLength = (int)(file.length()/2);
    short[] audio = new short[audioLength];
    DataInputStream dis = new DataInputStream(
new BufferedInputStream(new FileInputStream(file)));
    int i = 0;
    while (dis.available() > 0) {
        audio[audioLength] = dis.readShort();
        i++;
    }
    // Закрытие входящих потоков dis.close();

```

После инициализации объекта AudioTrack, чтобы начать асинхронное воспроизведение, необходимо вызвать метод play(). Метод write() позволяет добавить «сырые» аудиоданные в буфер проигрывателя.

```

    audioTrack.play();          audioTrack.write(audio,          0,
audioLength);

```

Запись в буфер объекта AudioTrack можно начать, как до вызова метода

play, так и после. В первом случае воспроизведение пойдет сразу после вызова, во втором – звук станет проигрываться, как только данные запишутся в буфер AudioTrack.

Распознавание речи

Начиная с версии 1.5 (API Level 3) Android поддерживает голосовой ввод и распознавание речи и позволяет встраивать в приложения голосовой ввод.

Для распознавания речи необходимо создать intent-объект, используя константу ACTION_RECOGNIZE_SPEECH класса RecognizerIntent:

```
Intent intent = new  
Intent (RecognizerIntent.ACTION_RECOGNIZE_SPEECH);  
языковую модель распознавания входящего звука:  
intent.putExtra (RecognizerIntent.EXTRA_LANGUAGE_MODEL,  
RecognizerIntent.LANGUAGE_MODEL_FREE_FORM);
```

зададим строку, которая будет отображаться в окне голосового ввода, она должна предлагать пользователю сказать что-нибудь в микрофон:

```
intent.putExtra (RecognizerIntent.EXTRA_PROMPT, «speak  
now»);
```

ограничим количество результатов, которые могут быть возвращены при распознавании, зададим 1:

```
intent.putExtra (RecognizerIntent.EXTRA_MAX_RESULTS,  
1);
```

укажем языковую константу из класса Locale, чтобы задать язык для ввода, отличный от языка по умолчанию на данном устройстве:

```
intent.putExtra (RecognizerIntent.EXTRA_LANGUAGE,  
Locale.ENGLISH);
```

Можно получить текущее значение по умолчанию, вызвав статический метод getDefault() из класса Locale.

После формирования intent-объекта передадим его в качестве параметра в метод startActivityForResult(), который вызывает стандартное диалоговое окно для распознавания речи:

```
startActivityForResult(intent, VOICE_RECOGNITION);
```

После завершения голосового ввода, движок для распознавания речи проанализирует и обработает записанный звук. Для получения результатов необходимо переопределить метод `onActivityResult()` активности, из которой был отправлен запрос на распознавание речи. В примере 6 показано переопределение метода `onActivityResult()`.

Пример 6.

```
@Override
protected void onActivityResult(int requestCode, int
resultCode,
Intent data){
    if (requestCode == VOICE_RECOGNITION && resultCode ==
RESULT_OK){ ArrayList<String> results;
        results = data.getStringArrayListExtra(
RecognizerIntent.EXTRA_RESULTS);
        // TODO Выполнить какие-то действия с распознанными
строками
    }
    super.onActivityResult(requestCode, resultCode,
data);
}
```

Задания к лабораторной работе

Задание 1. Управление громкостью и воспроизведением

Приложение, ориентированное на пользователя должно предоставлять ему привычный способ работы, если дело касается воспроизведения аудио контента, то пользователь вправе ожидать возможности управления громкостью звука с помощью клавиш, предусмотренных для этого на самом устройстве. В том числе, если на устройстве доступны кнопки управляющие самим процессом воспроизведения звука, то их использование должно вызывать соответствующие действия над аудиопотоком, используемым в приложении.

Для начала необходимо понять какой аудио поток будет использовать

приложение.

Android поддерживает отдельный аудио поток для воспроизведения музыки, входящих звонков, предупреждений, напоминаний, системных звуков. Это сделано, чтобы позволить пользователю управлять громкостью каждого потока независимо.

Большинство из этих потоков связывается с системными событиями, таким образом, если приложение не является заменой будильника, то аудио будет воспроизводиться в потоке `STREAM_MUSIC` (константа класса `AudioManager`).

По умолчанию кнопки управления громкостью изменяют громкость активного аудио потока, если в данный момент ничего не воспроизводится, регулируется громкость входящего звонка. Пользователь игрового приложения или аудиопроигрывателя, нажимая на кнопки управления громкостью, скорее всего ожидает изменения громкости игры или музыки, даже если в данный момент в приложении не воспроизводятся ни какие звуки. Android предоставляет метод `setVolumeControlStream()` для задания аудио потока, громкость которого будет регулироваться соответствующими кнопками устройства.

В метод `onCreate()` активности, в которой выполняется воспроизведение аудио, необходимо добавить строку:

```
setVolumeControlStream(AudioManager.STREAM_MUSIC);
```

На многих устройствах доступны кнопки управления воспроизведением: `play`, `pause`, `stop`, `skip` и `previous`. Как только пользователь нажимает на одну из этих кнопок, система рассылает сообщение о действии `ACTION_MEDIA_BUTTON`. Для обработки таких нажатий в приложении, необходимо в файле манифеста приложения зарегистрировать приемник широковещательных сообщений (`BroadcastReceiver`) следующим образом:

```
<receiver android:name=«.RemoteControlReceiver»>
  <intent-filter>
    <action
      android:name=«android.intent.action.MEDIA_BUTTON»/>
    </intent-filter>
  </receiver>
```

</example>

В реализации приемника необходимо определять какая кнопка была нажата и вызвала рассылку широковещательных сообщений. Intent-объект содержит эту информацию в ключе EXTRA_KEY_EVENT, а класс KeyEvent содержит набор констант KEYCODE_MEDIA_*, для представления каждой возможной медиа кнопки, например, KEYCODE_MEDIA_PLAY_PAUSE или KEYCODE_MEDIA_NEXT (полный список констант можно найти в описании класса KeyEvent).

Как получить информацию о нажатии медиа кнопки и выполнить соответствующие действия показывает следующий код:

```
public class RemoteControlReceiver extends
BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent)
    {
        if(Intent.ACTION_MEDIA_BUTTON.equals(intent.getAction()))
        {
            KeyEvent event =
            (KeyEvent)intent.getParcelableExtra(Intent.EXTRA_KEY_EVENT);
            if (KeyEvent.KEYCODE_MEDIA_PLAY == event.getKeyCode())
            {
                // реакция на нажатие кнопки
            }
        }
    }
}
```

Необходимо программно указывать, когда приложение может получать информацию о событиях нажатия медиа кнопок.

Сначала необходимо создать экземпляр класса AudioManager, который позволит регистрировать и отменять регистрацию приемника событий нажатия медиа кнопок: AudioManager am =

```
mContext.getSystemService(Context.AUDIO_SERVICE);
```

Далее необходимо зарегистрировать приемник:

```
//начать прослушивание нажатий кнопок  
am.registerMediaButtonEventReceiver(RemoteControlReceiver  
);
```

Зарегистрированный приемник широковещательных сообщений будет единственным приемником всех событий нажатий медиа кнопок, поэтому необходимо отменить регистрацию приемника, как только пропадет необходимость в нем:

```
//закончить прослушивание нажатий кнопок  
am.unregisterMediaButtonEventReceiver(RemoteControlReceiver);
```

Обычно приложения отменяют регистрацию большинства своих приемников, когда они становятся неактивными или невидимыми, т.е. при вызове метода `onStop()`. Для приложения, воспроизводящего аудио, обычно важно получать события нажатия кнопок тогда, когда оно невидимо, поэтому лучше регистрировать и снимать регистрацию приемника событий нажатия медиа кнопок, когда приложение получает или теряет аудио фокус, соответственно.

Управление аудио фокусом

Для избежания одновременного воспроизведения музыки разными приложениями Android использует понятие аудио фокус. Музыку может воспроизводить только то приложение, которое в данный момент времени владеет аудио фокусом, поэтому до начала воспроизведения музыки приложение должно запросить и получить аудио фокус. Более того оно должно знать, как отслеживать потерю аудио фокуса и соответствующим образом реагировать в случае потери.

Запрос на получение аудио фокуса выполняется вызовом метода `requestAudioFocus()`, который в случае успеха возвращает константу `AUDIOFOCUS_REQUEST_GRANTED`. При этом необходимо указывать какой поток используется и какого рода аудио фокус требуется: временный или

постоянный. Временный фокус предполагает воспроизведение короткого аудио произведения, например, инструкций по навигации. Постоянный фокус предполагает длительное воспроизведение аудио контента, например, прослушивание музыки.

Запрос фокуса необходимо выполнять непосредственно перед началом воспроизведения, например, когда пользователь нажимает кнопку play. Следующий код демонстрирует запрос на постоянный аудио фокус:

```
AudioManager am =
mContext.getSystemService(Context.AUDIO_SERVICE);

...
//запрос аудио фокуса для воспроизведения
int result = am.requestAudioFocus (afChangeListener,
//используя поток музыки
AudioManager.STREAM_MUSIC,
//запрос постоянного фокуса
AudioManager.AUDIOFOCUS_GAIN);
if (result == AudioManager.AUDIOFOCUS_REQUEST_GRANTED)
{
    am.registerMediaButtonEventReceiver (RemoteControlReceiver);
    // начало воспроизведения
}
```

Как только заканчивается воспроизведение, необходимо вызвать метод `abandonAudioFocus()`, который сообщает системе, что аудио фокус больше не требуется и отменяет регистрацию соответствующего `AudioManager.OnAudioFocusChangeListener`. В случае освобождения временного фокуса, любое приостановленное приложение может продолжить воспроизведение.

```
//освобождение аудио фокуса при завершении
воспроизведения am.abandonAudioFocus (afChangeListener);
```

При запросе временного аудио фокуса существует дополнительная опция:

возможность так называемого «ducking». В обычных условиях приложение, теряющее аудио фокус, заглушает воспроизведение, при запросе временного аудио фокуса с возможностью «ducking», работавшему до этого аудио приложению дается возможность лишь приглушить воспроизведение до возвращения фокуса к нему.

```
//запрос аудио фокуса для воспроизведения
int result = am.requestAudioFocus (afChangeListener,
//используя поток музыки
AudioManager.STREAM_MUSIC,
//запрос фокуса
AudioManager.AUDIOFOCUS_GAIN_TRANSIENT_MAY_DUCK);
if (result == AudioManager.AUDIOFOCUS_REQUEST_GRANTED)
{
// начало воспроизведения
}
```

«Ducking» особенно подходит для приложений, которые используют аудио поток периодически, например, для озвучивания рекомендаций водителю.

Приложение может потерять аудио фокус, как ему реагировать на потерю фокуса зависит от способа этой потери. Метод `onAudioFocusChange()` слушателя изменений аудио фокуса, зарегистрированного при запросе аудио фокуса, получает параметр, описывающий событие изменения фокуса. Потеря фокуса может быть постоянной или временной с возможностью «ducking» или без нее.

Временная потеря аудио фокуса заставляет приложение заглушить воспроизведение аудио потока, но сохраняет состояние воспроизведения. Необходимо следить за изменениями состояния аудио фокуса и быть готовым продолжить воспроизведение после возвращения фокуса.

Если потеря аудио фокуса является постоянной, приложение должно корректно завершиться, т.е. остановить воспроизведение, удалить слушателей событий от медиа кнопок и освободить аудио фокус. Для возобновления проигрывания аудио контента необходимо дождаться действия от пользователя, например, нажатия кнопки play.

Следующий фрагмент кода описывает приостановку воспроизведения в случае временной потери фокуса, возобновление воспроизведения при возвращении фокуса. В случае постоянной потери выполняется отмена регистрации приемника событий нажатия медиа кнопок и останавливается отслеживание событий изменения фокуса. `OnAudioFocusChangeListener afChangeListener = new`

```
OnAudioFocusChangeListener() {
    public void onAudioFocusChange(int focusChange) {
        if (focusChange == AUDIOFOCUS_LOSS_TRANSIENT) {
            //приостановить воспроизведение
        } else if (focusChange ==
AudioManager.AUDIOFOCUS_GAIN) {
            //продолжить воспроизведение
        } else if (focusChange ==
AudioManager.AUDIOFOCUS_LOSS) {
            am.unregisterMediaButtonEventReceiver(RemoteControlRe
ceiver); am.abandonAudioFocus(afChangeListener);
            // прекратить воспроизведение
        }
    }
};
```

В случае временной потери аудио фокуса с возможностью «ducking» вместо паузы лучше использовать снижение уровня громкости.

«Ducking» это и есть процесс снижения уровня громкости воспроизведения аудио потока, чтобы позволить другому приложению воспроизвести свой аудио контент. Следующий фрагмент кода снижает громкость при временной потере фокуса, после возвращения фокуса восстанавливает прежний уровень громкости

```
OnAudioFocusChangeListener afChangeListener = new
OnAudioFocusChangeListener() {
    public void onAudioFocusChange(int focusChange) {
```

```

        if (focusChange == AUDIOFOCUS_LOSS_TRANSIENT_CAN_DUCK)
    {
        // снижение громкости
    } else if (focusChange ==
AudioManager.AUDIOFOCUS_GAIN) {
        // восстановление уровня громкости
    }
}
};

```

Взаимодействие с оборудованием для воспроизведения аудио

Пользователи имеют возможность выбора при прослушивании аудио на Android устройствах. Большинство устройств имеют встроенный динамик, выход для наушников, поддержку A2DP аудио и т.д.

Поведение приложения может зависеть от устройства, на которое направлен вывод аудио контента. Можно запросить AudioManager определить какое устройство используется для вывода аудио, динамик мобильного устройства, проводные наушники или Bluetooth устройство.

```

if (isBluetoothA2dpOn()) {
    //адаптировать вывод для Bluetooth
} else if (isSpeakerphoneOn()) {
    //адаптировать вывод для динамика устройства
} else if (isWiredHeadsetOn()) {
    // адаптировать вывод для наушников
} else {
    //никто его не слышит, аудио все еще играет?
}

```

Когда наушники отсоединяются или Bluetooth устройство становится недоступным, аудио поток автоматически перенаправляется на встроенный динамик. Если громкость воспроизведения высокая, то неожиданный шум из динамиков может неприятно удивить, особенно окружающих.

К счастью, система рассылает сообщение

ACTION_AUDIO_BECOMING_NOISY, когда это происходит. Хорошей практикой является привычка регистрировать приемник широковещательных сообщений для получения этого сообщения, каждый раз как приложение воспроизводит аудио. В случае с воспроизведением музыки пользователи обычно ожидают приостановки воспроизведения, в случае с играми ожидается существенное понижение уровня громкости.

```
private class NoisyAudioStreamReceiver extends
BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent)
    {
        if
(AudioManager.ACTION_AUDIO_BECOMING_NOISY.equals(intent.g
etAction())) {
            // приостановить воспроизведение
        }
    }

    private IntentFilter intentFilter =
new
IntentFilter(AudioManager.ACTION_AUDIO_BECOMING_NOISY);

    private void startPlayback() {
registerReceiver(myNoisyAudioStreamReceiver,
intentFilter);
    }

    private void stopPlayback() {
unregisterReceiver(myNoisyAudioStreamReceiver);
    }
```

Содержание отчета и его форма

Отчет к лабораторной работе оформляется в виде текстового документа по форме простого реферата и должен включать:

1. Название лабораторной работы.
2. Цель лабораторной работы.
3. Краткие выводы по результатам выполнения лабораторной работы согласно пунктам раздела «Методики выполнения работы».
4. Ответы на Вопросы к лабораторной работе.
5. Формулировку задания для самостоятельной работы и результат его выполнения.

Задания для самостоятельной работы

1. Для тренировки постройте приложение для записи и воспроизведения медиаконтента, добавьте возможности управления громкостью и управления аудио фокусом.
2. Постройте приложение с элементами распознавания речи, пример создания такого приложения:
http://www.pandacoder.com/android_speech_recognition/.

Вопросы к лабораторной работе

1. Технология воспроизведение аудио и видео.
2. Характеристика экземпляра класса MediaPlayer.
3. Характеристика метода create().
4. Технология записи аудиои видео-данных.
5. Технология добавления мультимедийных файлов в MediaStore.
6. Технология работы с несжатым звуком.
7. Технология распознавания речи.

Лабораторная работа 6

Установка и настройка среды программирования ADT Bundle

Цель работы: установка и настройка среды программирования ADT Bundle.

Теоретическое обоснование

Лабораторная работа посвящена описанию работы в среде ADT Bundle (Android IDE). В работе рассказывается о программном обеспечении, которое необходимо скачать с официальных сайтов, установить и настроить, создании и запуске простейшего приложения на эмуляторе и мобильном устройстве. В связи с тем, что отладка на устройствах сопряжена с рядом трудностей, приводится подробная инструкция по настройке устройств и операционной системы Windows (версия не ниже 7).

Задания к лабораторной работе

Задание 1. Установка среды

Большинство приложений для OS Android написано на Java. Одной из самых популярных сред разработки является Eclipse (для неё также необходим JDK) с установленным плагином ADT и Android SDK. Раньше приходилось ставить все компоненты отдельно. Сейчас появилась версия среды Eclipse с уже настроенными дополнениями ADT Bundle. Здесь есть минимум инструментов, необходимый для разработки приложений. С этой версией мы и будем работать. Однако в ней есть далеко не всё, поэтому, если при разработке какого-либо проекта вам потребуются инструменты, не входящие в ADT Bundle, вы можете скачать их с сайта разработчиков и дополнить свою среду (рис. 6.1).

Скачать среду можно с сайта для разработчиков Android (<http://developer.android.com/sdk/index.html>).

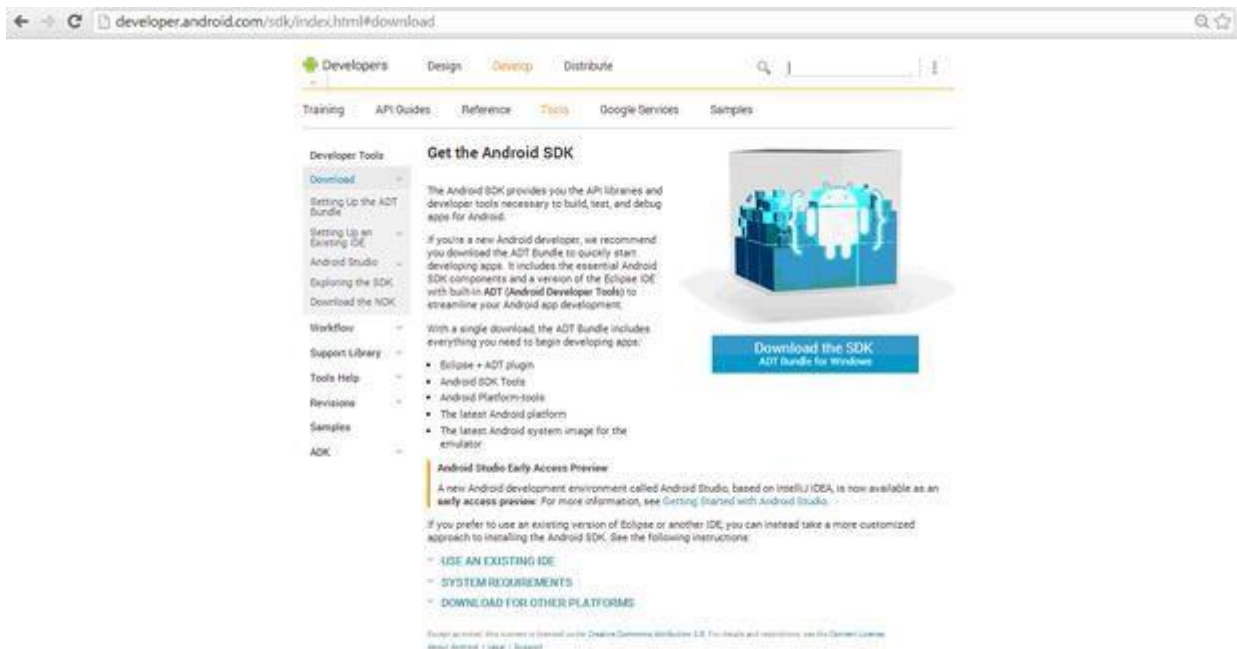


Рисунок 6.1 – Сайт разработчика

Для того, чтобы скачать среду (рис. 6.2), необходимо принять условия лицензионного соглашения и выбрать вашу версию Windows (32-bit или 64-bit).

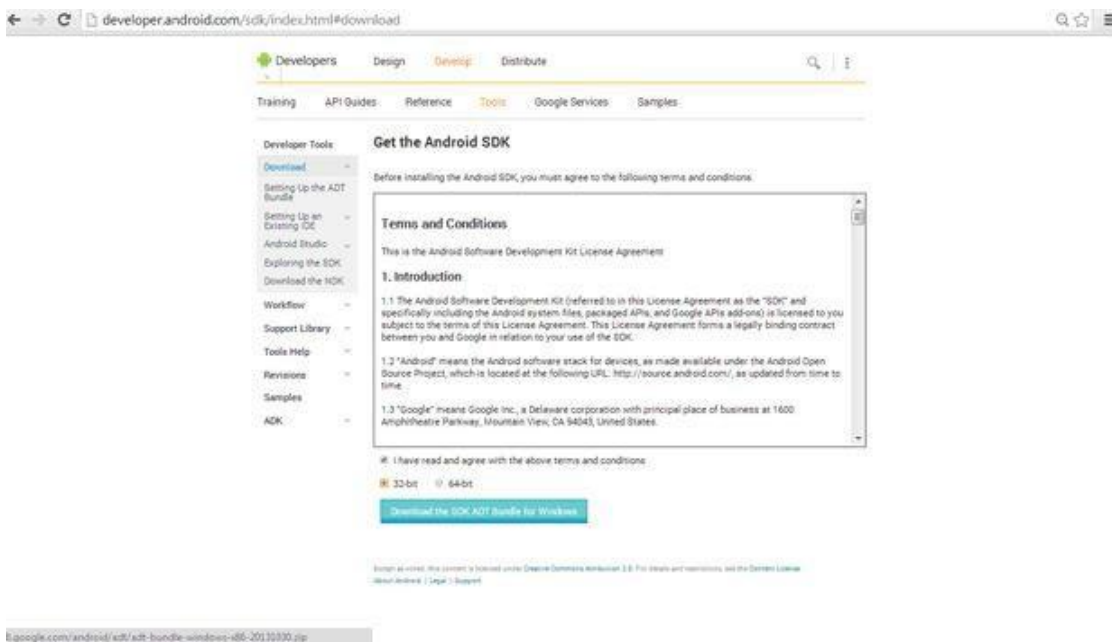


Рисунок 6.2 – Скачивание среды

После скачивания распакуйте архив в ту папку, где собираетесь работать (среда не требует специальной установки). После распаковки зайдите в папку и запустите Eclipse. Здесь возможна небольшая проблема: если у вас не

установлен JDK, среда не запустится и потребует указать путь к папке с JDK или установить его.

Скачать JDK можно с сайта Oracle (рис. 6.3).



Рисунок 6.3 – Сайт компании Oracle

Чтобы скачать JDK нужно сначала принять условия лицензионного соглашения, а затем выбрать нужную версию (рис. 6.4).

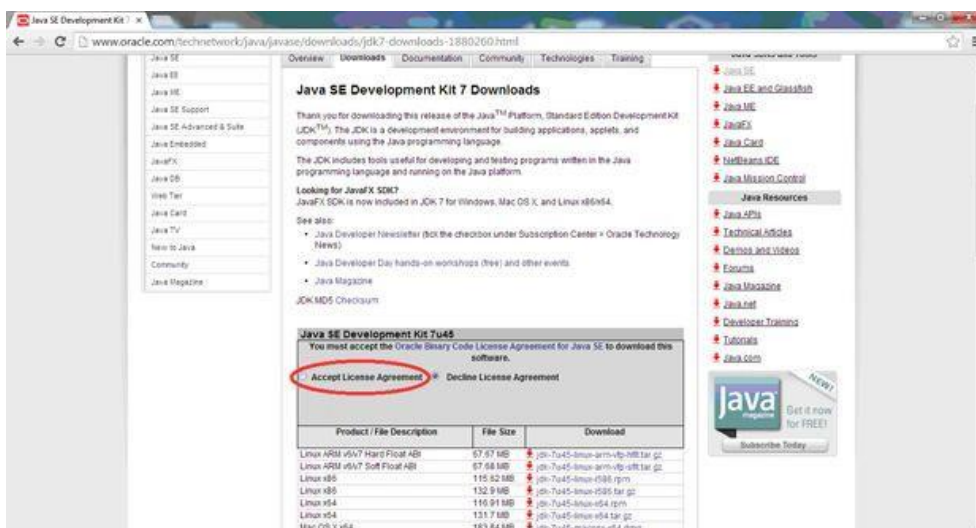


Рисунок 6.4 – Скачивание JDK После скачивания запустите setup-файл и установите JDK



Рисунок 6.5 – Установка JDK

После установки JDK среда должна запускаться.

Далее вам необходимо выбрать (или создать новое) рабочее пространство, т.е. место, где будут находиться ваши проекты. Если поставить галочку, то это рабочее пространство будет выбираться по умолчанию, а противном случае это окно будет появляться при каждом запуске Eclipse (рис. 6.6).

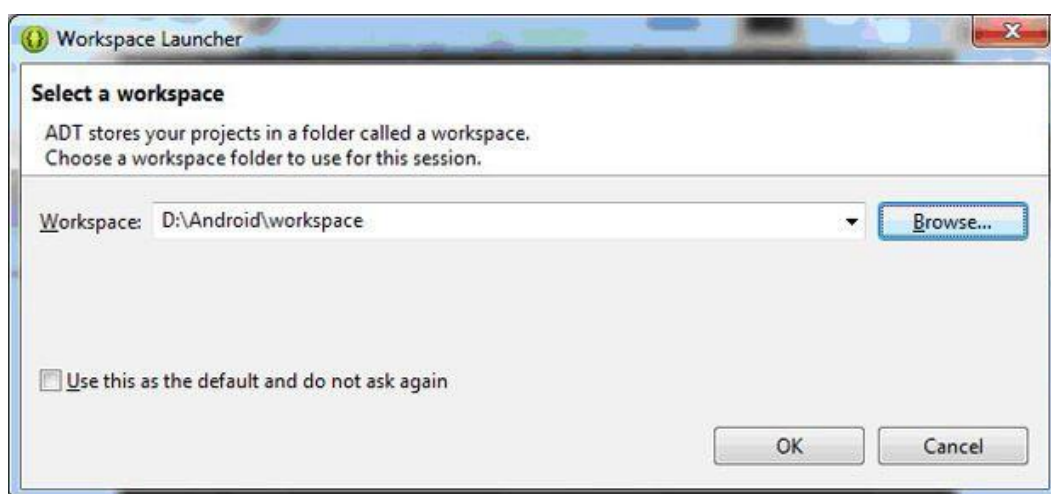


Рисунок 6.6 – Выбор рабочего пространства

Затем появляется окно, в котором разработчики предлагают отправлять статистику для дальнейшего улучшения SDK. Вы можете согласиться или отказаться.

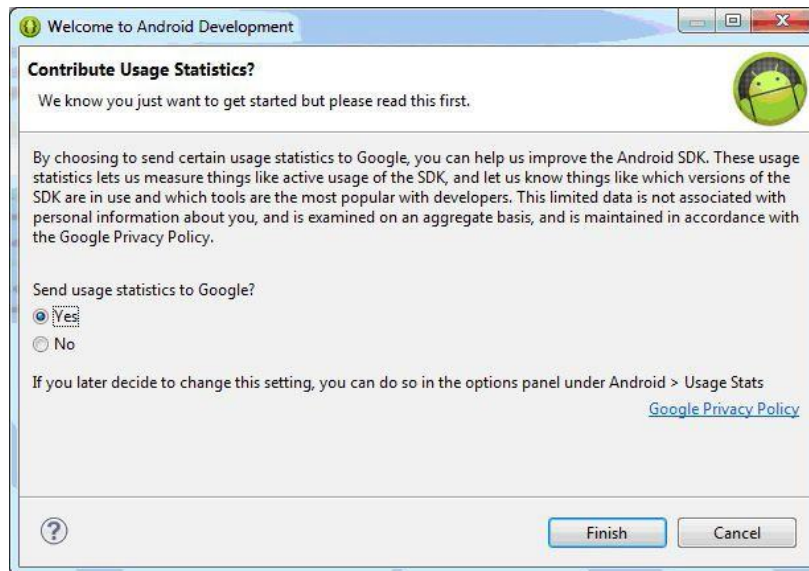


Рисунок 6.7 – Отправка статистики

Обратите внимание на значок Android SDK Manager, находящийся на панели инструментов (его также можно найти в меню Window). С его помощью вы сможете добавлять в свою среду новые инструменты.

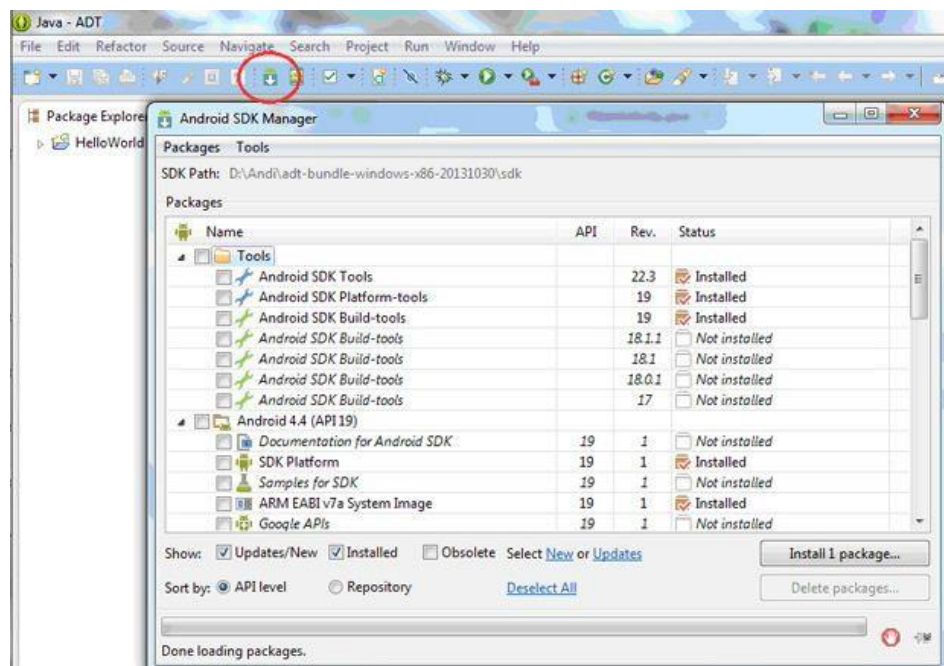


Рисунок 6.8 – Android SDK Manager

Задание 2. Создание проекта

Итак, наконец, мы подошли к самому главному созданию проекта (рис. 6.9).

Чтобы создать проект, зайдите в меню **File->New->Android Application Project**.

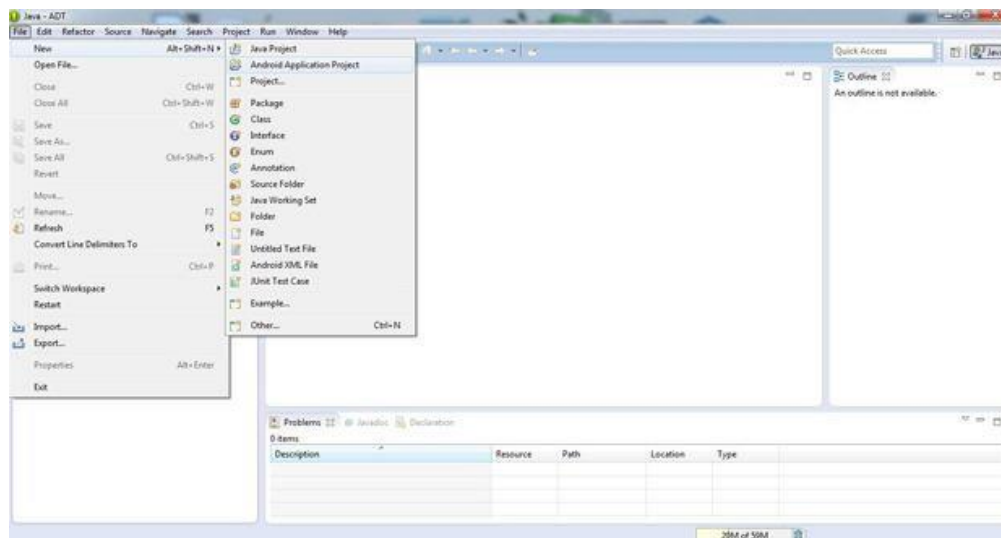


Рисунок 6.9 – Создание проекта

В появившемся окне обязательно нужно прописать имя приложения, имя проекта, а также имя пакета (package). Лучше не оставлять его именем example, т.к. пакет с таким именем нельзя разместить в Google Play. Конечно, учебные приложения туда не загружают, однако, следует иметь это в виду на будущее.

Minimum Required SDK минимальная версия Android, которую будет поддерживать приложение. Чаще всего по умолчанию указывается версия 2.2, чтобы поддерживать как можно больше устройств. Если определенная функция вашего приложения работает только на более новых версиях Android, и это не является критическим для основного набора функций приложения, вы можете включить ее в качестве опции на версиях, которые поддерживают его.

Target SDK версия Android, под которую будет написано ваше приложение; определяет максимальную версию Android, на которой вы тестировали приложение. Это нужно для режимов совместимости.

Compile With определяет, возможности какой версии Android будет использовать приложение.

Оставьте пока установки, заданные по умолчанию в качестве значений для этого проекта (рис. 6.10, 6.11).

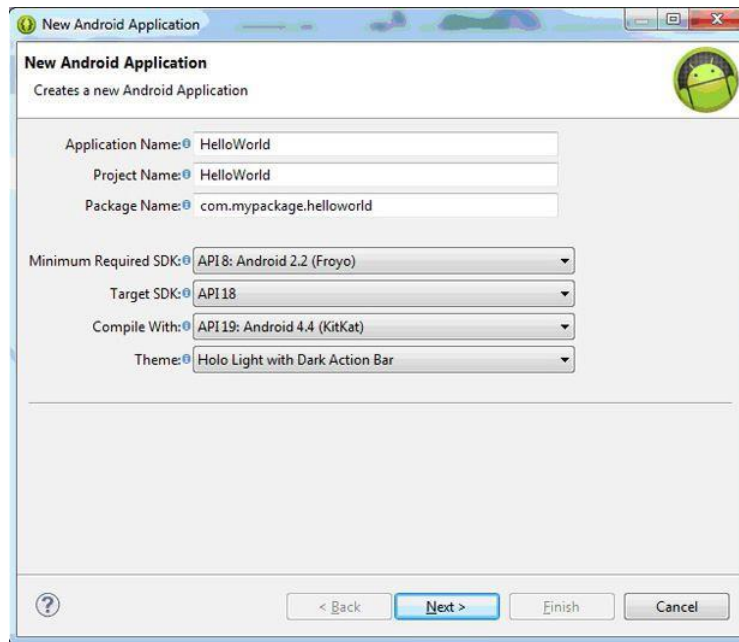


Рисунок 6.10 – Наименование проекта Следующее окно можно пропустить без изменений

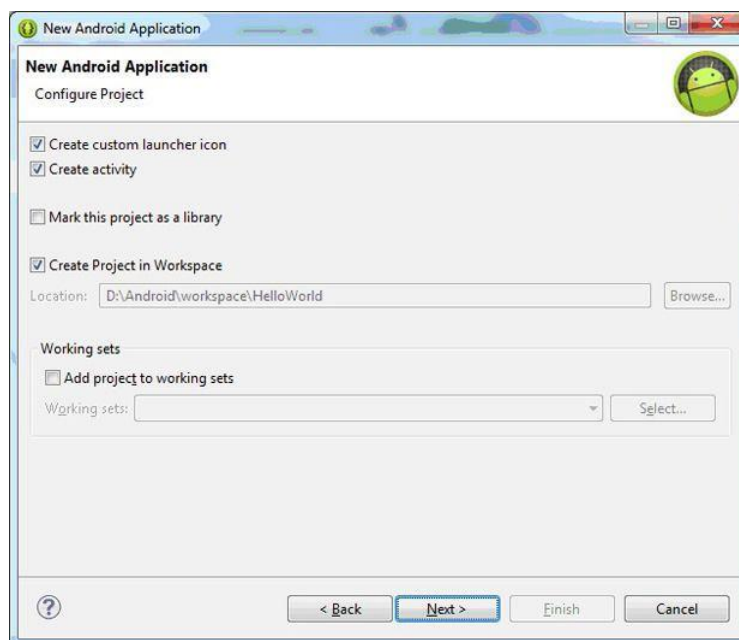


Рисунок 6.11 – Конфигурация проекта

Create custom launcher icon создать значок приложения.

Create activity создать Activity (активность, деятельность).

Mark this project as library создать проект, как библиотеку. Сейчас в этом нет необходимости, наше приложение в других проектах использоваться не будет.

Create Project in Workspace создать проект в папке Workspace. В этой папке будут храниться все наши проекты.

Следующий этап создание иконки. Можете оставить стандартную или создать свою собственную. В нашем примере изменена цветовая гамма, форма, а также выбрана фигурка из клипарта (рис. 6.12).

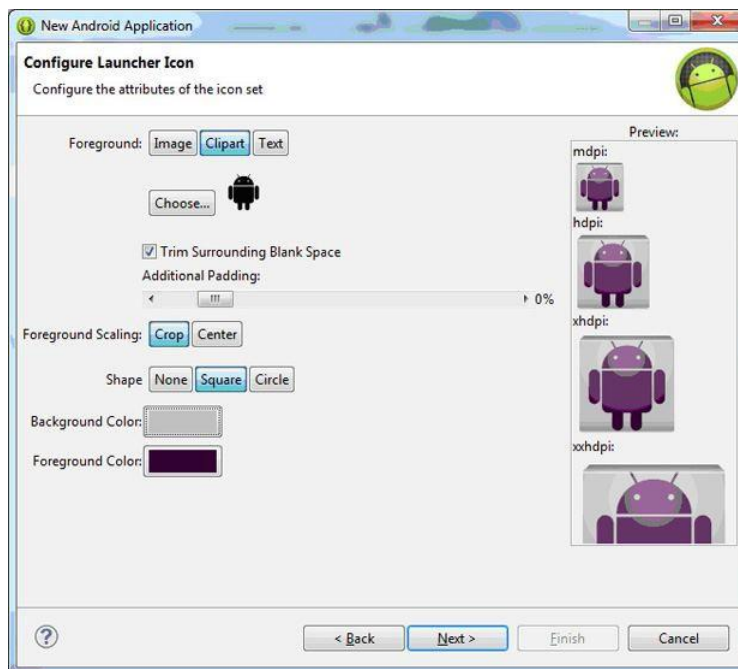


Рисунок 6.12 – Создание иконки приложения

Большинство приложений на Android имеют свой экран (форму, окно), которое называется активностью или деятельностью (Activity).

Следующее два окна создают пустую активность. В первом ничего пока менять не нужно.

Во втором вы можете переименовать свою активность (в приложении их может быть несколько) (рис. 6.13).

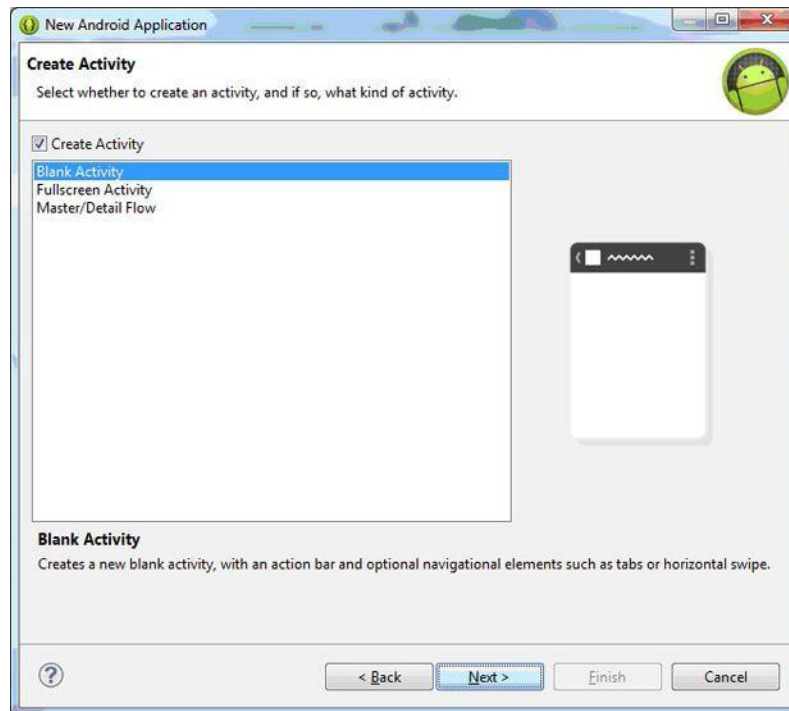


Рисунок 6.13 – Создание активности

Blank Activity шаблон, предназначенный для мобильных телефонов.

Fullscreen Activity шаблон, позволяющий растянуть приложение на весь экран (без навигационной панели и статус-бара).

Master/Detail Flow шаблон, предназначенный для планшетных компьютеров (рис. 6.14).

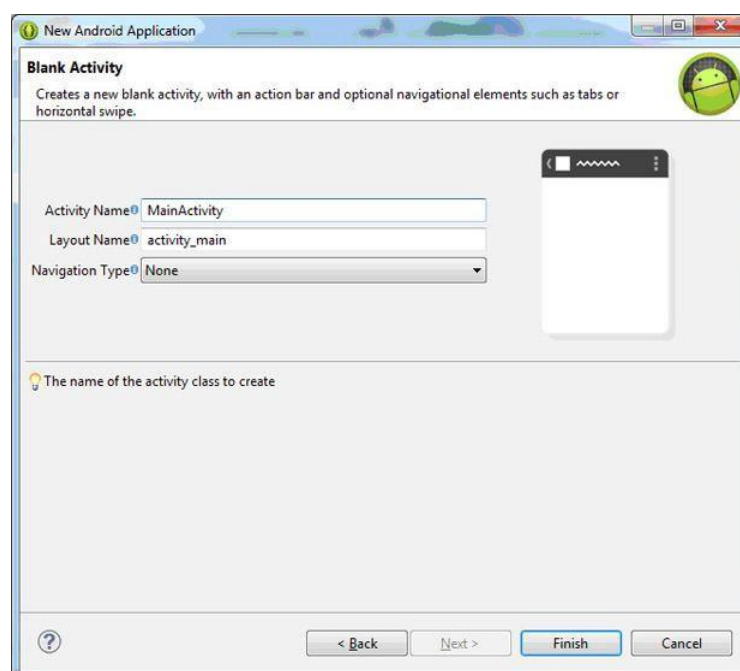


Рисунок 6.14 – Переименование активности

Итак, вы создали свой первый проект. Конечно, это всего лишь встроенное в среду приложение для проверки корректной установки инструментария, однако множество приложений создаются именно из него.

Посмотрим на его структуру. Она показана в области слева.

В первую очередь нас интересует файл активности. Он находится в папке **src** в вашем пакете. Он имеет расширение **.java** (рис. 6.15).

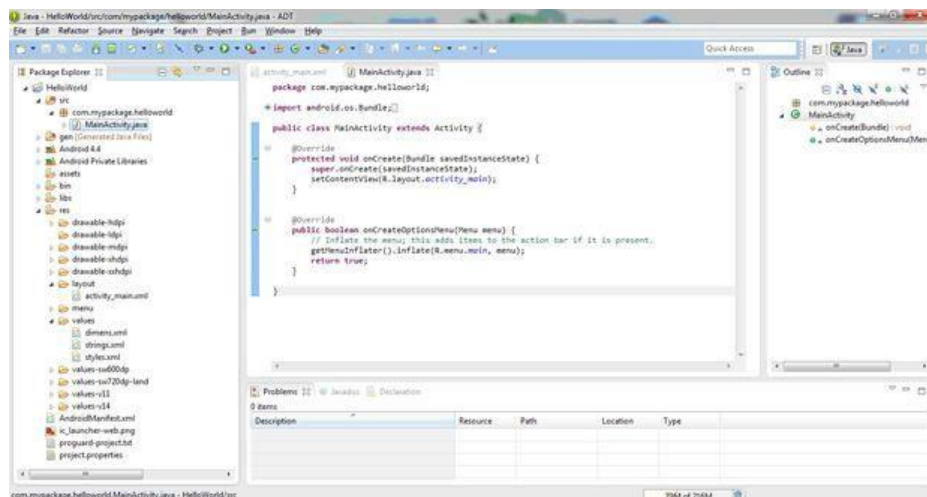


Рисунок 6.15 – Активность

В папке **res** в подпапке **layout** находится xml-файл, который является оболочкой нашей активности. Именно этот файл будет виден на экране устройства.

С xml-файлами можно работать как в режиме графического редактора, так и непосредственно редактировать код (рис. 6.16, 6.17).

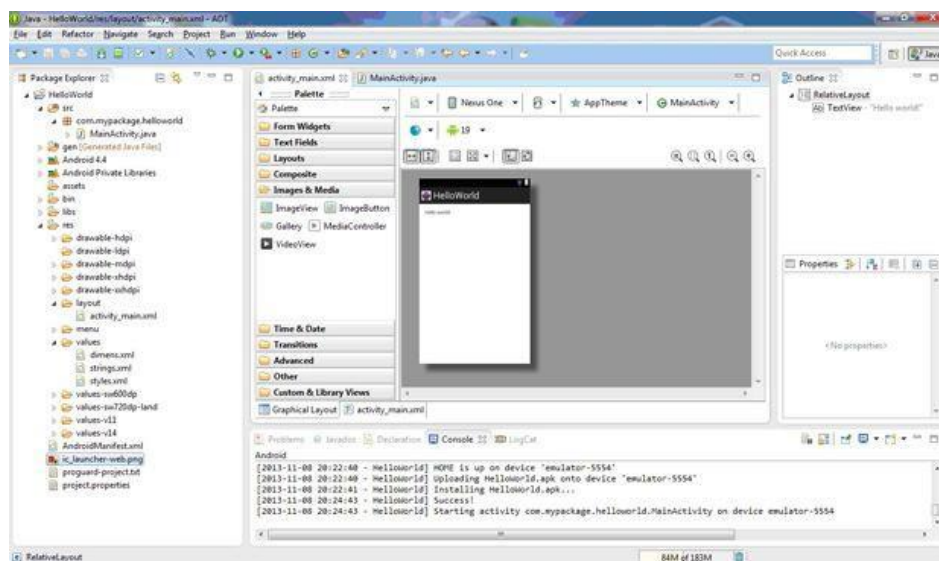


Рисунок 6.16 – Xml-файл. Графический редактор

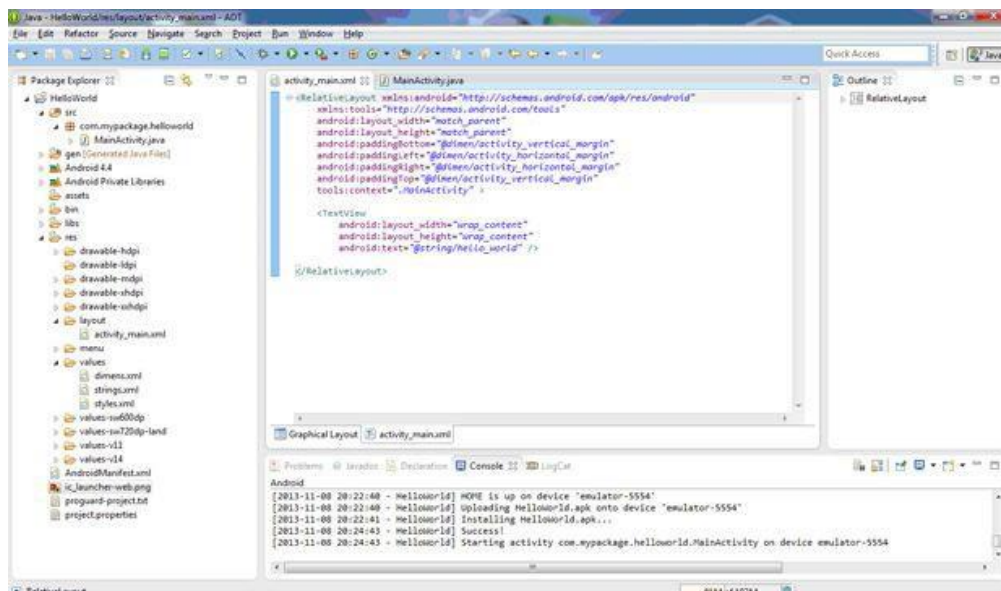


Рисунок 6.17 – Xml-файл

Задание 3. Запуск проекта на эмуляторе устройства

В первую очередь нужно создать эмулятор устройства. Это можно сделать, нажав на кнопку на панели инструментов, изображающую смартфон. Если кнопки нет на панели, ее можно найти в меню **Window** (рис. 6.18).

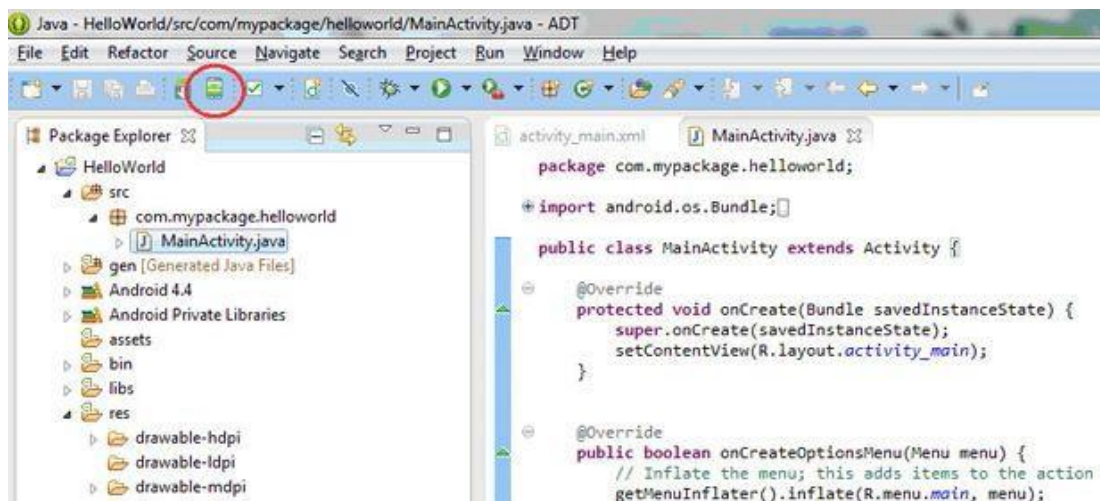


Рисунок 6.18 – Запуск проекта

Откроется Android Virtual Device Manager. Пока в нем нет ни одного виртуального устройства (рис. 6.19).

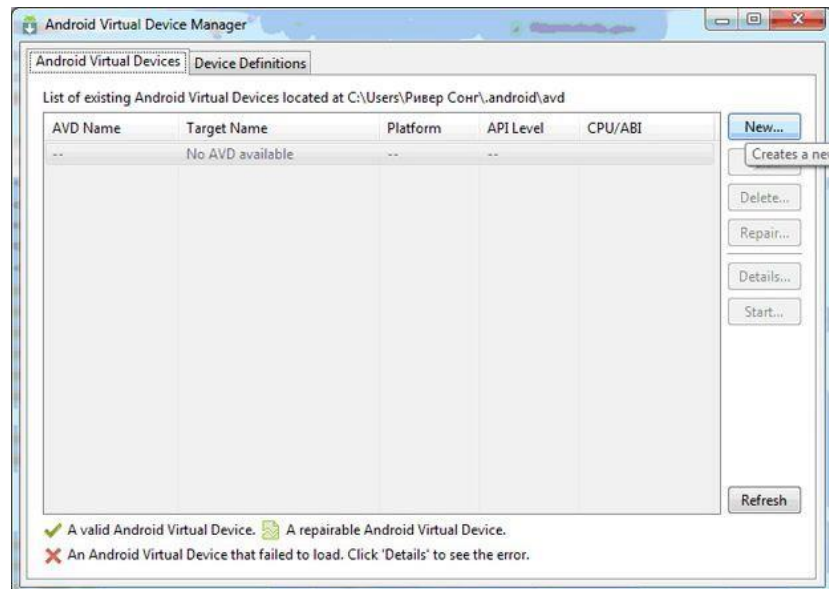


Рисунок 6.19 – Android Virtual Device Manager

Чтобы создать виртуальное устройство, нажмите кнопку **New**. Появится окно создания. Вам нужно назвать устройство и выбрать обязательные характеристики: **Device** модель вашего устройства, и **Target** версия Android. Также можно изменять дополнительные параметры: размер sd-карты, встроенной памяти и т.п (рис. 6.20 - 6.25).

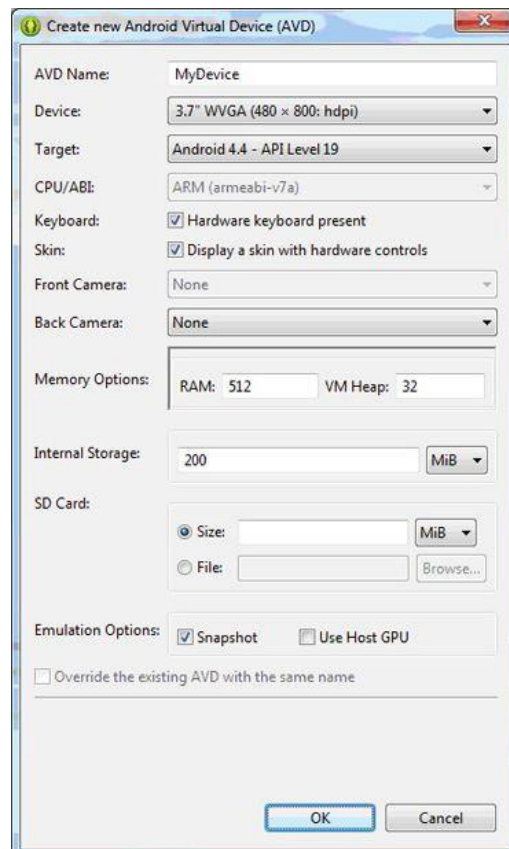


Рисунок 6.20 – Создание AVD

Теперь можно запускать приложение. Для этого нужно нажать на кнопку **Run** (белый треугольник в зеленом кружочке) на панели инструментов. Проблемы с запуском можно отследить в консоли.

Если приложение не запускается, попробуйте нажать на черный треугольник справа от кнопки **Run**, выбрать **Run Configurations**, затем во вкладке **Target** выбрать созданное устройство и запустить проект снова (рис. 6.21).

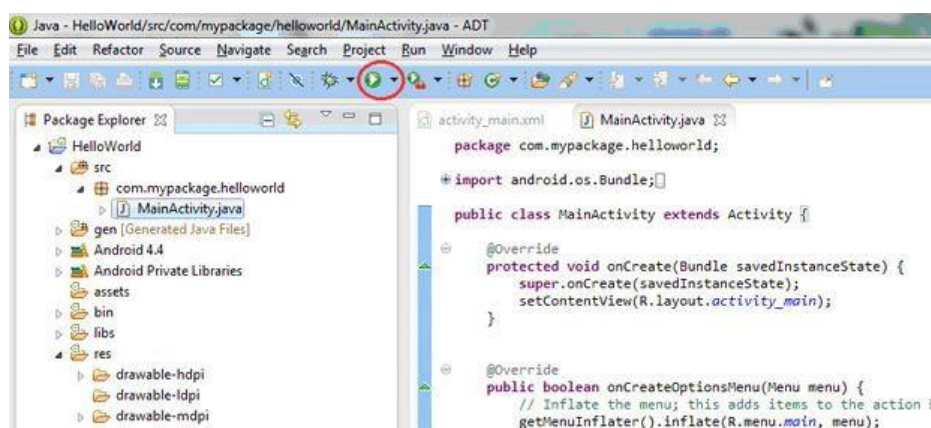


Рисунок 6.21 – Запуск приложения

Если все сделано правильно, должен запускаться эмулятор (рис. 6.22). Время запуска зависит от размера оперативной памяти на вашем компьютере. В дальнейшем эмулятор можно не закрывать, приложения будут запускаться в работающем (рис. 6.23, 6.24).



Рисунок 6.22 – Запуск эмулятора



Рисунок 6.23 – Запущенный эмулятор

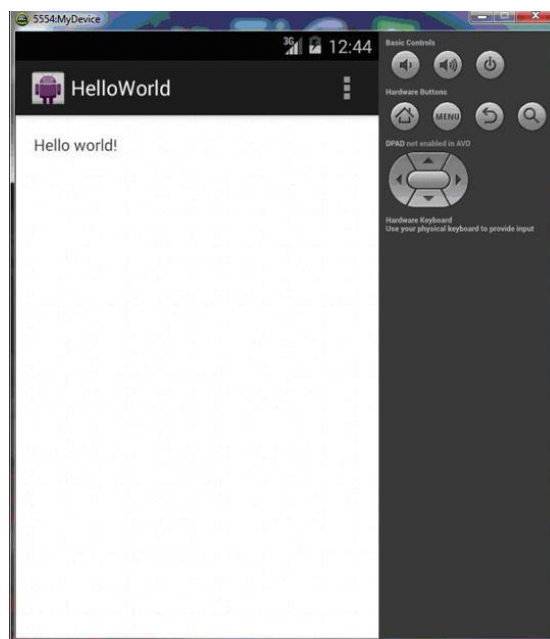


Рисунок 6. 24 – Приложение Hello, world!

Если ваше приложение сразу не запустилось, его можно найти в меню приложений устройства (рис. 6.25).

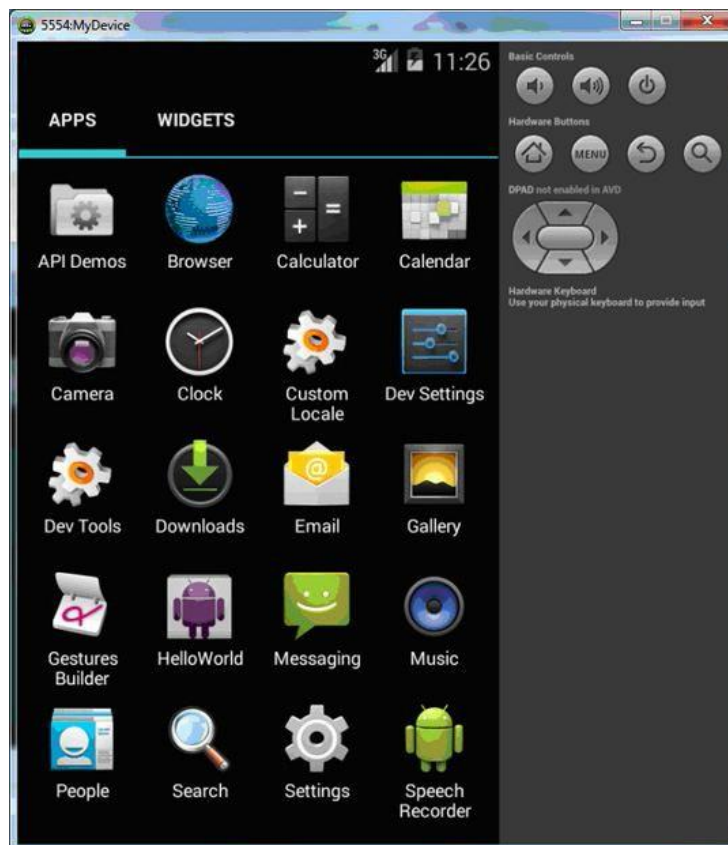


Рисунок 6.25 – Меню приложений

Задание 4. Запуск проекта на устройстве

Прежде чем запустить проект на реальном устройстве, необходимо:

- Настроить устройство
- Настроить компьютер
- Настроить среду

Задание 5. Настройка устройства

Настройка устройства состоит в следующем:

1. Включить режим отладки по USB.
2. Для запуска файлов с расширением *.apk, полученных не из магазина приложений Google Play, необходимо разрешить установку приложений из альтернативных источников (рис. 6.26).

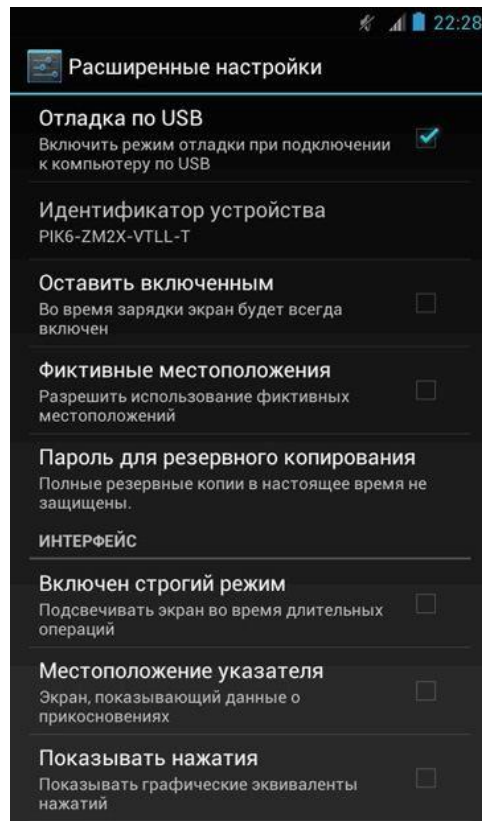


Рисунок 6.26 – Настройка устройства

Задание 6. Настройка компьютера

Чтобы настроить компьютер необходимо выполнить следующие действия:

1. Найти в Диспетчер устройств (**Пуск->Панель управления->Система и безопасность->Система->Диспетчер устройств**) и найти ваше устройство (рис. 6.27).

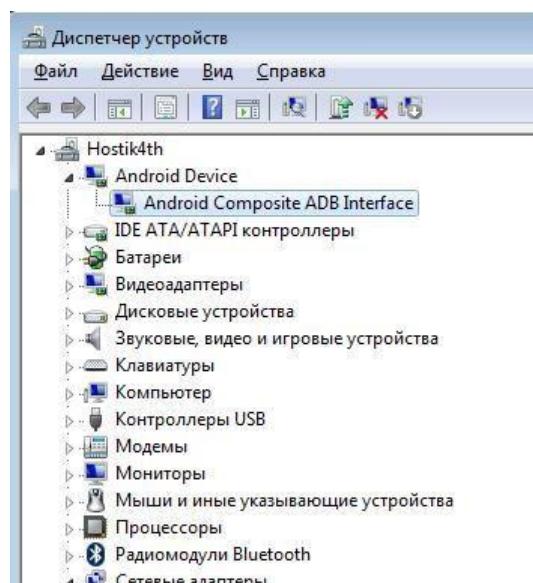


Рисунок 6.27 – Диспетчер устройств

2. Щелкнуть по нему правой кнопкой мыши и вызвать меню **Свойства**.

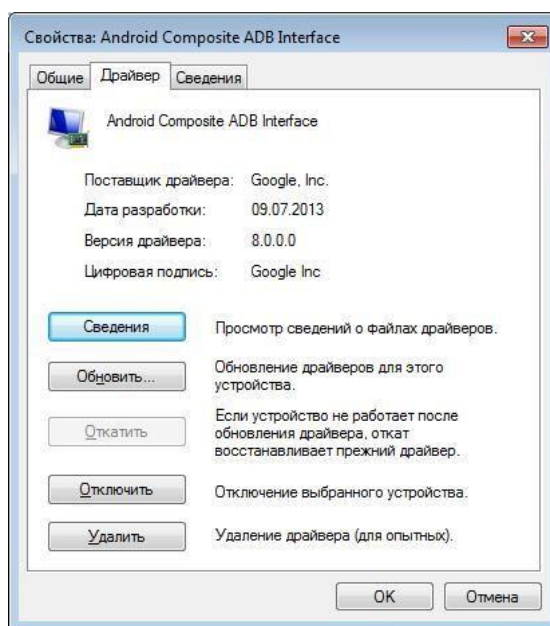


Рисунок 6.28 – Меню Свойства

3. Во вкладке **Драйвер** нажать на кнопку **Обновить** драйвер (рис. 6.29).

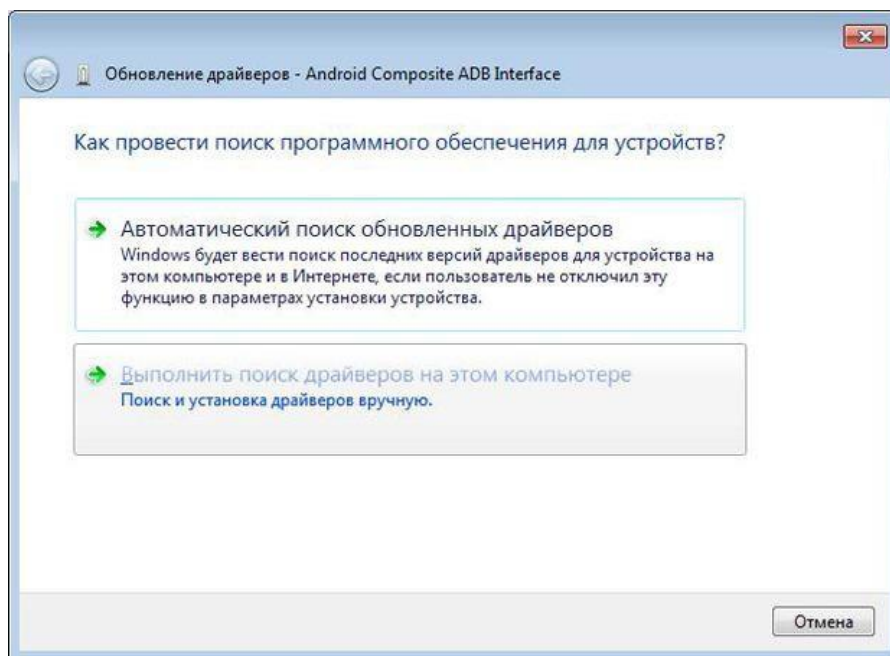


Рисунок 6.29 – Обновление драйвера

4. В появившемся окне выбрать **Выполнить поиск драйвера на этом компьютере** (рис. 6.30).

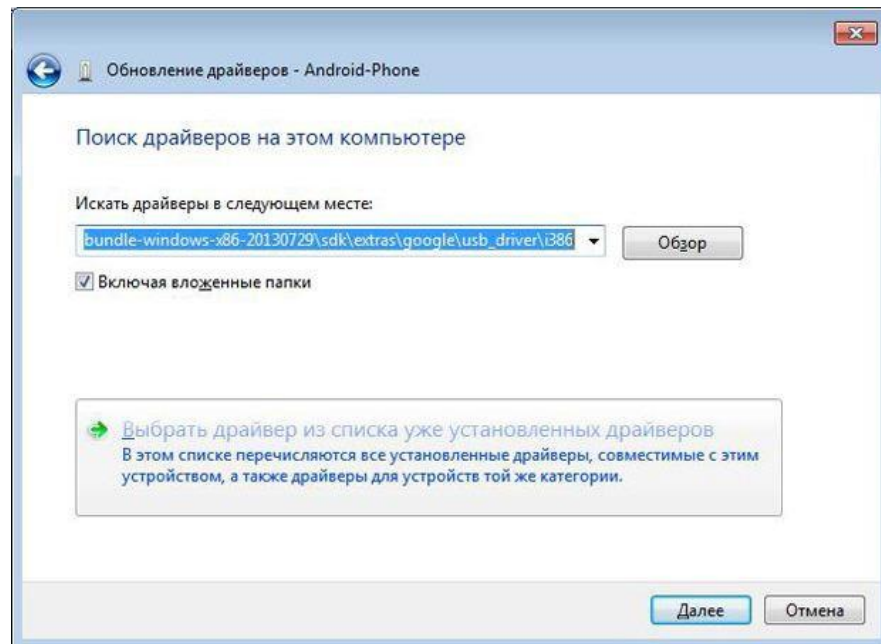


Рисунок 6.30 – Поиск драйверов на компьютере

5. В следующем окне нужно прописать путь, откуда будет установлен драйвер.

В нашем случае путь следующий: **adt-bundle-windows-x8620130729\sdk\extras\google\usb_driver\i386**.

Или можно отыскать драйвер следующим образом:

1) В следующем окне нажмите **Установить с диска** (рис. 6.31)

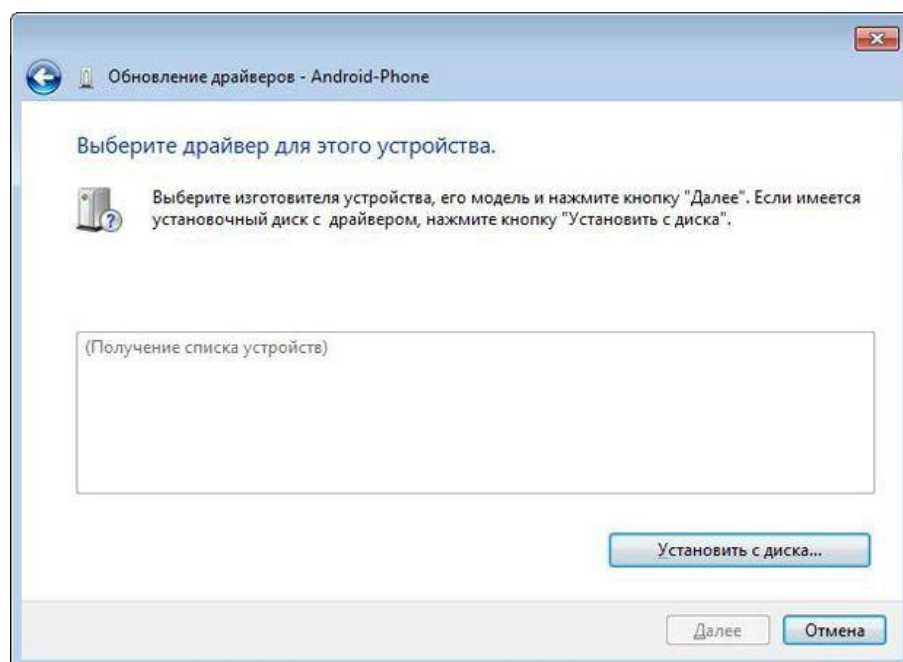


Рисунок 6.31 – Выбор драйвера устройства

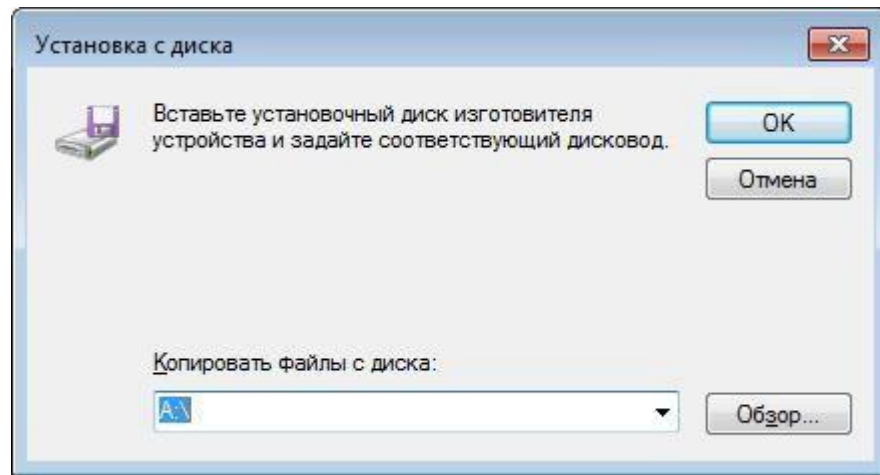


Рисунок 6.32 – Поиск драйвера

- 2) Далее выберите файл android_winusb.inf (рис. 6.33)

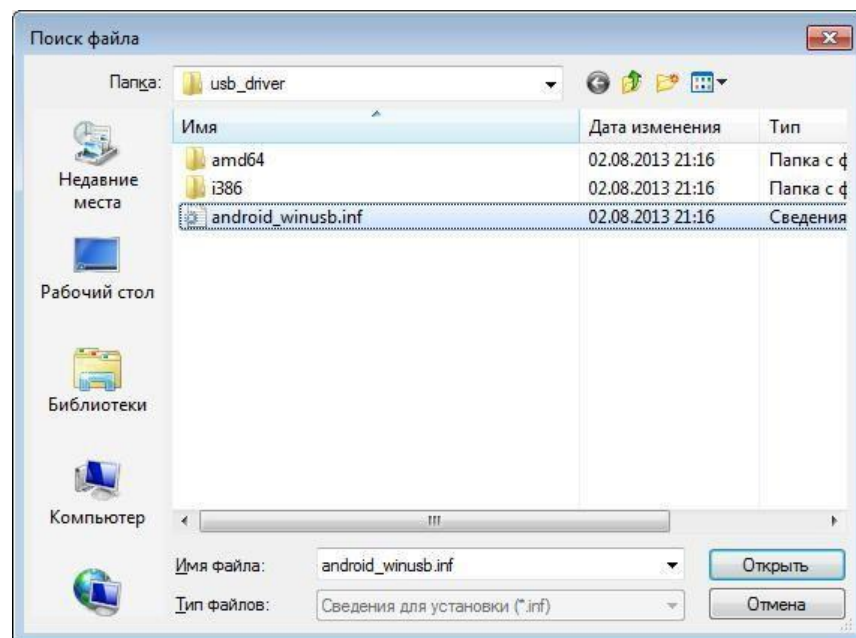


Рисунок 6.33 – Сведения для установки

- 3) Нажмите **Далее** в этом и следующем окнах.
- 4) Из предложенного списка выберите **Android ADB Interface** и нажмите **Далее** и **Да** (рис. 6.34).

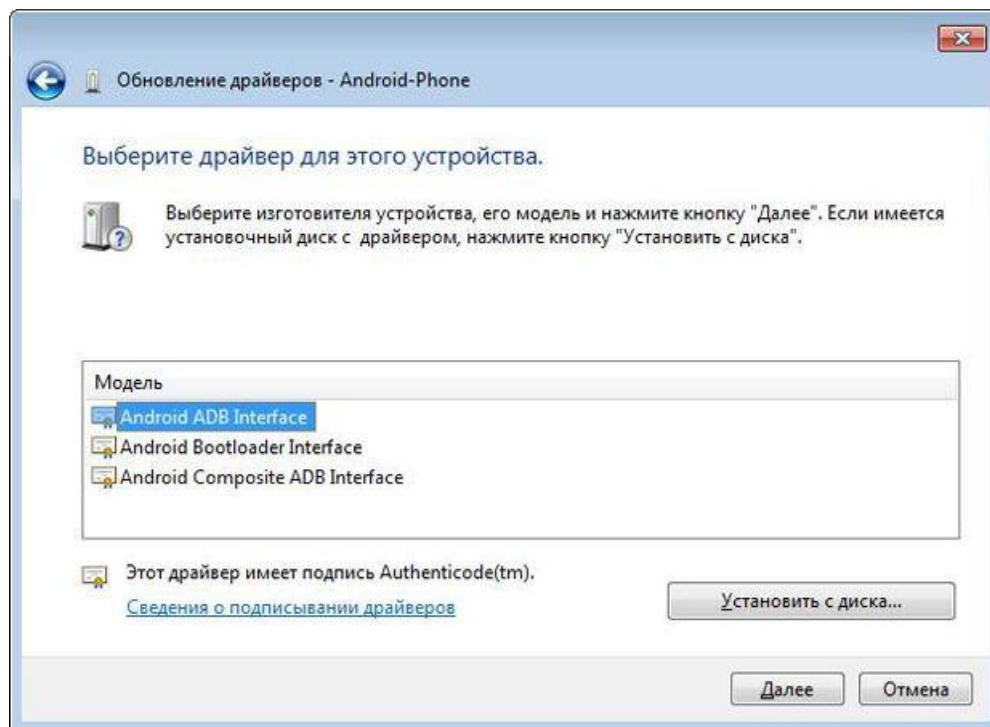


Рисунок 6.34 – Завершение установки драйвера

Задание 7. Настройка среды

1. В ADT зайдите в меню **Run\Debug Configurations** и перейдите на вкладку **Target**.

Поставьте флажок напротив **Always prompt to pick device** (рис. 6.35).

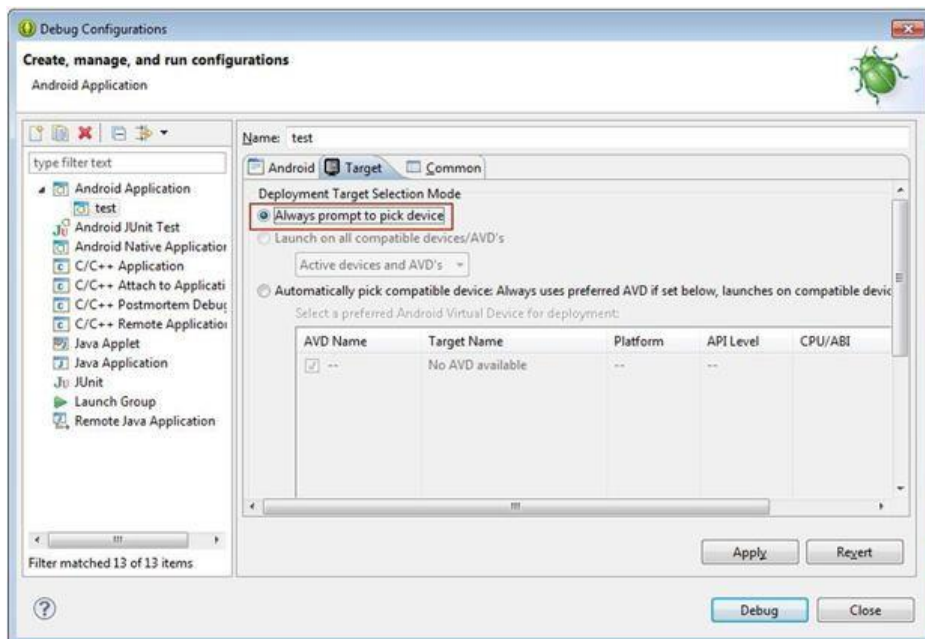


Рисунок 6.35 – Настройка среды

2. В открывшемся окне выберите подключенное устройство, поставив

флажок напротив **Choose a running Android Device** (рис. 6.36).

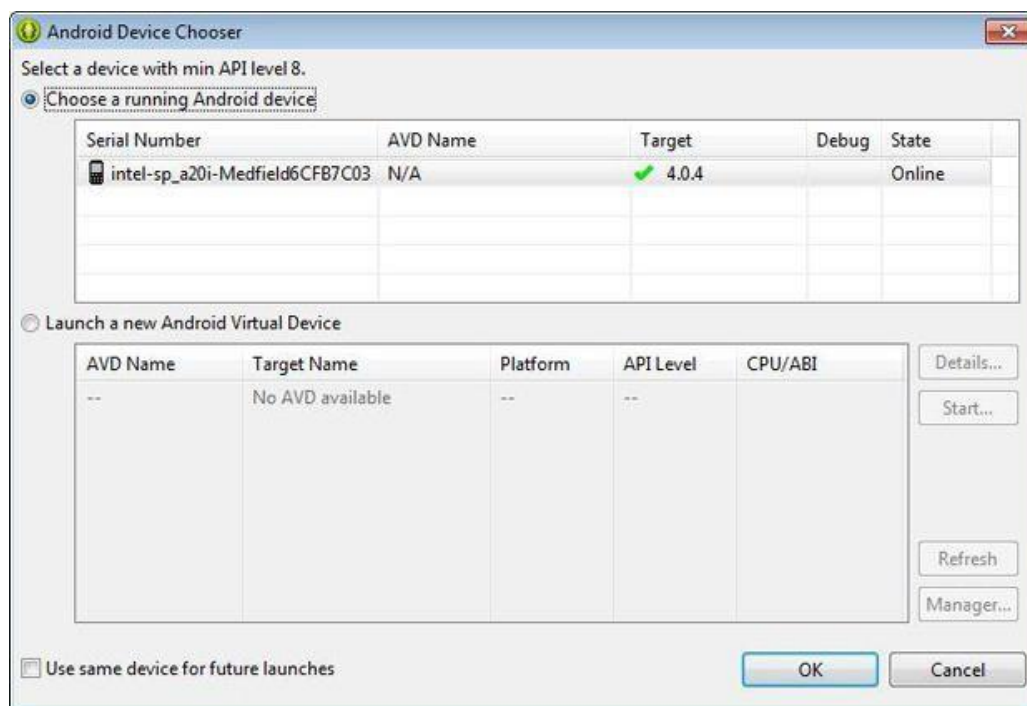


Рисунок 6.36 – Выбор устройства

3. Теперь можно запустить приложение.

Содержание отчета и его форма

Отчет к лабораторной работе оформляется в виде текстового документа по форме простого реферата и должен включать:

1. Название лабораторной работы.
2. Цель лабораторной работы.
3. Краткие выводы по результатам выполнения лабораторной работы согласно пунктам раздела «Методики выполнения работы».
4. Ответы на Вопросы к лабораторной работе.
5. Формулировку задания для самостоятельной работы и результат его выполнения.

Задания для самостоятельной работы

1. Создайте и запустите на вашем устройстве проект Hello, world!
2. Создайте и запустите на вашем устройстве аналогичный проект, поменяв размер, цвет и рисунок иконки и изменив текст приветствия с Hello, world! на Hello, Android!

Вопросы к лабораторной работе

1. Характеристика среды Android IDE.
2. Порядок установки и настройки среды.
3. Технология создания проекта в Android IDE.
4. Технология запуска проекта на эмуляторе устройства.

Лабораторная работа 7

Основные этапы разработки приложения с использованием Android IDE

Цель работы: разработка простого приложения, помогающего понять структуру приложения, освоить основные операторы, привыкнуть к среде разработки.

Теоретическое обоснование

Для достижения поставленной цели в лабораторной работе создадим приложение в среде разработки Android IDE (Eclipse и ADT), подробно рассмотрим структуру полученного проекта и разберем назначение основных его элементов.

Чтобы дальнейшие действия приобрели некоторый смысл, сформулируем задачу, которую будет решать наше приложение, назовем его «Угадай число». Суть приложения в том, что программа случайным образом «загадывает» число от 0 до 100, а пользователь должен угадать это число. При каждом вводе числа, программа сообщает пользователю результат: введенное число больше загаданного, меньше или же «Ура, победа!» число угадано. Разрабатываемое приложение выполняет свои функции только когда видимо на экране, когда оно не видимо его работа приостанавливается, т. е. имеем дело с приложением переднего плана. Для выполнения всей работы достаточно определить одну активность в приложении, фоновые процессы не предусмотрены.

Далее в работе рассмотрим простейшие элементы интерфейса пользователя и добавим их в приложение, а также рассмотрим вопросы, связанные непосредственно с программированием: научимся обрабатывать события, возникающие при взаимодействии приложения с пользователем;

реализуем логику проверки числа на совпадение с загаданным.

Задания к лабораторной работе

Задание 1. Создание приложения и изучение его структуры

Создайте новый проект в среде Android IDE (Eclipse с ADT). Процесс создания нового проекта и описание основных настроек подробно рассмотрен в лабораторной работе к первой лекции.

В процессе создания проекта, мы назвали его ProjectN, среда разработки подготавливает необходимые папки и файлы. Полный иерархический список обязательных элементов проекта можно увидеть на вкладке Package Explorer (аналогичную информацию предоставляет вкладка Project Explorer), иерархия полученных папок и файлов для нашего проекта изображена на рисунке 7.1.



Рисунок 7.1 – Структура проекта ProjectN

В настоящее время нас будет интересовать назначение нескольких файлов и папок. Рассмотрим папки:

5) папка **src** содержит файлы с исходным кодом на языке Java. Именно в этой папке размещаются все классы, создаваемые в процессе

разработки приложения. Сейчас в этой папке в пакете `com.example.projectn` размещается единственный класс `FullscreenActivity.java`. Этот класс определяет главную и единственную активность в этом приложении.

Комментарий 1: Имя пакету присваивается в процессе создания приложения в поле **Package Name**, использовать `com.example` не рекомендуется, т. к. пакет с таким именем нельзя загрузить в Google Play. Часто рекомендуют использовать в качестве имени пакета название сайта программиста, записанное в обратном порядке, можно просто использовать свои имя и фамилию. Последнее слово в имени пакета формируется автоматически и совпадает с именем проекта.

Комментарий 2: Имя файлу присваивается в процессе создания приложения на этапе настройки активности. Имя определяется в поле **Activity Name**.

Комментарий 3: Package Explorer отображает структуру папок, которая создается в каталоге, выбранном в качестве рабочего (Workspace) при запуске Eclipse. Например, рабочий каталог называется `workspaceADT`, в нем для нашего проекта появилась папка с именем `ProjectN`, в ней есть папка `src`, в ней `com`, в ней `example`, в ней `projectn` (заметьте, что название пакета распалось на три папки, каждое слово, отделенное точкой определило новую папку). И уже в папке `projectn` находится файл `MainActivity.java` и будут размещаться другие java-файлы проекта.

- папка **gen** содержит java-файлы, которые не требуется изменять и лучше вообще не трогать. Эти файлы генерируются автоматически. Нас может заинтересовать файл `R.java` он содержит идентификаторы (ID) для всех ресурсов приложения.

- папка **res** содержит структуру папок ресурсов приложения, рассмотрим некоторые из них:

- **layout** в данной папке содержатся xml-файлы, которые описывают внешний вид форм и их элементов, пока там находится только `activity_main.xml`;

- **values** содержит XML файлы, которые определяют простые значения, таких ресурсов как, строки, числа, цвета, темы, стили, которые можно

использовать в данном проекте;

– **menu** содержит XML файлы, которые определяют все меню приложения.

Рассмотрим файл **AndroidManifest.xml** файл в формате xml, который описывает основные свойства проекта, разрешение на использование ресурсов устройства и др. Сразу после создания приложения файл AndroidManifest.xml выглядит так, как показано на рисунке 7.2.

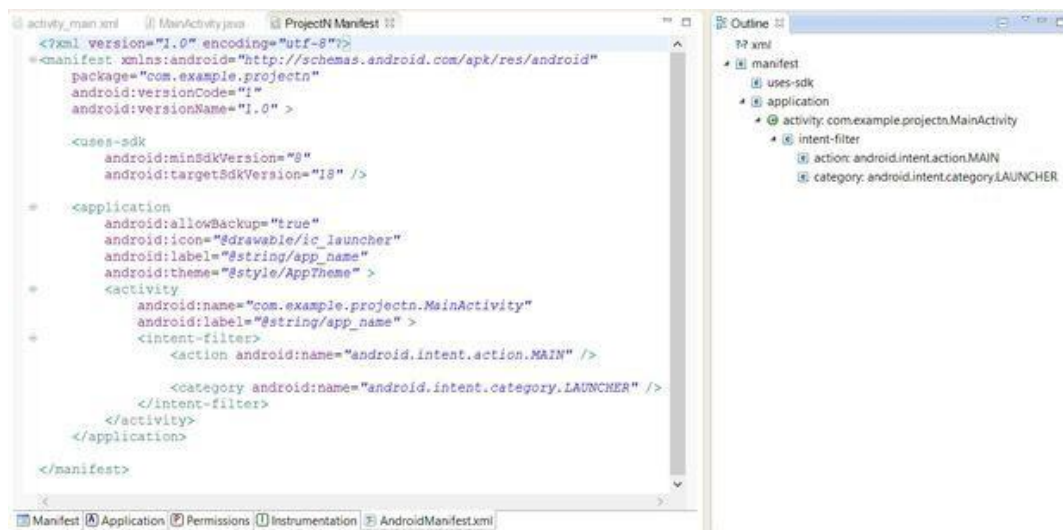


Рисунок 7.2 – Файл AndroidManifest.xml только созданного проекта

Рассмотрим подробно файл манифеста

Первый обязательный элемент <manifest> является корневым элементом файла, должен содержать обязательный элемент <application> и все остальные элементы по необходимости. Рассмотрим основные атрибуты этого элемента:

xmlns:android	определяет пространство имен Android, всегда должен иметь значение: «http://schemas.android.com/apk/res/android». Обязательный атрибут.

package	полное имя пакета, в котором располагается приложение. Обязательный атрибут. Имя должно быть уникальным, может содержать заглавные и строчные латинские буквы, числа и символ подчеркивания. Однако начинаться должно только с буквы. Для избежания конфликтов с другими разработчиками рекомендуется использовать имя вашего сайта (если он есть) записанное в обратном порядке. В нашем случае пакет имеет имя «com.example.projectn» и наше приложение не удастся разместить в Google Play (но мы на это и не претендуем). Внимание: если Вы опубликовали свое приложение, Вы не можете менять имя пакета, т.к. имя пакета служит уникальным идентификатором для приложения и в случае его смены приложение будет рассматриваться, как совсем другое и пользователи предыдущей версии не смогут его обновить.
android:versionCode	внутренний номер версии приложения не виден пользователю. Этот номер используется только для определения является ли одна версия более современной по сравнению с другой, больший номер показывает более позднюю версию.
android:versionNumber	номер версии, является строкой и используется только для того, чтобы показать пользователю номер версии приложения.
android:shareUserID, android:sharedUserLabel, , android:installLocation	эти атрибуты в нашем файле манифеста не представлены, про их назначение можно почитать по ссылке: http://developer.android.com/guide/topics/manifest/manifest

Рассмотрим элемент `<uses-sdk>`, который показывает совместимость приложения с версиями Android. Основные атрибуты:

android:minSdkVersion	указывает значение минимального уровня API, необходимого для работы приложения. Система Android не позволит установить приложение, если уровень API ниже, чем уровень, указанный в этом атрибуте. Внимание: если этот атрибут не указан, система установит значение по умолчанию равным «1», которое означает, что приложение совместимо со всеми версиями Android. И в случае, если приложение не совместимо со всеми версиями, установка пройдет на любую версию Android, а во время работы приложение сломается, когда попытается обратиться к недоступным элементам API. Поэтому необходимо всегда указывать значение этого атрибута.
android:targetSdkVersion	указывает уровень API целевой платформы Android приложения, если этот атрибут пропущен, по умолчанию принимается значение android:minSdkVersion.
android:maxSdkVersion	указывает максимальное значение уровня API, под который разрабатывалось приложение. Если значение этого атрибута ниже, чем уровень API соответствующий версии Android, на которую устанавливается приложение, то система не позволит установить такое приложение.

Если внимательно посмотреть на файл манифеста (рис. 7.2), можно заметить, что данный атрибут в нем отсутствует. На самом деле разработчики не рекомендуют задавать значение этого атрибута. Во-первых, это значение будет препятствовать использованию приложений на новых версиях Android при их появлении, несмотря на то, что все новые версии полностью обратно-совместимы. Во-вторых, стоит иметь в виду, что задание этого атрибута может

привести к тому, что приложение будет удалено с устройства после обновления Android до более высокого уровня API.

Подробнее с элементом `<uses-sdk>` и его атрибутами можно ознакомиться по ссылке:

<http://developer.android.com/guide/topics/manifest/uses-sdk-element.html>.

Рассмотрим элемент `<application>`, который является обязательным элементом манифеста, полностью определяет состав приложения. Представляет собой контейнер для элементов `<activity>`, `<service>`, `<receiver>`, `<provider>` (и не только), каждый из которых определяет соответствующий компонент приложения. Содержит набор атрибутов, действие которых распространяется на все компоненты приложения. Рассмотрим атрибуты элемента `<application>`, представленные в манифесте:

android:allowBackup	определяет разрешение для приложения участвовать в резервном копировании и восстановлении. Если значение этого атрибута false,
android:icon	определяет иконку для приложения целиком, а также иконку по умолчанию для компонентов приложения, которая может быть переопределена атрибутом android:icon каждого компонента. Задается как ссылка на графический ресурс, содержащий изображение, в нашем случае значение этого атрибута равно «@drawable/ic_launcher».
android:label	определяет видимый для пользователя заголовок приложения целиком, а также заголовок по умолчанию для компонентов приложения, который может быть переопределен атрибутом android:label каждого компонента. Задается как ссылка на строковый ресурс, в нашем случае значение атрибута равно «@string/app_name».

android:theme	определяет тему по умолчанию для всех активностей приложения, может быть переопределен атрибутом android:theme каждой активности. Задается как ссылка на стилевой ресурс, в нашем случае значение атрибута равно «@style/AppTheme».

На самом деле у элемента `<application>` гораздо больше атрибутов, чем нам удалось рассмотреть, найти полный список атрибутов с описаниями можно по ссылке: <http://developer.android.com/guide/topics/manifest/application-element.html>.

В приложении всего одна активность, других компонентов нет, в связи с этим элемент `<application>` в манифесте содержит ровно один элемент `<activity>` и больше никаких других элементов не содержит. Рассмотрим элемент `<activity>`, который определяет активность. Для каждой активности обязательно необходим свой элемент `<activity>` в манифесте. Рассмотрим атрибуты элемента `<activity>`, представленные в манифесте:

android:name	определяет имя класса, который задает активность. Значение атрибута должно полностью определять имя класса с указанием пакета, в котором располагается класс. В нашем случае атрибут имеет значение: «com.example.projectn.MainActivity». Можно использовать сокращенную запись «.MainActivity», в этом случае добавляется имя пакета, определенное соответствующим атрибутом элемента <code><manifest></code> .

android:configChanges	перечисляет изменения конфигурации, которыми может управлять активность. Если конфигурация меняется во время работы, то по умолчанию активность останавливается и перезапускается. Если же изменение конфигурации указано в этом атрибуте, то при появлении этого изменения активность не перезапускается, вместо этого она продолжает работать и вызывает метод <code>onConfigurationChanged()</code> . В нашем случае атрибут имеет значение «orientation keyboardHidden screenSize», т. е. при смене ориентации экрана, смене размера экрана и изменении доступности клавиатуры не произойдет перезапуск активности.
android:label	определяет видимый пользователю заголовок активности, если он отличается от общего заголовка приложения. Задается как ссылка на строковый ресурс, в нашем случае значение атрибута равно «@string/app_name» (т.е. можно было и не задавать).
android:theme	определяет тему активности, если она отличается от общей темы приложения, заданной соответствующим атрибутом элемента <code><application></code> . Задается как ссылка на стилевой ресурс, в нашем случае значение атрибута равно «@style/FullscreenTheme».

На самом деле у элемента `<activity>` гораздо больше атрибутов, чем нам удалось рассмотреть, найти полный список атрибутов с описаниями можно по ссылке: <http://developer.android.com/guide/topics/manifest/application-element.html>.

В манифесте для нашего приложения элемент `<activity>` содержит ровно один элемент:

`<intent-filter>`, определяющий типы намерений, которые может принимать активность.

Этот элемент содержит два элемента: `<action>` и `<category>`.

Первый элемент определяет действия, которые проходят в фильтр намерений, при этом `<intent-filter>` должен содержать хотя бы один элемент `<action>`, в противном случае ни один объект-намерение не сможет пройти через фильтр и активность не возможно будет запустить. Элемент `<action>` имеет единственный атрибут `android:name=«android.intent.action.MAIN»`.

Второй элемент определяет имя категории в фильтре намерений. Имеет единственный атрибут `android:name=«android.intent.category.LAUNCHER»`.

На этом разбор манифеста приложения закончим, подробно с описанием всех элементов этого файла можно познакомиться по ссылке:

<http://developer.android.com/guide/topics/manifest/manifest-intro.html>.

Чаще всего, при создании приложения приходится иметь дело с папками **src**, **res/layout** и **res/values**, т.к. там находятся основные файлы проекта.

Задание 2. Настройка интерфейса приложения

До того как начнем формировать интерфейс, имеет смысл подготовить возможность проверки разрабатываемого приложения на ошибки. Чтобы не загружать каждый раз приложение на реальное устройство, в Android SDK предусмотрена возможность использования виртуального устройства (AVD или Android virtual device), эмулирующего работу реального смартфона. Процесс создания виртуального устройства или эмулятора подробно расписан в «Установка и настройка среды программирования ADT Bundle».

Пришло время задуматься о внешнем виде приложения. Для начала необходимо определить какие элементы графического интерфейса нам нужны, как эти элементы будут располагаться на форме и каким образом будет реализовано взаимодействие с пользователем. Так как приложение очень простое, то и интерфейс особой сложностью отличаться не будет. Нам потребуется поле для ввода чисел (**TextEdit**), текстовая метка для вывода информации (**TextView**) и кнопка для подтверждения введенного числа (**Button**). Располагать элементы интерфейса будем друг под другом, сверху информационная часть, ниже поле ввода, кнопку разместим в самом низу приложения. Взаимодействие приложения с пользователем организуется очень

просто: пользователь вводит число в поле для ввода и нажимает кнопку, читает результат в информационном поле и, либо радуется победе, либо вводит новое число.

Если пользователь вводит правильное число, то приложение предлагает ему сыграть снова при этом кнопка будет играть роль подтверждения, а в информационное поле будет выведено приглашение к повторной игре.

Схематично интерфейс приложения изображен на рисунке 7.3.

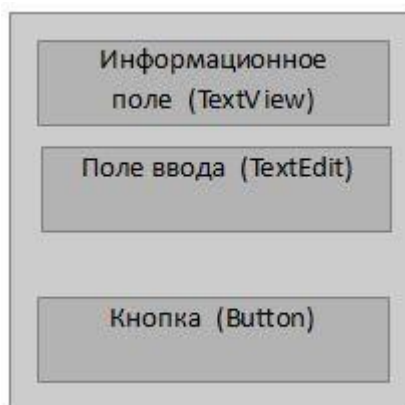


Рисунок 7.3 – Схема интерфейса приложения «Угадай число»

Нам необходимо добавить на форму три элемента: информационное поле (**TextView**), поле ввода (**TextEdit**) и кнопку (**Button**).

Android IDE поддерживает два способа для выполнения действий по формированию интерфейса приложения: первый основан на XML-разметке, второй относится к визуальному программированию и позволяет перетаскивать объекты интерфейса и размещать их на форме с помощью мыши. Считается, что визуальный способ подходит для новичков, а более продвинутые разработчики могут писать код вручную, однако чаще всего используется комбинированный подход.

Для формирования интерфейса будем работать с файлом **res/layout/activity_main.xml**. На рисунке 7.4 можно увидеть редактор, соответствующий визуальному способу формирования интерфейса, этому режиму соответствует вкладка **Graphical Layout**.

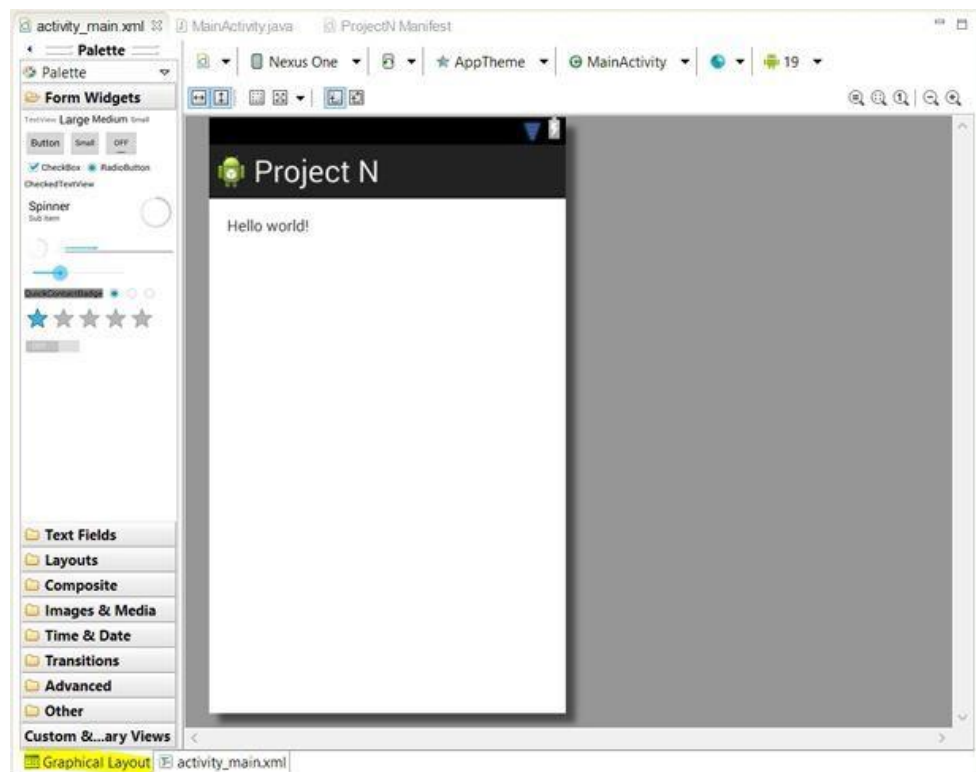
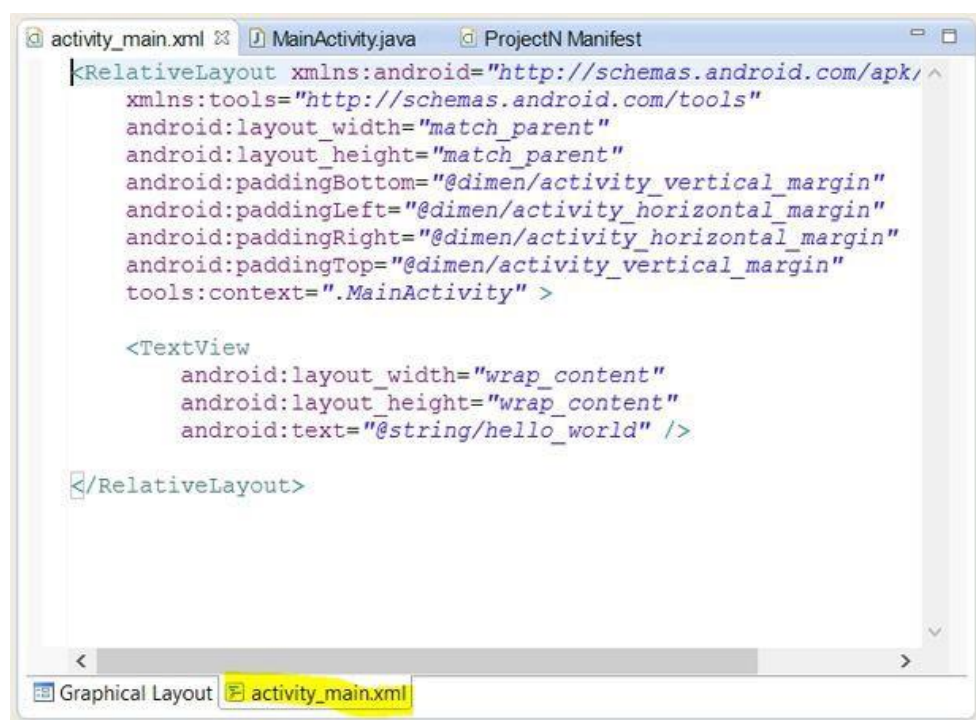


Рисунок 7.4 – Графическое изображение активности приложения

На рис.7.5 рядом с вкладкой **Graphical Layout** расположена вкладка **activity_fullscreen.xml**. Она соответствует режиму редактирования интерфейса путем формирования XML файла. На рисунке 7.5 можно увидеть редактор XML файла.



Рисунке 7.5 – Описание активности в XML формате

Зададим табличное расположение компонентов на форме, для этого выберем вкладку **Layouts**, найдем там **TableLayout** и добавим его на форму. На рисунке 7.6 можно увидеть результат этих действий.

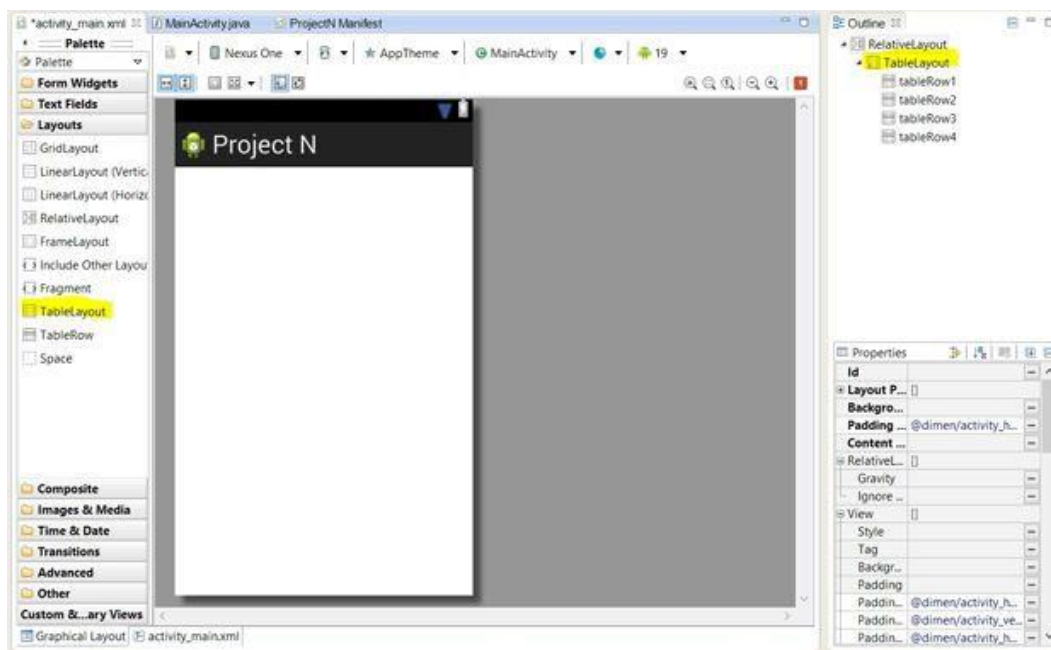


Рисунок 7.6 – Настройка интерфейса, добавление TableLayout

Теперь начнем добавлять элементы интерфейса, будем использовать графический режим правки.

Во-первых, нам необходимо добавить информационное поле. Для этого на панели

=**Palette**= выбираем вкладку **Form Widgets**, на этой вкладке найдем поле **TextView**, перенесем в окно приложения, разместим в первой строке таблицы (**TableLayout**).

Во-вторых, нам потребуется поле ввода информации, на вкладке **Text Fields** найдем текстовое поле **Number** и разместим во второй строке таблицы.

В-третьих, вернемся на вкладку **Form Widgets**, выберем там элемент **Button** и добавим в третью строку таблицы. Не нужную четвертую строку таблицы удалим, получим следующий вид приложения (рис. 7.7).

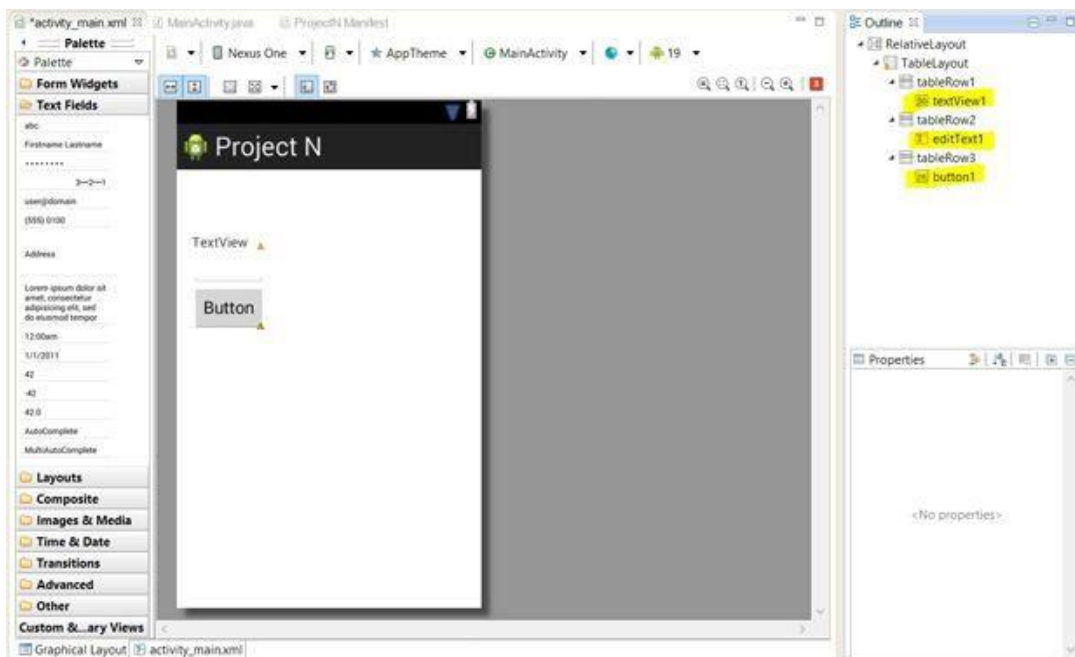


Рисунок 7.7 – Интерфейс приложения

После настройки интерфейса можно заглянуть в файл **activity_main.xml**, в этом файле прописано, что используется **TableLayout** и дано описание каждой из трех строк. На рисунке 7.8 можно увидеть, как выглядит такое описание на примере первой строки таблицы.

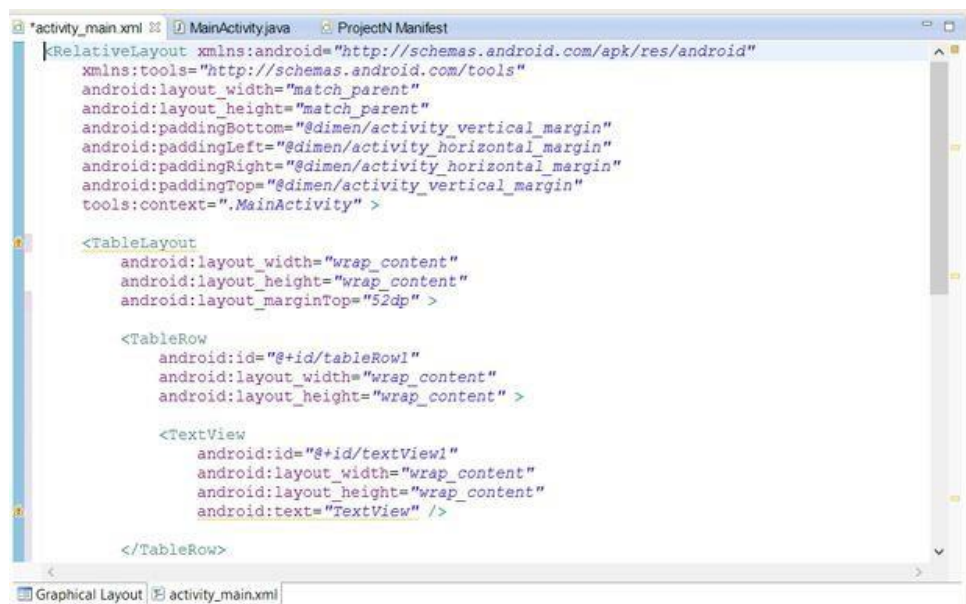


Рисунок 7.8 – Фрагмент файла activity_fullscreen.xml, описание строки в TableLayout

Теперь необходимо наполнить наши элементы интерфейса смыслом, нам понадобится текст для общения с пользователем, при программировании под

Android существует практика разделять ресурсы и код приложения. Для хранения любых строк, которые могут понадобиться приложению, используется файл **strings.xml**. Хранение всех строковых ресурсов в этом файле серьезно облегчает локализацию приложения на другие языки. Этот файл можно найти в Package Explorer в папке **res/values**. Откроем его и посмотрим, что там есть (рис. 7.9).

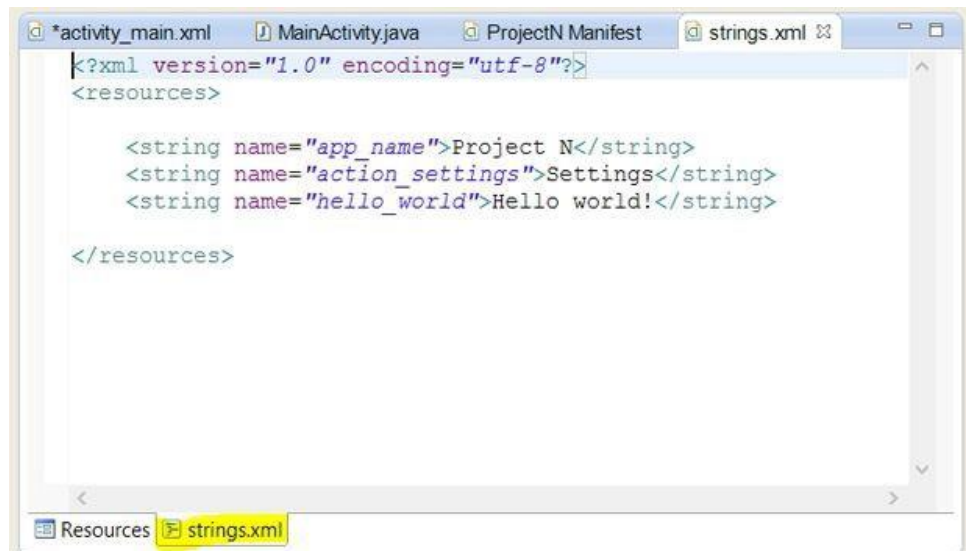


Рисунок 7.9 – Файл string.xml

Уберем лишние строки и добавим новые, результат можно посмотреть на рисунке 7.10.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <string name="app_name">Угадай число!</string>
    <string name="behind">Недолёт!</string>
    <string name="ahead">Перелёт!</string>
    <string name="hit">В точку!</string>
    <string name="input_value">Ввести значение</string>
    <string name="play_more">Сыграть ещё</string>
    <string name="try_to_guess">Попробуйте угадать число (1 &#8211; 100) </string>
    <string name="error">Неверный ввод!</string>

</resources>
```

Рисунок 7.10 – Отредактированный файл string.xml

Данные переменные будут выполнять следующие задачи:

- 1) app_name установит «видимое» название приложения;
- 2) behind, ahead, hit оповестят пользователя об его успехах в игре;
- 3) play_more и try_to_guess установит название кнопки, которое объяснит

её функции;

4) `input_value` пригласит пользователя к вводу числа;

5) `error` сообщит о неверном вводе.

После изменения **strings.xml**, при переходе на другую вкладку, не забудьте сохранить изменения (самый быстрый способ нажать **Ctrl+S**).

Настроим текст в информационном поле. Для этого на вкладке **Properties** в правой части окна выберем элемент **textView1** (это и есть наше информационное поле, имеет смысл придумать ему более осмысленное имя). Найдем свойство **Text**, подставим в него значение строки с именем `try_to_guess` (рис. 7.11).

Аналогично можно настроить текст, которым нас будет приветствовать кнопка, только в этом случае надо работать с элементом **button1**.

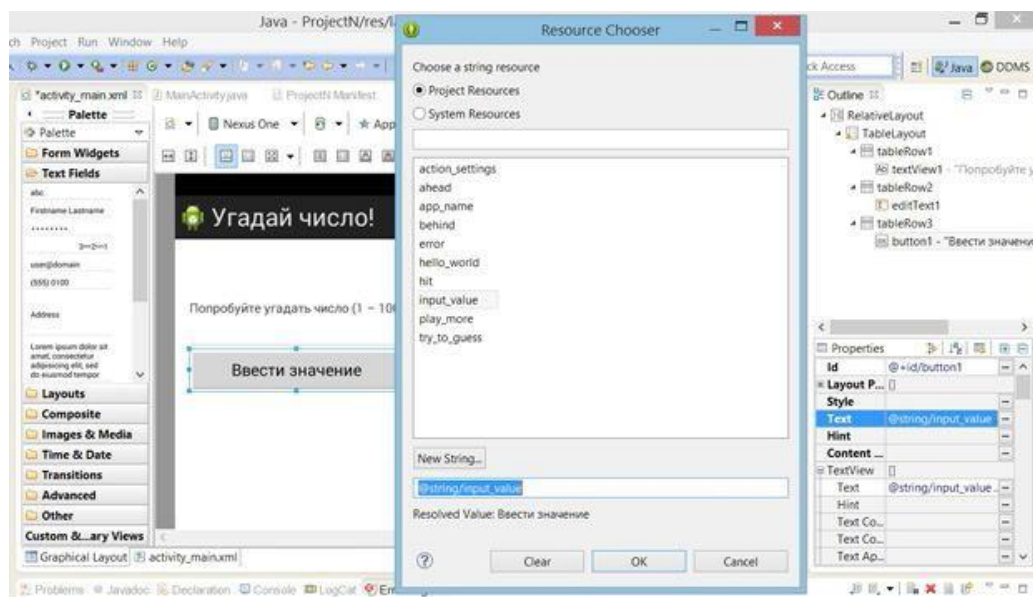


Рисунок 7.11 – Настройка текста для кнопки `button1`

Пришло время вспомнить о виртуальном устройстве, если оно работает, уже можно запустить проект и посмотреть, как приложение будет выглядеть на экране устройства, а выглядеть оно может как показано на рисунке 7.12.

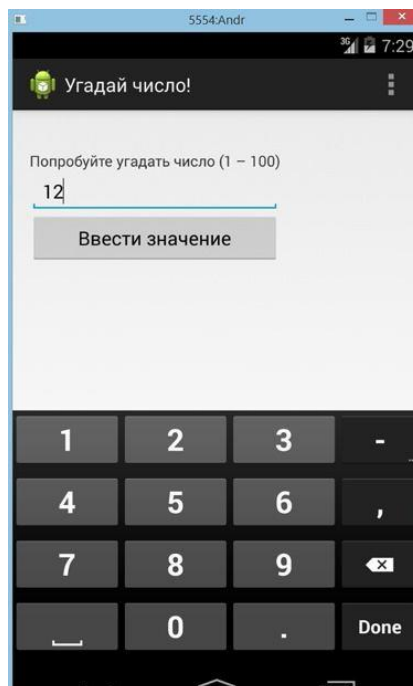


Рисунок 7.12 – Запуск приложения на виртуальном устройстве

Приложение выглядит довольно просто, но мы на многое и не рассчитывали. Главное, что нас интересует, это наличие всех элементов на экране, верный текст в каждом элементе, где он предусмотрен и возможность вводить числа в поле ввода. На рисунке 7.12 видно, что все требования выполнены. Приложение есть, его можно запустить на виртуальном или реальном устройстве, но оно ничего не делает. Следующим шагом будет реализация логики приложения, т. е. обработка события нажатия на кнопку, как было прописано в задании.

Задание 3. Реализация логики приложения

Приступим непосредственно к программированию, работать будем с файлом `src/com.example.projectn/MainActivity.java`. Найдем этот файл в Package Explorer (рис. 7.13), откроем и начнем редактировать.

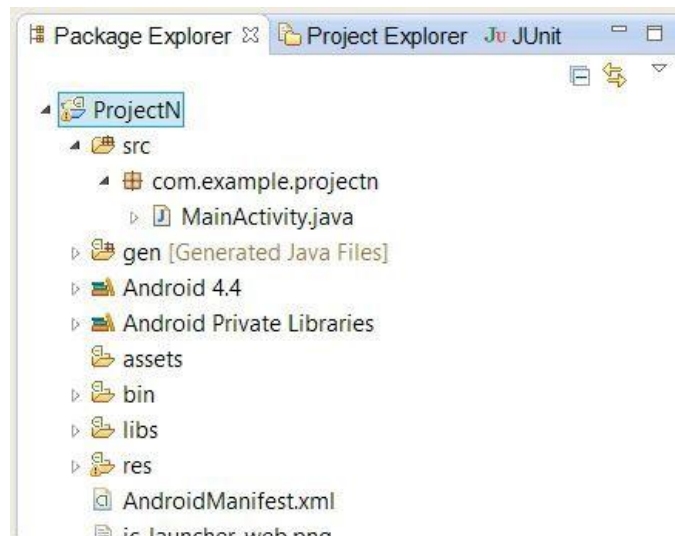


Рисунок 7.13 – Файл MainActivity.java в Package Explorer

Пока файл выглядит следующим образом (рис. 7.14).

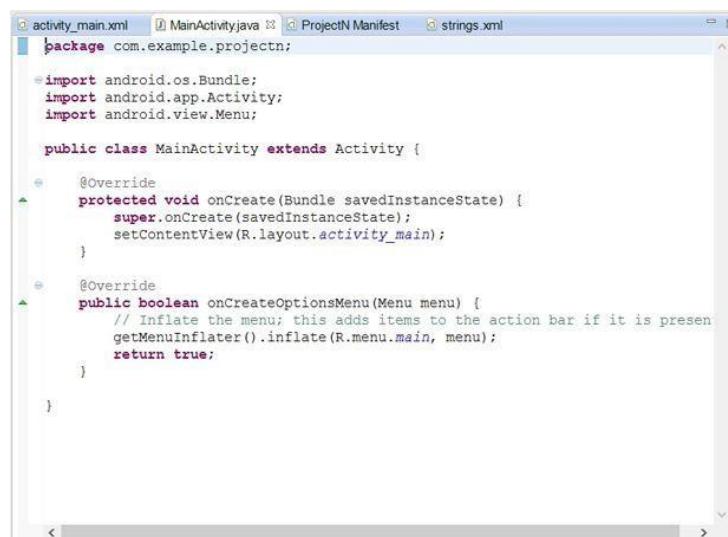


Рисунок 7.14 – Файл MainActivity.java после создания приложения

Можно заметить, что класс MainActivity является наследником класса Activity и в нем уже реализован метод onCreate(), который запускается при первоначальном создании активности, нам потребуется его дополнить, но об этом чуть позже.

Мы предполагаем программно менять информацию в поле **TextView**, получать значение из поля **EditText** и обрабатывать события нажатия на кнопку **Button**, поэтому необходимо объявить соответствующие переменные, как поля

класса MainActivity:

```
TextView tvInfo; EditText etInput; Button bControl;
```

Чтобы не было ошибок, необходимо импортировать пакет `android.widget`, который содержит все элементы графического интерфейса:

```
import android.widget.*;
```

На самом деле среда разработки подскажет, что делать.

Теперь необходимо связать эти переменные с элементами интерфейса, уже добавленными нами в **activity_main.xml**, сделать это необходимо в методе `onCreate()`, а для получения уже созданного элемента интерфейса воспользуемся методом `findViewById()`. Итак в метод `onCreate()` добавим следующие строки:

```
tvInfo      =      (TextView) findViewById(R.id.textView1);  
etInput = (EditText) findViewById(R.id.editText1); bControl  
= (Button) findViewById(R.id.button1);
```

Метод `findViewById()` возвращает объект класса `View`, который является общим предком для всех компонентов пользовательского интерфейса, для того чтобы избежать возможных ошибок в скобках перед вызовом метода указываем до какого конкретно компонента необходимо сузить возможности объекта `View`.

Пришло время выполнить обработку нажатия на кнопку. Вернемся к файлу **activity_main.xml** в графический режим редактирования, выберем элемент **Button** и на вкладке со свойствами элемента найдем свойство **On Click** и запишем в него **onClick** имя метода, который будет обрабатывать нажатие на кнопку. Как это выглядит, показывает рисунок 7.15.

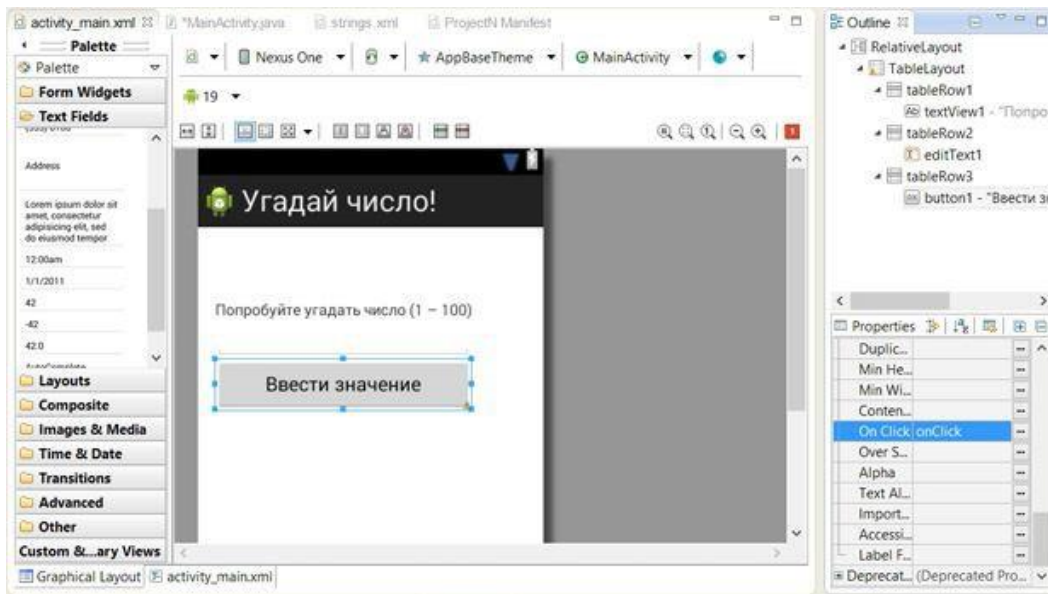


Рисунок 7.15 – Настройка свойства On Click для кнопки

Эти же действия можно выполнить в файле **activity_main.xml**, достаточно дописать выделенную на рисунке 7.16 строку:

```
<Button
    android:id="@+id/button1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="onClick"
    android:text="@string/input_value" />
```

Рисунок 7.16 – Настройка свойствами On Click для кнопки в файле XML

Для настройки свойств элементов интерфейса достаточно использовать любой способ: графический или редактирование XML файла.

Вернемся в файл **MainActivity.java**, в класс активности необходимо добавить метод:

```
public void onClick(View v){...}
```

Имя метода не обязательно должно быть `onClick()`, главное, чтобы оно совпадало с именем, указанным в свойстве **On Click**. В этом методе и будет происходить все наше программирование в этой лабораторной работе.

Нам потребуются две переменные;

- 6) целочисленная для хранения загаданного числа (случайное число от

1 до 100);

7) логическая для хранения состояния закончена игра или нет.

Обе эти переменные имеет смысл объявить как поля класса активности, первоначальные значения присвоить в методе onCreate.

Получить целочисленное значение из поля ввода, можно с помощью следующей конструкции:

```
Integer.parseInt(etInput.getText().toString())
```

изменить значение текста в информационно поле можно с помощью следующей конструкции:

```
tvInfo.setText(getResources().getString(R.string.ahead));
```

в данном случае в информационном поле появится значение строкового ресурса с именем ahead.

Осталось реализовать логику приложения в методе onClick(). Предлагаем написать код этого метода самостоятельно, для контроля в приложении предложен листинг, который содержит один из вариантов кода описанного приложения.

Немного о работе с эмулятором

При выполнении данной работы мы использовали эмулятор для проверки работоспособности приложения. Рассмотрим некоторые возможности, облегчающие и ускоряющие взаимодействие с виртуальным устройством.

Во-первых, существует набор полезных комбинаций клавиш для управления виртуальным устройством:

8) **Alt+Enter** разворачивает эмулятор до размеров экрана;

9) **Ctrl+F11** меняет ориентацию эмулятора с портретной на альбомную и обратно;

10) **F8** включает/выключает сеть.

Полный список комбинаций клавиш для работы с эмулятором можно найти по ссылке:

<http://developer.android.com/tools/help/emulator.html>.

Во-вторых, кто бы ни работал с эмулятором, тот на себе прочувствовал насколько терпеливым надо быть, чтобы взаимодействовать с ним, так медленно он работает. Существует решение для ускорения работы эмулятора Android и этим решением является Intel Hardware Accelerated Execution Manager (Intel® HAXM).

Intel Hardware Accelerated Execution Manager (Intel® HAXM) это приложение с поддержкой аппаратной виртуализации (гипервизор), которое использует технологию виртуализации Intel для ускорения эмуляции приложений Android на компьютере для разработки. (<http://software.intel.com/ru-ru/android/articles/intel-hardware-accelerated-execution-manager>)

Intel HAXM способен ускорить работу эмулятора для x86 устройств. При этом эмулятор будет работать со скоростью, приближенной к скорости работы реального устройства, что поможет сократить время на запуск и отладку приложения. Подробно познакомиться с установкой Intel HAXM и настройкой эмулятора на работу с ускорителем можно в статье по ссылке: <http://habrahabr.ru/company/intel/blog/146114/>.

В-третьих, не стоит прерывать процесс запуска виртуального устройства, наберитесь терпения, на старых компьютерах первый запуск AVD может занять до 10 минут, на современных от одной до трех минут. После того, как создали и запустили виртуальное устройство имеет смысл оставить его открытым, при всех последующих запусках приложения будет использоваться уже открытое виртуальное устройство, что позволит сэкономить время.

В лабораторной работе рассмотрен процесс разработки простого приложения переднего плана. Описано создание активности, настройка интерфейса и реализация логики приложения. Других компонентов в приложении не предусмотрено. В последующих работах будут рассматриваться приложения, содержащие несколько активностей. Смешанные приложения, работающие на переднем плане и при этом поддерживающие сервисы, работающие в фоновом режиме.

Отчет к лабораторной работе оформляется в виде текстового документа по форме простого реферата и должен включать:

1. Название лабораторной работы.
2. Цель лабораторной работы.
3. Краткие выводы по результатам выполнения лабораторной работы согласно пунктам раздела «Методики выполнения работы».
4. Ответы на Вопросы к лабораторной работе.
5. Формулировку задания для самостоятельной работы и результат его выполнения.

Задания для самостоятельной работы

1. Создать новое приложение и изучить его структуру.
2. Настроить интерфейс приложения.
3. Реализовать логику приложения.
4. Что произойдет, если кнопка будет нажата до ввода какого-либо числа? Скорей всего

приложение будет остановлено, необходимо как-то отслеживать этот вариант развития событий и адекватно реагировать. Предлагаем поработать самостоятельно над этой проблемой.

Что произойдет, если пользователь введет число меньшее нуля или большее 100? Скорей всего приложение обработает этот ввод, но было бы лучше, если бы появилось сообщение о том, что введенное число не удовлетворяет условиям задачи.

5. Как завершить приложение? И надо ли это делать?

Предлагаем подумать, как еще можно улучшить приложение и реализовать эти улучшения.

Вопросы к лабораторной работе

1. Характеристика структуры проекта.
2. Характеристика структуры папок проекта.
3. Технология настройки интерфейса приложения.
4. Реализация логики приложения.


```

package com.example.projectn; import
android.os.Bundle;
import android.app.Activity;
import android.view.Menu; import android.view.View;
import android.widget.*;

public class MainActivity extends Activity { TextView
tvInfo;

EditText etInput;
Button bControl;
int guess;
boolean gameFinished;

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);

tvInfo = (TextView)findViewById(R.id.textView1);
etInput = (EditText)findViewById(R.id.editText1); bControl
= (Button)findViewById(R.id.button1);

```

Приложение 1

```

guess = (int) (Math.random()*100); gameFinished =
false;
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
// Inflate the menu; this adds items to the action bar
if it is present.

getMenuInflater().inflate(R.menu.main, menu);
return true;
}

public void onClick(View v) {
if (!gameFinished){

```

```

        int
inp=Integer.parseInt(etInput.getText().toString());
        if (inp > guess)
            tvInfo.setText(getResources().getString(R.string.ahead)); if (inp < guess)
            tvInfo.setText(getResources().getString(R.string.behind)); if (inp == guess)
            {
                tvInfo.setText(getResources().getString(R.string.hit));
            };
bControl.setText(getResources().getString(R.string.play_more)); gameFinished = true;
        }
    }
    else
    {
        guess = (int) (Math.random()*100);
        bControl.setText(getResources().getString(R.string.input_value));
        tvInfo.setText(getResources().getString(R.string.try_to_guess)); gameFinished = false;
    }
    etInput.setText(««);
}}

```

Лабораторная работа 8

Основы разработки интерфейсов мобильных приложений

Цель работы: изучение основ разработки интерфейсов мобильных приложений

Теоретическое обоснование

Лабораторная работа посвящена разработке интерфейсов мобильных приложений. Работа содержит подробное описание построения гармоничного понятного пользовательского интерфейса для главной активности приложения и описание основных элементов интерфейса. Лабораторная работа поможет выбрать концепцию своего приложения и начать разработку его интерфейса.

Более подробную информацию о разработке интерфейсов мобильных приложений под Android можно узнать на сайте [Design\[AndroidDevelopers\]](#).

Задания к лабораторной работе

Задание 1. Создание прототипа интерфейса

Рассмотрим пример разработки интерфейса приложения, которое ищет в сети Интернет изображения по запросу пользователя, позволяет оценивать их, скачивать, и посещать интернет-страницы сайтов, на которых было найдено изображение.

1.1 Создание заготовки для приложения

Выглядеть главное окно будет примерно так (рис. 8.1):

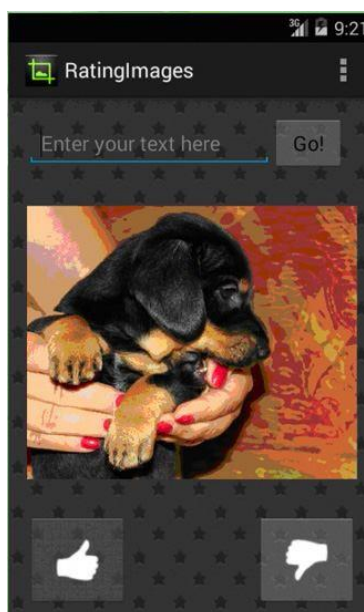


Рисунок 8.1 – Интерфейс главной активности

На нём присутствуют поле ввода текста для запроса пользователя и кнопка, начинающая поиск изображений. Внизу экрана две кнопки: «**like**» и «**dislike**», с их помощью пользователь сможет оценить изображение. После того, как пользователь сделает оценку изображения, текущее изображение закрывается и загружается следующее.

Итак, начнём с создания нового проекта. Назовём его «**RatingImages**» («Рейтинг изображений»).

На данном этапе изучения программирования под Android не обязательно менять иконку запуска, но эта лабораторная работа направлена на разработку дизайна интерфейса, и потому, приступим.

Создайте иконку на свой вкус (рис. 8.2).

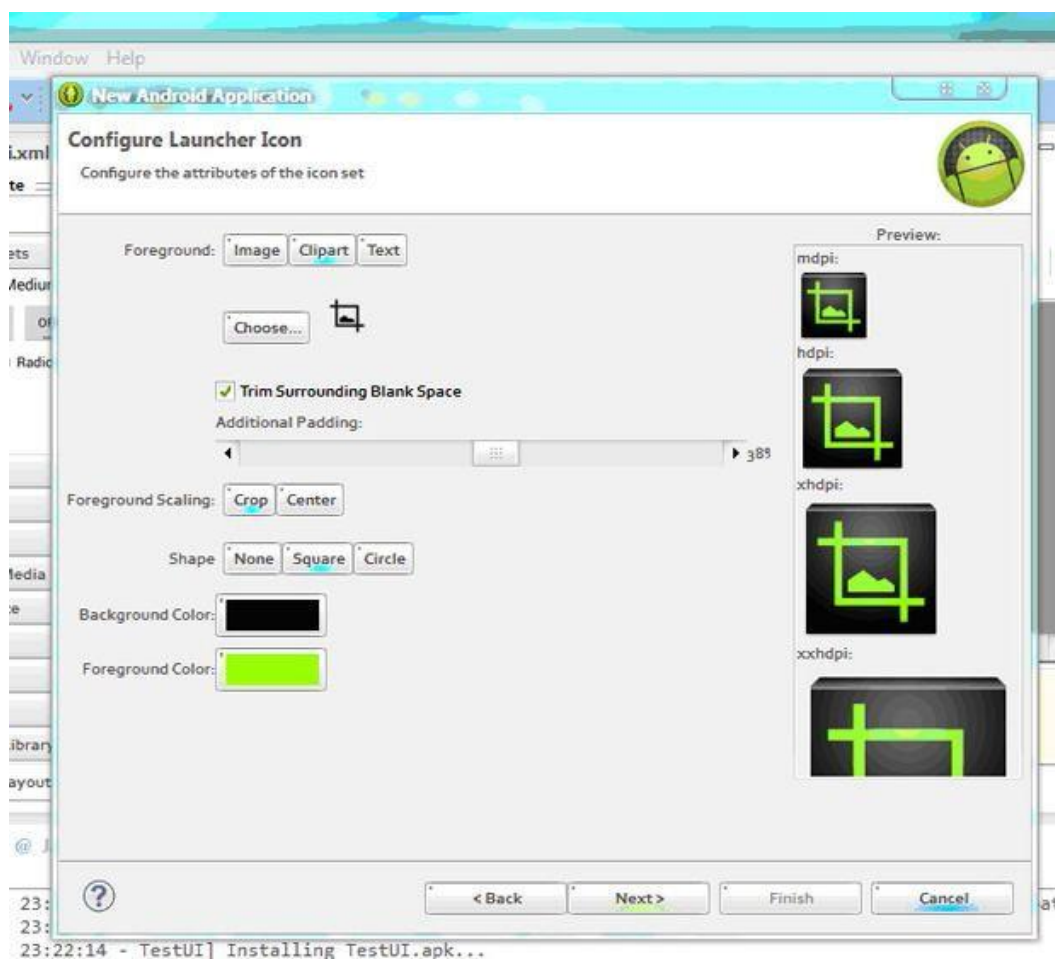


Рисунок 8.2 – Создание иконки

После создания проекта откройте **activity_main.xml** из каталога **res/layout/**.

Когда вы откроете файл **activity_main.xml**, вы увидите графический редактор макета. Благодаря этому редактору создание интерфейсов стало ещё

интереснее, поскольку добавить элемент на форму можно при помощи перетаскивания мышью, к тому же, благодаря графическому редактору, не обязательно запускать эмулятор, чтобы увидеть результат своих трудов.

Теперь щелкните по вкладке **activity_main.xml** в нижней части экрана. Открылся XML-редактор кода. Этот способ редактирования стандартный, но все изменения, вносимые в этот документ, можно так же ощутить визуально, перейдя на графический редактор. Вернёмся на вкладку с графическим редактором. Во-первых, подготовим документ к началу работы, для этого удалите `<TextView>`.

Результат выглядит так (рис. 8.3):

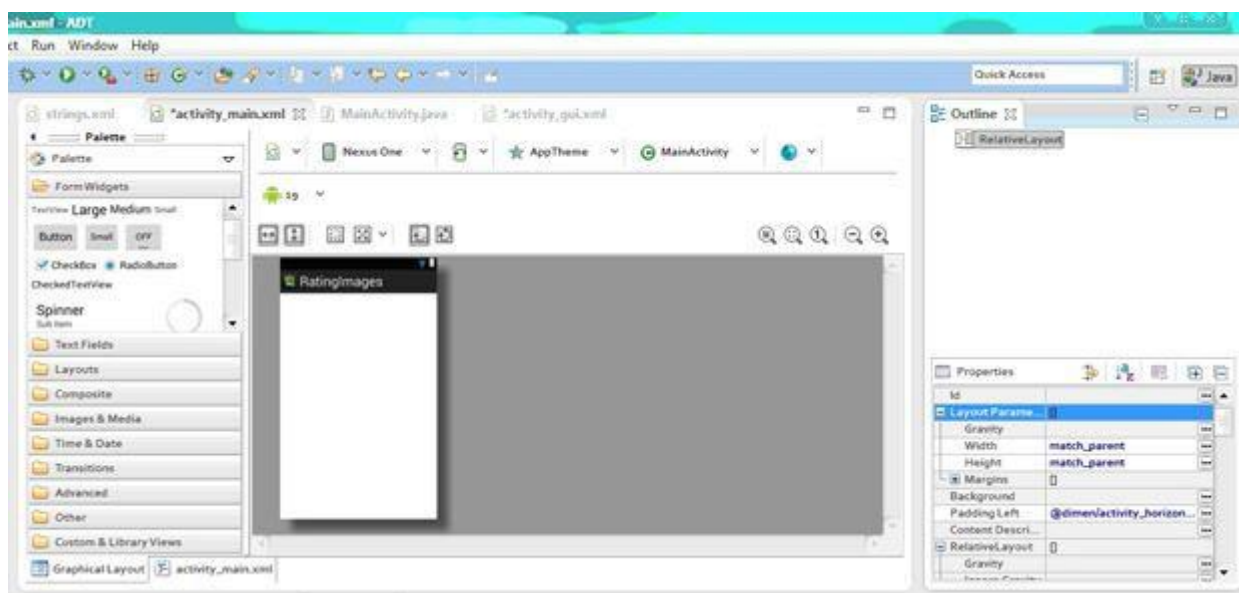


Рисунок 8.3 – Проект, готовый к началу разработки

На рабочей области экрана остался один элемент. Это макет `<RelativeLayout>`. В нём позиция дочерних элементов может быть описана по отношению друг к другу или к родителю.

Два атрибута, ширина и высота (`android:layout_width` и `android:layout_height`), требуются для всех элементов для того, чтобы указать их размер.

Так как `<RelativeLayout>` это корень в макете, то нужно, чтобы он заполнял всю область экрана. Это достигается при помощи установки параметра «`match_parent`» для ширины и высоты. Это значение указывает, что ширина и

высота элемента будет равна ширине и высоте родителя (рис. 8.4).

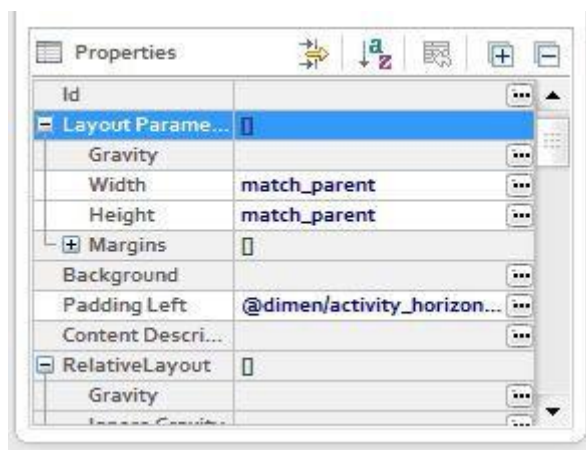


Рисунок 8.4 – Свойства <RelativeLayout> элемента

Однократным щелчком левой кнопкой мыши по надписи «**Width**» активируйте строку с параметрами ширины (рис. 8.5):

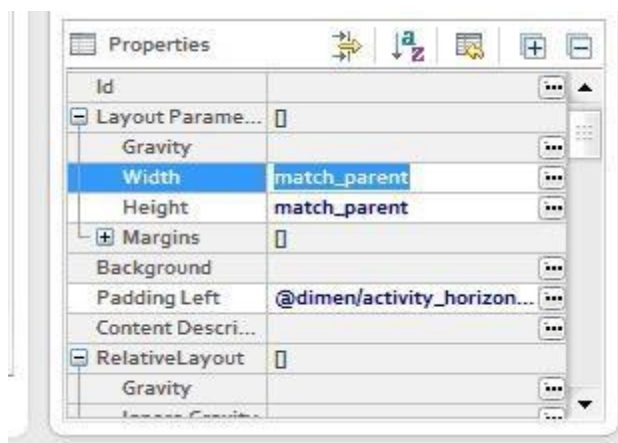


Рисунок 8.5 – Свойства <RelativeLayout> элемента, выбран атрибут android:layout_width

Щелчком левой кнопки мыши по области ввода вызовите диалоговое окно, и двойным щелчком сделайте выбор параметра (рис. 8.6):

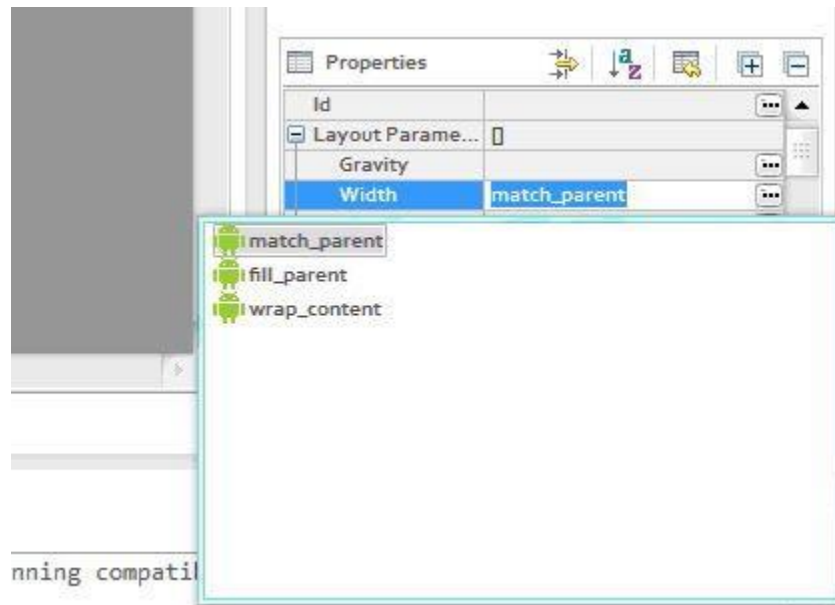


Рисунок 8.6 – Свойства <RelativeLayout> элемента

Далее выбран параметр «match_parent» Или щелчком правой кнопки мыши по <RelativeLayout> в Outline (рис. 8.7):

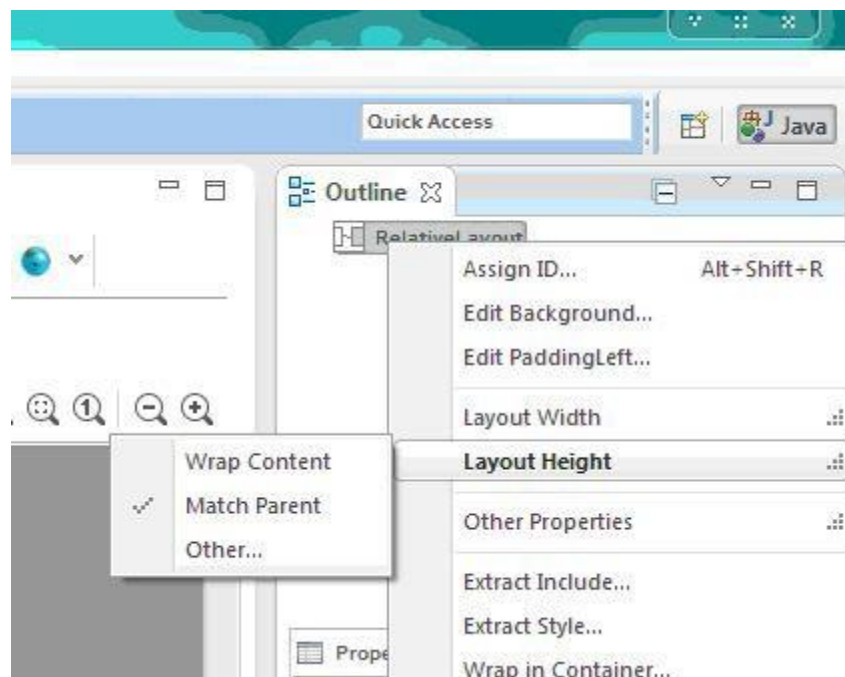


Рисунок 8.7 – Контекстное меню <RelativeLayout> элемента

При выполнении первого способа вы увидели еще два возможных параметра: «fill_parent» и «wrap_content».

На самом деле, match_parent = fill_parent, но «fill_parent» считается

устаревшим, и к использованию в новых проектах предлагается «match_parent».

Параметр «wrap_content» указывает, что представление будет увеличиваться при необходимости, чтобы поддерживать соответствие содержанию экрана.

1.2 Добавление текстового поля

Для начала добавьте элемент `<LinearLayout>` с горизонтальной ориентацией в `<RelativeLayout>`, и укажите для ширины и высоты параметр «wrap_content». Теперь, для создания пользовательского редактируемого текстового поля, добавьте элемент `<EditText>` с параметром «wrap_content» для ширины и высоты в `<LinearLayout>`.

Сейчас должно получиться примерно следующее (рис. 8.8):



Рисунок 8.8 – Добавление текстового поля

Возможно, появился желтый предупреждающий знак, но сейчас это не важно, со временем он исчезнет. Наличие таких предупреждений никак не влияет на компилируемость проекта.

Теперь переходим к настройке добавленных нами элементов.

Для многих элементов нужно назначать id, он обеспечивает уникальный идентификатор, который можно использовать как ссылку на объект из кода вашего приложения для управления им.

Откроем редактор кода XML-файла и обратим внимание на элемент

```
<EditText>:
```

```
<EditText
```

```
android:id=«@+id/editText1»
```

```
android:layout_width=«wrap_content»
```

```
android:layout_height=«wrap_content»>
```

```
<requestFocus/>
```

```
</EditText>
```

Строка `<requestFocus/>` появляется добавлением элемента `requestFocus` (папка `Advanced`), и позволяет установить фокус на нужном компоненте. Важно использовать этот элемент, когда у вас имеется, к примеру, три текстовых поля, и нужно, чтобы фокус был на втором из них.

При указании `id`, знак (`@`) требуется в том случае, если вы имеете в виду любой ресурс объекта из XML-файла. За ним следуют тип ресурса (в данном случае `ID`), косая черта (слеш) и имя ресурса (`editText1`).

Знак плюс (`+`) перед типом ресурсов необходим только тогда, когда вы впервые определяете идентификатор ресурса.

По сути, `id`, который создается автоматически, уже уникален, но грамотнее переименовывать `id` в соответствии со назначением элемента.

Зададим `id` для текстового поля. Для этого прямо в коде строку `android:id=«@+id/editText1»` заменяем на `android:id=«@+id/edit_message»`, жмём **CTRL+S** и открываем графический редактор. Если всё хорошо, то в свойствах текстового поля в графе **id** будет следующее (рис. 8.9):

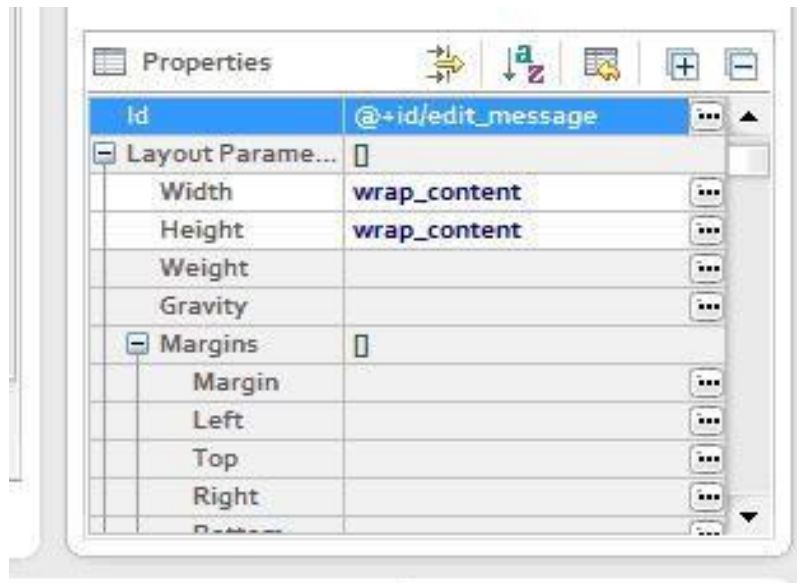


Рисунок 8.9 – Идентификатор текстового поля

Добавим в код ещё две строки: `android:ems=«10»` задает соответствия для симметричного отображения шрифтов, `android:hint=«@string/edit_message»` содержание тестового поля «по умолчанию», т.е. пока пользователь не начал вводить в поле текст.

Вместо того, чтобы использовать просто слово (например `android:hint=«message»`), что крайне не удобно при изменении основного языка приложения, используется ссылка на значение, хранящееся в файле **strings.xml**. Поскольку это относится к конкретному ресурсу (а не только к **id**), знак плюс не нужен. Однако, мы ещё не определили строку ресурсов файле **strings.xml**, и потому вы получите следующее (рис. 8.10):



Рисунок 8.10 – Предупреждение: не найден ресурс, на который прописана ссылка

Для того, чтобы ссылка на ресурс начала работать, нужно этот ресурс создать. Откройте файл **res/values/strings.xml**. Очевидно, что его тоже можно редактировать двумя способами: графически и вручную.

Выбирайте тот способ, который больше по душе (рис. 8.11).

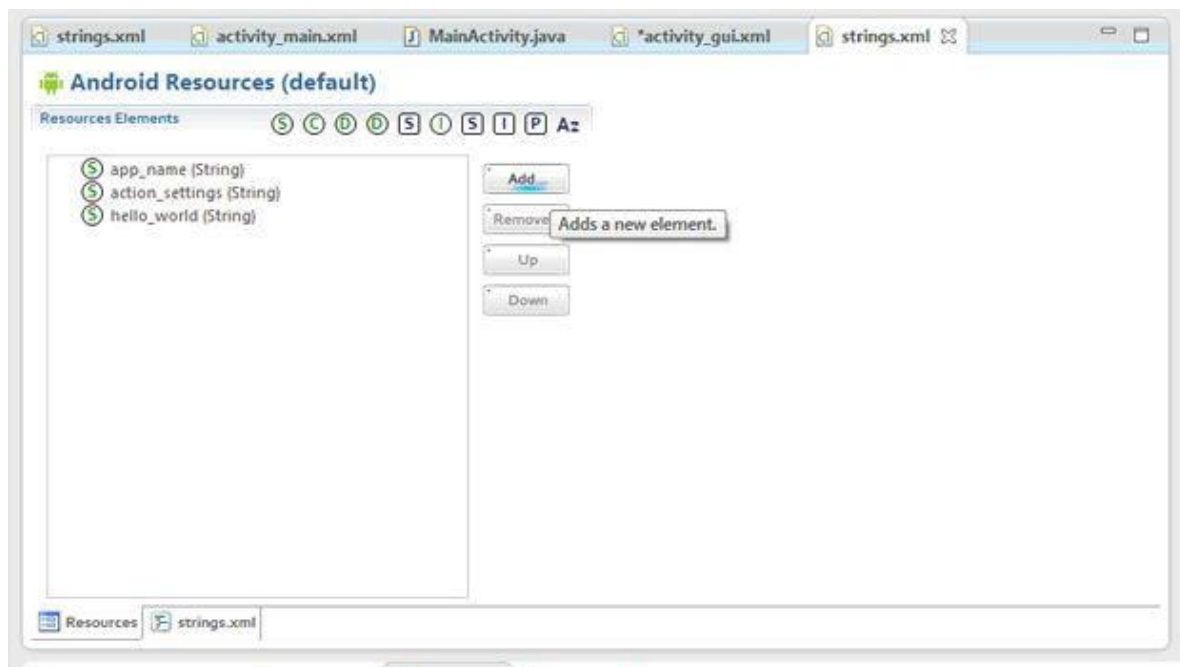


Рисунок 8.11 – Редактирование файла ресурсов графическим способом;
добавление нового ресурса

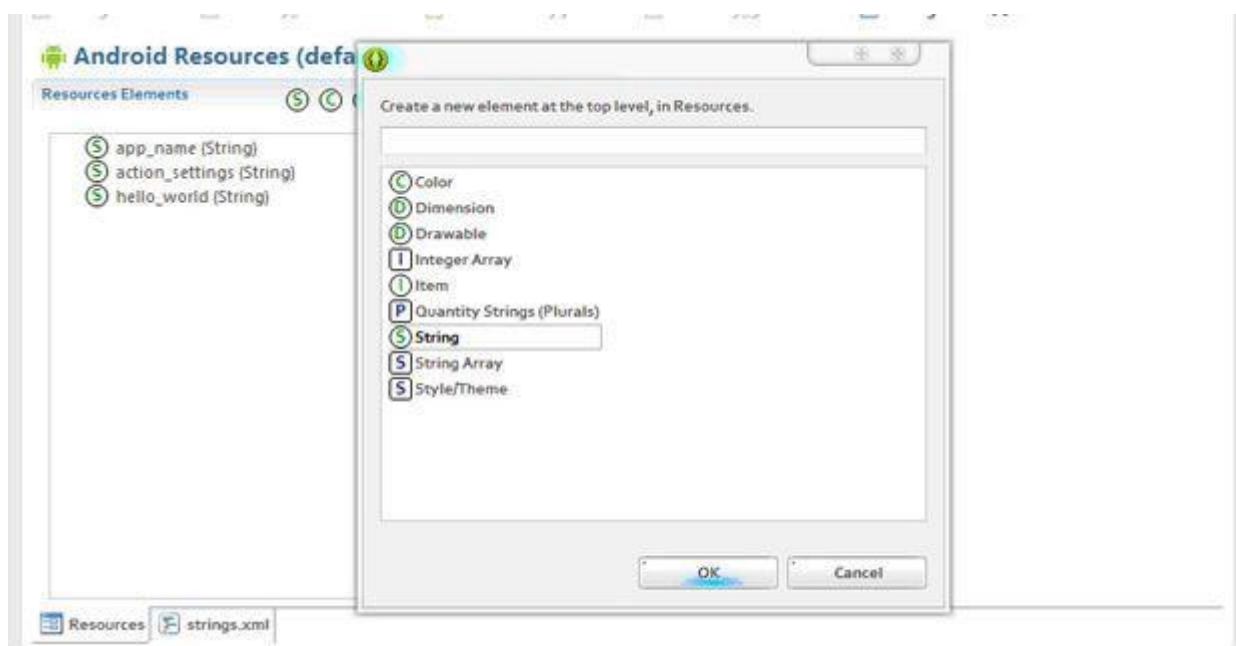


Рисунок 8.12 – Редактирование файла ресурсов графическим способом;
выбор типа ресурса, выбран тип String

Заполняем поля «Имя» и «Значение» (рис. 8.13):

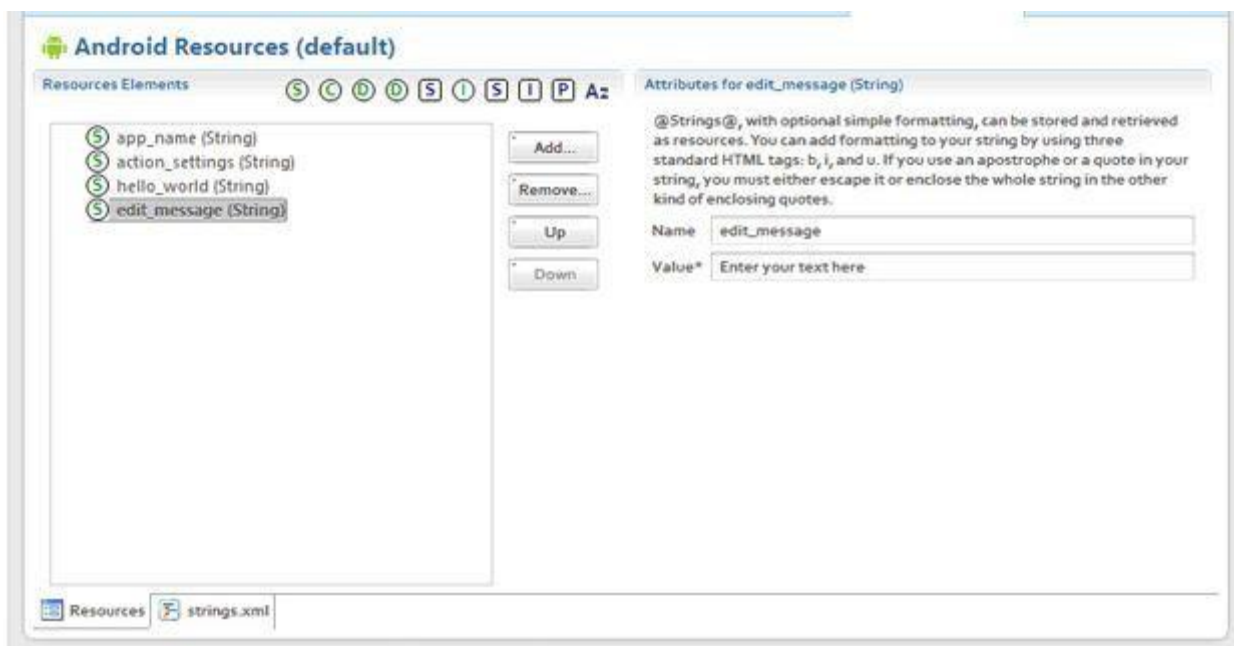


Рисунок 8.13 – Редактирование файла ресурсов графическим способом

Сохраняем и любимся результатом (рис. 8.14):

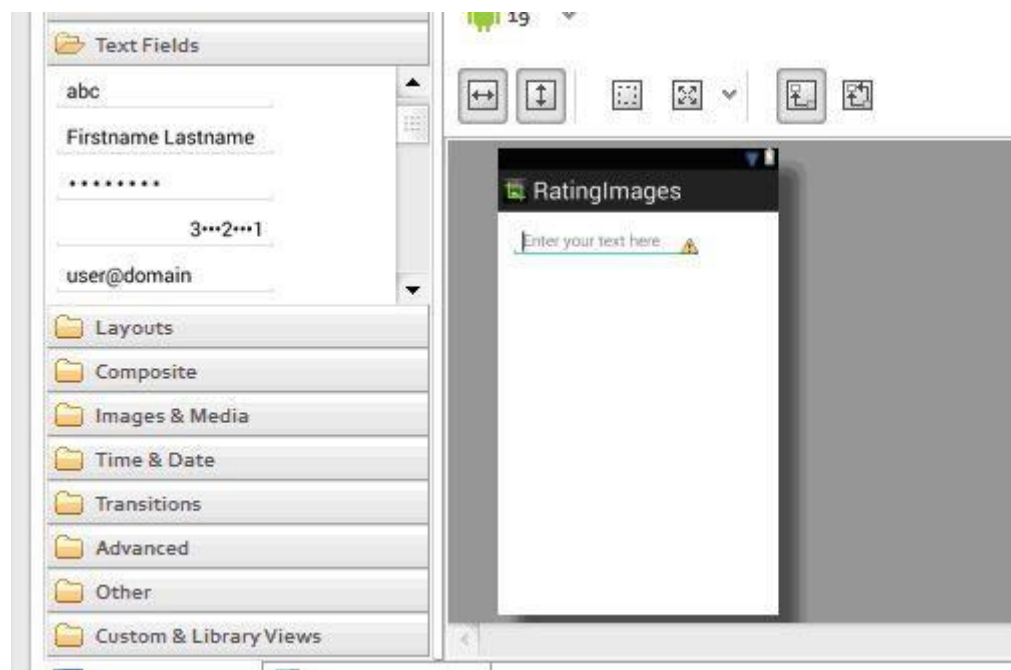


Рисунок 8.14 – Отображение текста «По умолчанию» в поле ввода

Аналогично создадим id для <LinearLayout>

```
android:id="@+id/linear1"
```

Сохраните изменения.

1.3 Добавление кнопки

Теперь добавьте <Button> в макет после элемента <EditText> (рис.8.15):

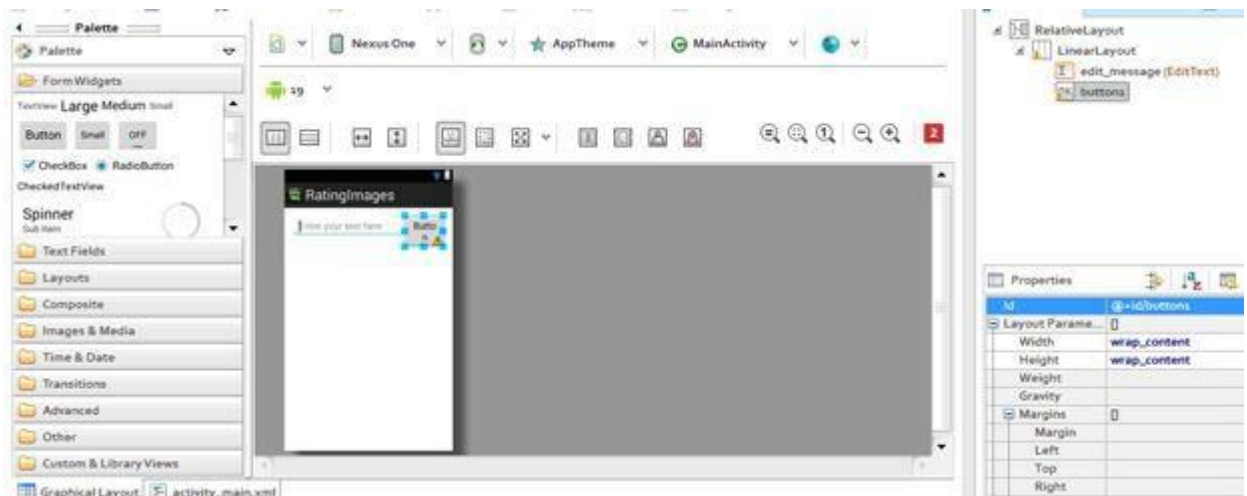


Рисунок 8.15 – Новая кнопка

Чтобы кнопка трансформировалась в соответствии с текстом кнопки, ширина и высота должны быть установлены во «wrap_content».

Теперь поменяем надпись на кнопке на «**Go!**» с помощью ссылки на ресурс в XML-коде главной активности и добавления одного ресурса в файл **strings.xml** (рис. 8.16):



Рисунок 8.16 – Редактор XML-кода файла strings.xml

Сохраните.

На графическом редакторе главной активности будут изменения (рис. 8.17):



Рисунок 8.17 – Кнопка приняла новую форму, в соответствии с надписью на ней

Теперь, когда мы поместили два главных представления на `<LinearLayout>` элемент, настало время добавить ещё два параметра для этого элемента.

Речь идет о «приращении» правого и левого краёв лейаута к правому и левому краям `<RelativeLayout>` элемента соответственно. А сделать это проще простого просто потяните мышкой один край к другому (рис. 8.18).

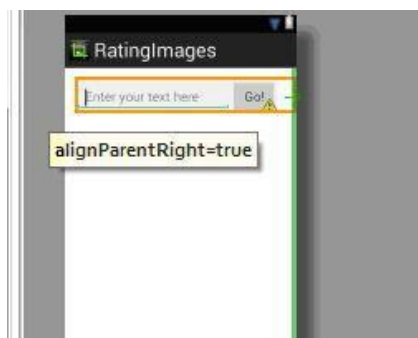


Рисунок 8.18 – «Приращение» правого края

Зелёные стрелки по краям дают понять, к какому элементу удалось прицепить край (рис. 8.19):

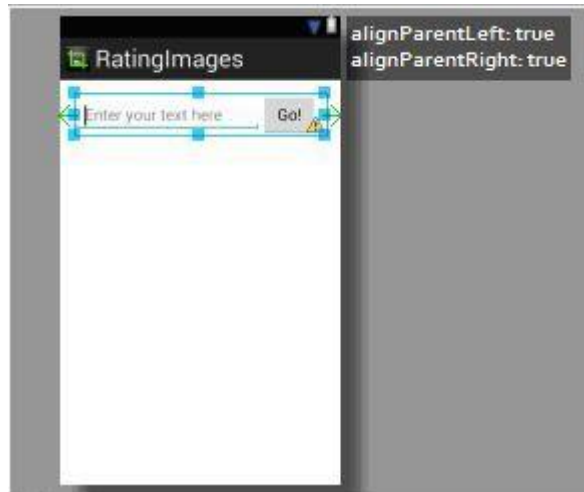


Рисунок 8.19 – <LinearLayout> после выравнивания

Конечно, это отобразится в свойствах (рис. 8.20):

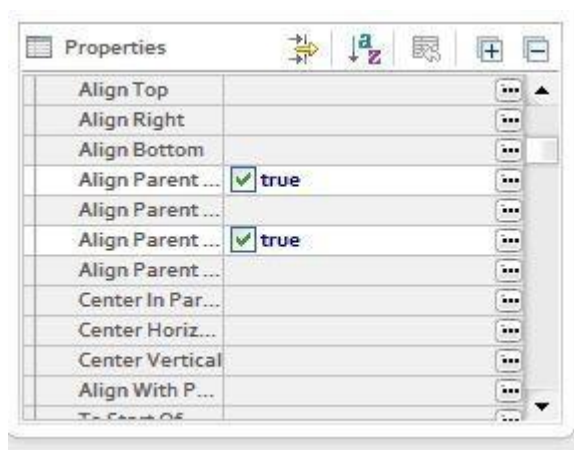


Рисунок 8.20 – Свойства <LinearLayout>

1.4 Смена фона

Идём дальше попробуем поменять фон. Чтобы изменить цвет фона на чёрный, нужно в XML-коде главной активности написать одну строку в блоке <RelativeLayout> элемента: `android:background=«#000000»` (рис. 8.21). Сохраните и проверьте результат, открыв графический редактор.

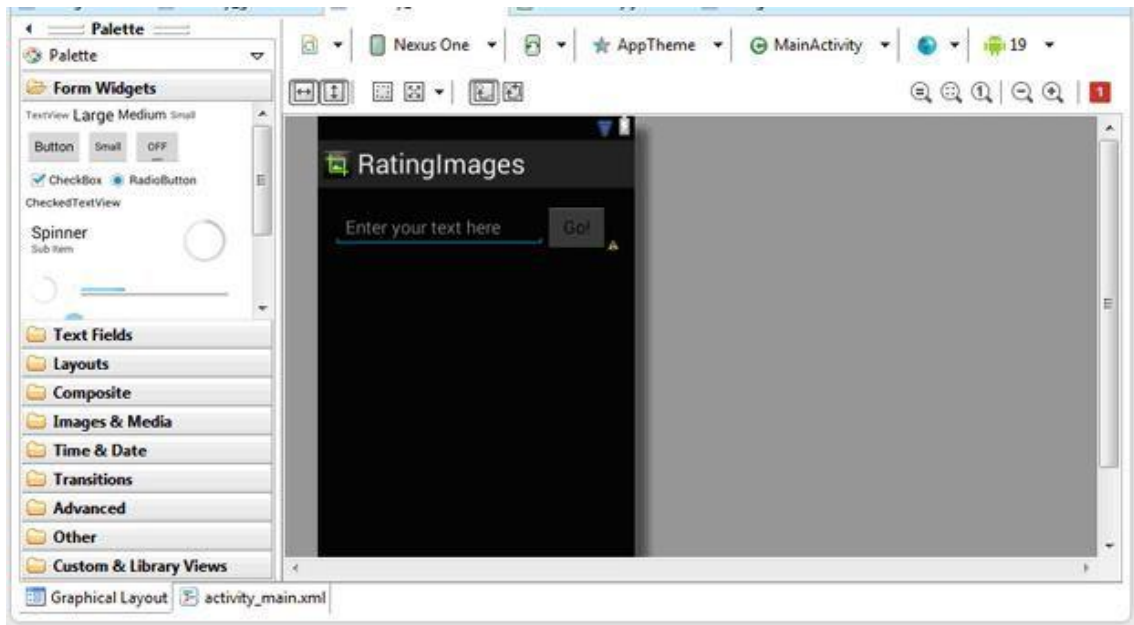


Рисунок 8.21 – Фон стал чёрным

Красиво, но скучно. Как сделать фон ещё интереснее? Поместить на него рисунок! Для этого сначала в папке **res/** создадим папку **drawable** (рис. 8.22, 8.23).

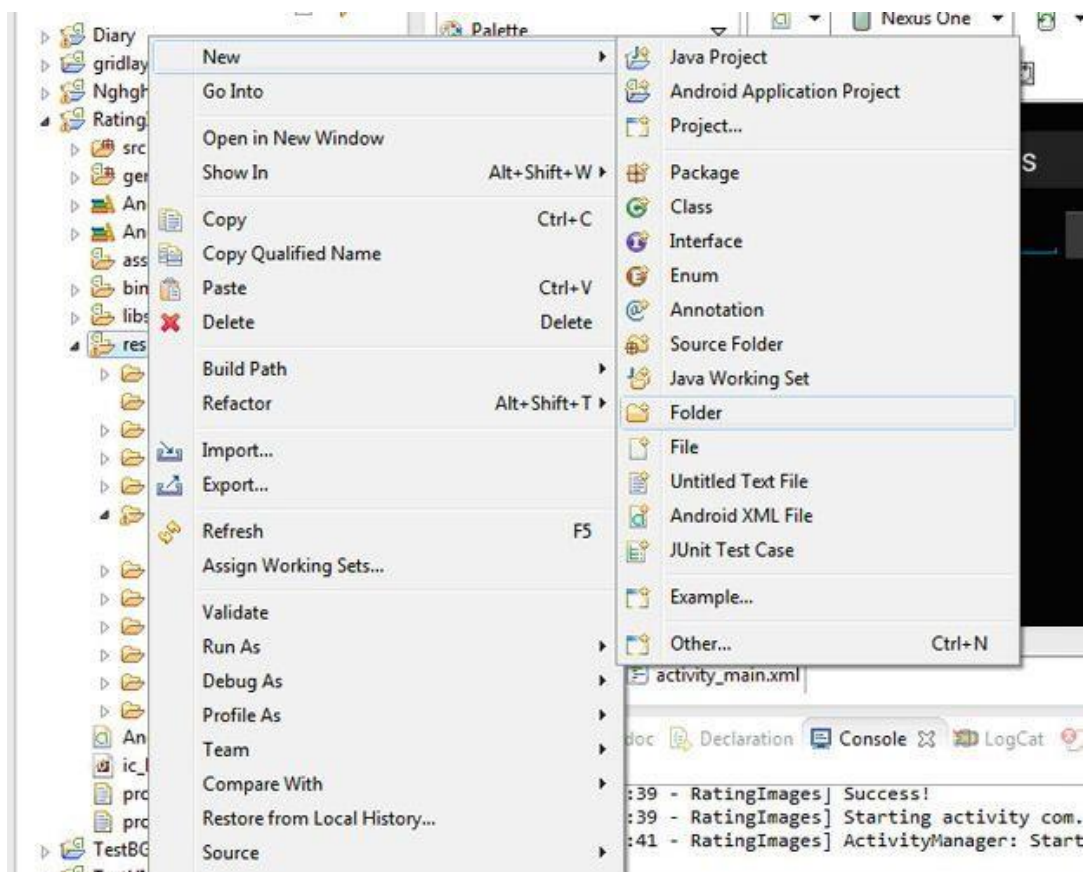


Рисунок 8.22 – Создание папки

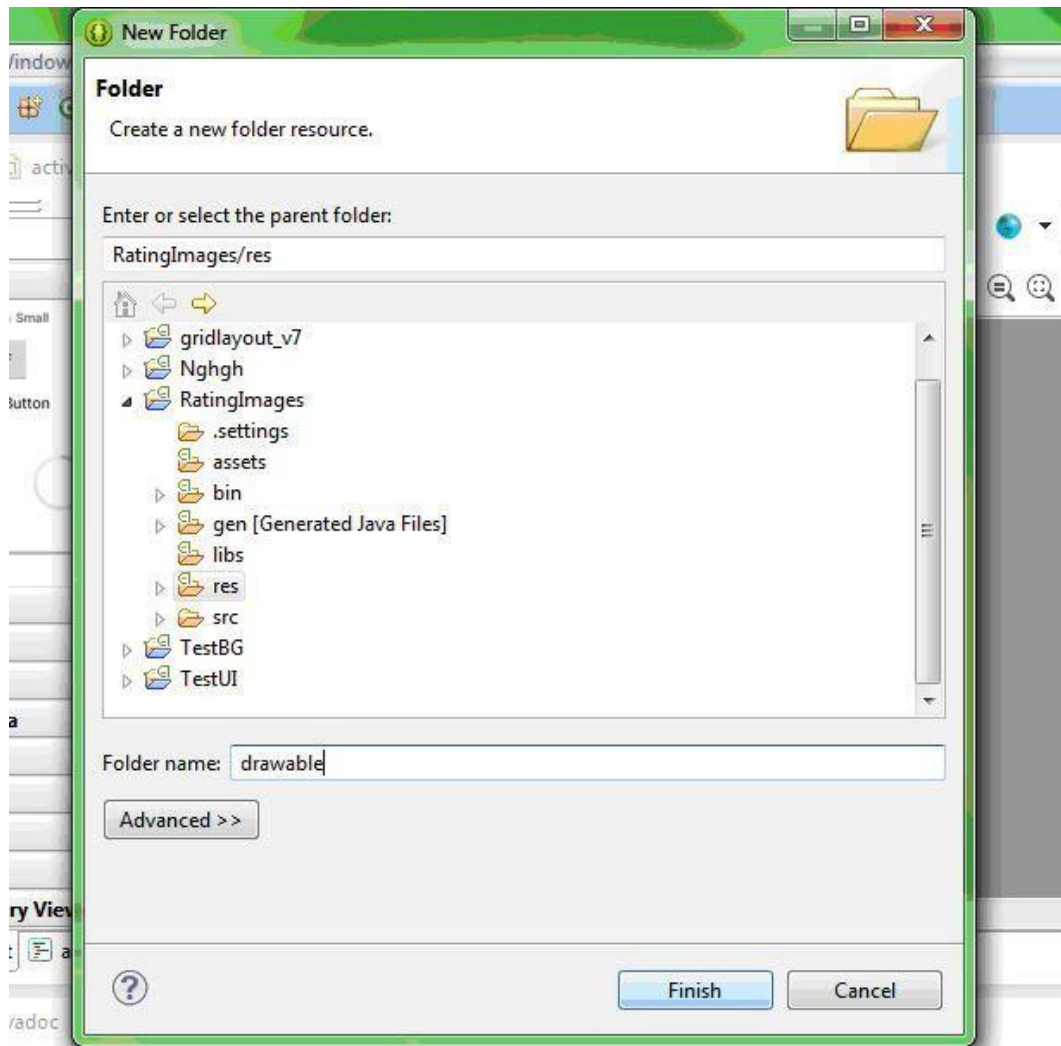


Рисунок 8.23 – Создание папки: имя папки

После того, как папка создана (рис. 8.24), нужно положить в эту папку изображение картинка называется **got.png**:

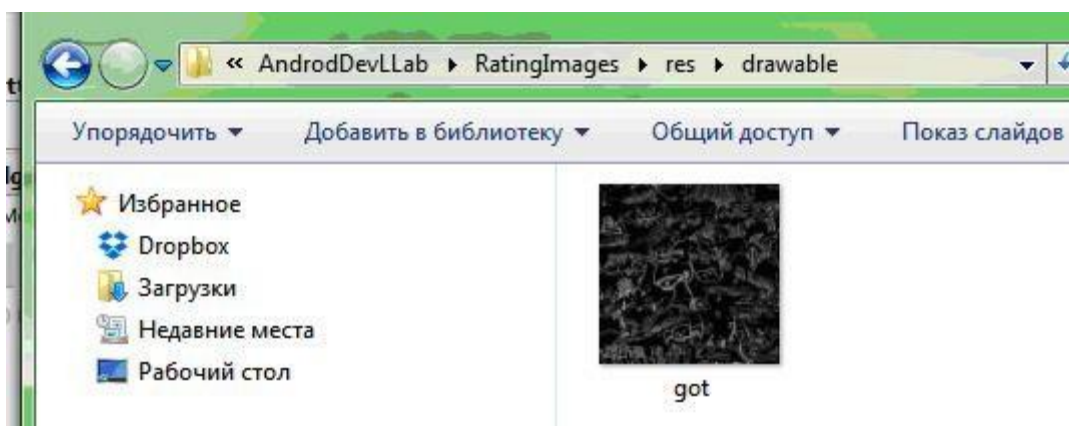


Рисунок 8.24 – Изображение в папке drawable/

После этого в папке **drawable/** нужно создать файл **background.xml**, важно

при создании выбрать параметр bitmap (рис. 8.25, 8.26).

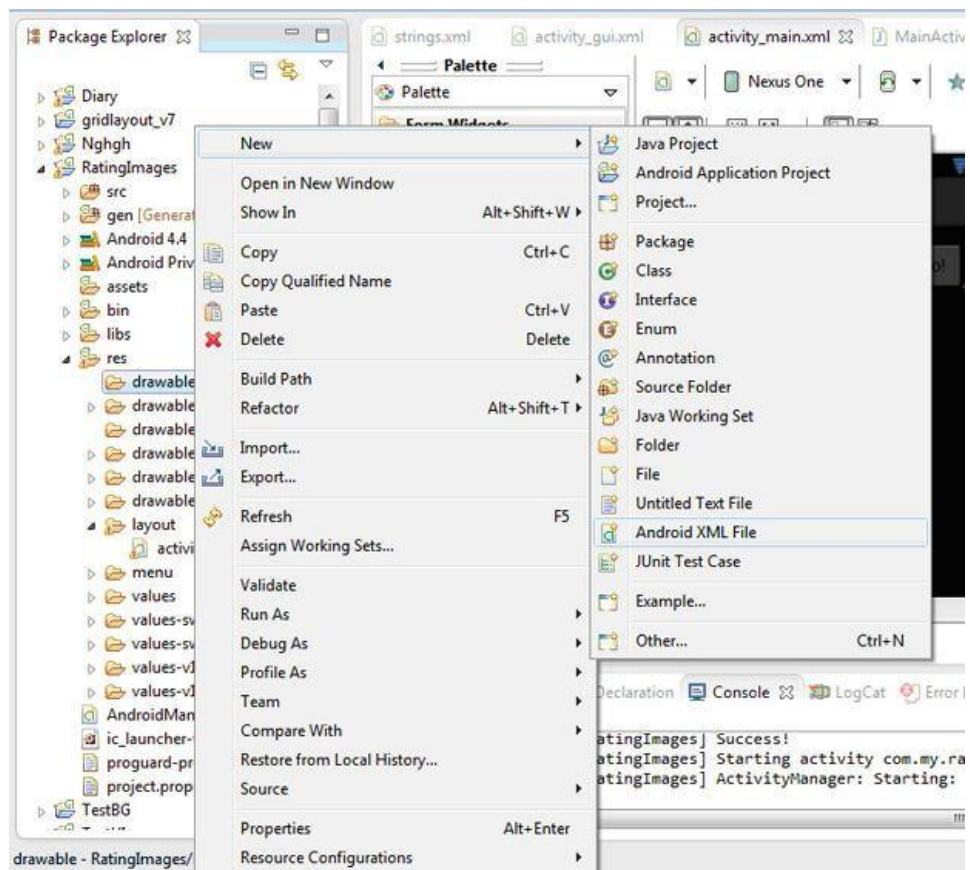


Рисунок 8.25 – Создание нового XML-файла

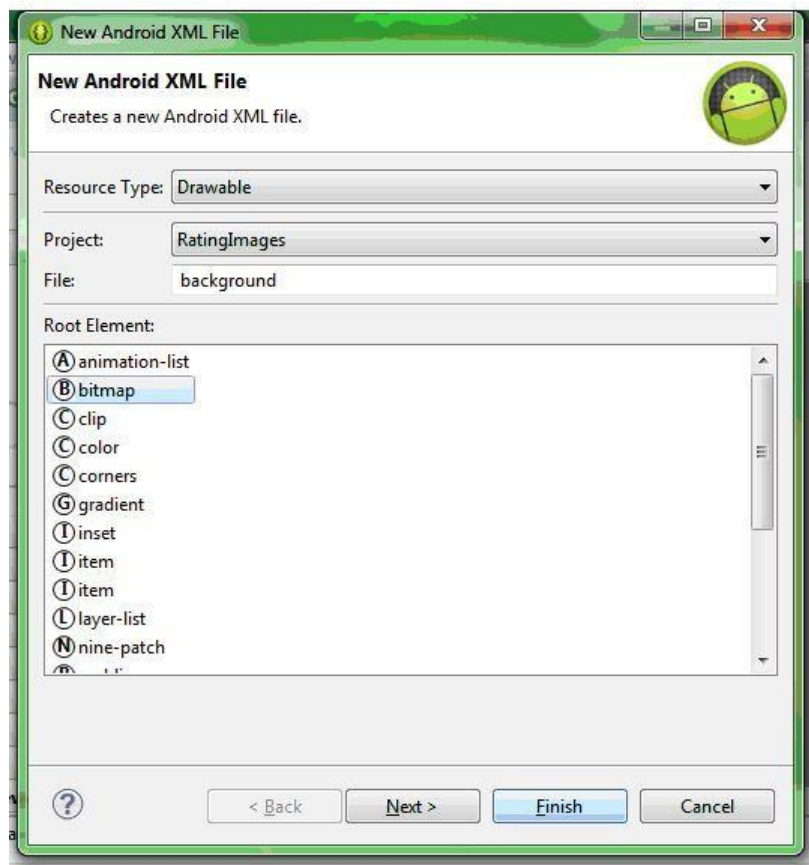


Рисунок 8.26 – Создание нового XML-файла

Как только новый файл открылся, пропишем в него одну строчку, с указанием на то, откуда и какой файл использовать:

```
<?xml version=«1.0» encoding=«utf-8»?>

<bitmap

xmlns:android=«http://schemas.android.com/apk/res/android
« android:src=«@drawable/got»>

</bitmap>
```

Вернемся в редактор XML-кода, туда, где прописывали цвет фона.

Вместо строки `android:background=«#000000»` напишем ссылку на XML-файл `android:background=«@drawable/background»`.

Сохраняем и видим результат (рис.8.27):

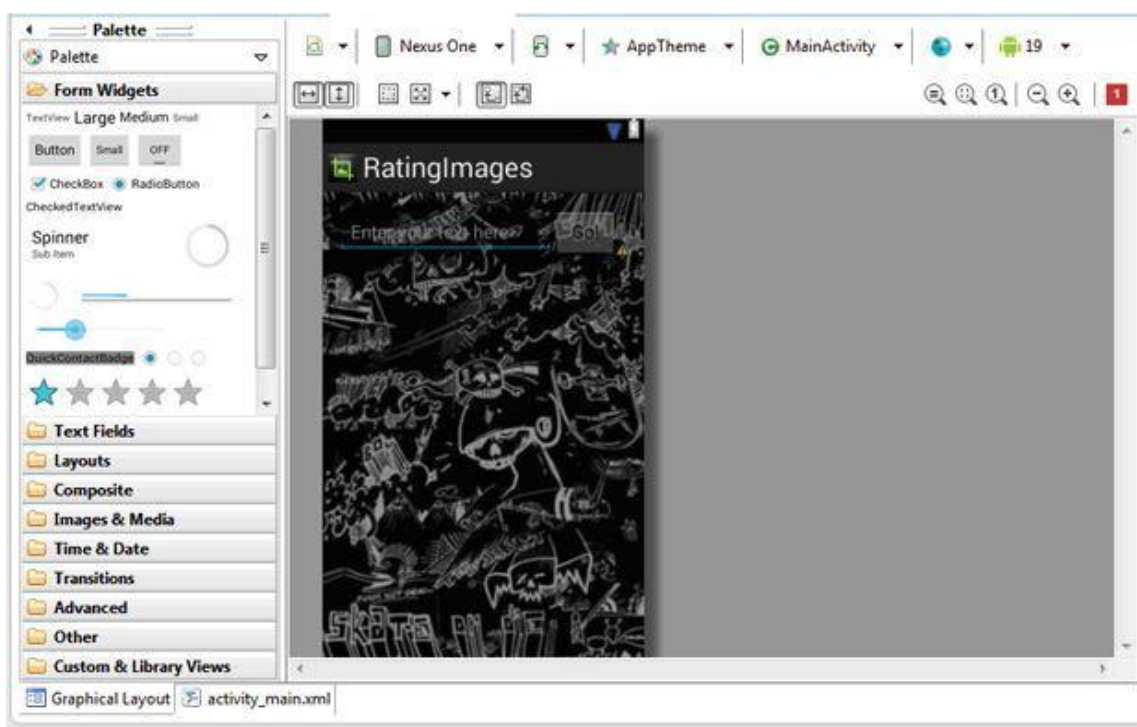


Рисунок 8.27 – Новый фон

Несомненно, фон смотрится хорошо, но очевидно, что кнопка и поле ввода просто затерялись, а это значит, что для этого приложения такой фон не подходит. Можно продолжить подбирать изображения на фон, но лучше создать черепичную заливку небольшим изображением.

Когда вы выбрали узор и скачали его, скопируйте изображение в папку **drawable/** (рис. 8.28, 8.29).

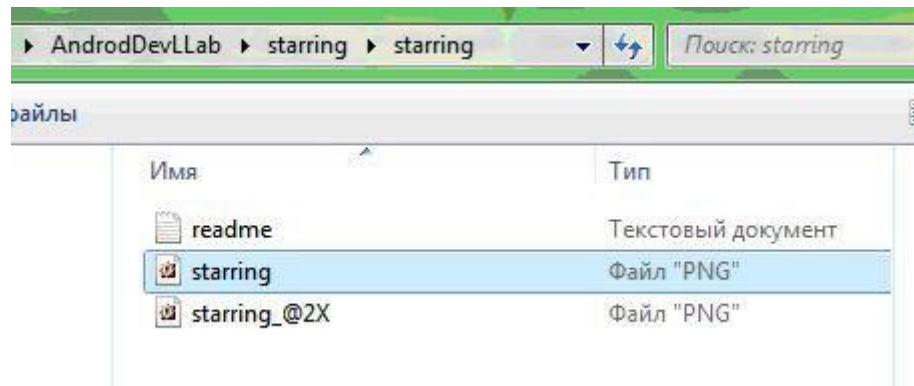


Рисунок 8.28 – Копирование изображения

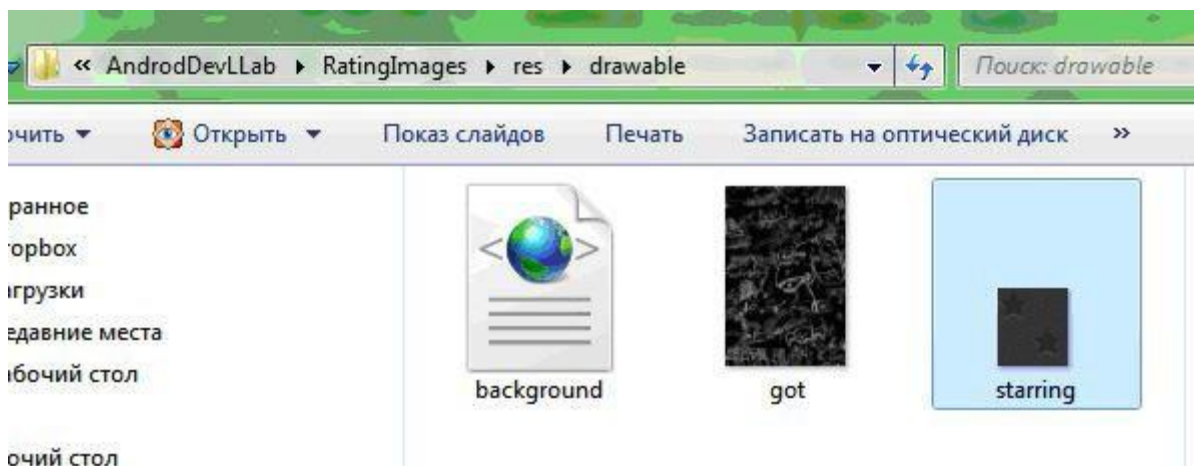


Рисунок 8.29 – Скопированное изображение

Теперь немного изменим файл **background.xml**.

Во-первых, нужно изменить имя изображения со старого на новое.

```
android:src="@drawable/starring"
```

Во-вторых, добавим такую строчку:

```
android:tileMode="repeat"
```

Сохраняем.

Атрибут `android:tileMode` задает тип заполнения, в данном случае простое повторение исходного изображения. Кроме `repeat` возможны варианты `clamp` и `mirror`. Помните, что данный приём применим только к `bitmap`, к фигурам, созданным при помощи XML, применить данную операцию нельзя.



Рисунок 8.30 – Варианты заполнения

Настало время посмотреть, что из этого получилось (рис. 8.31).

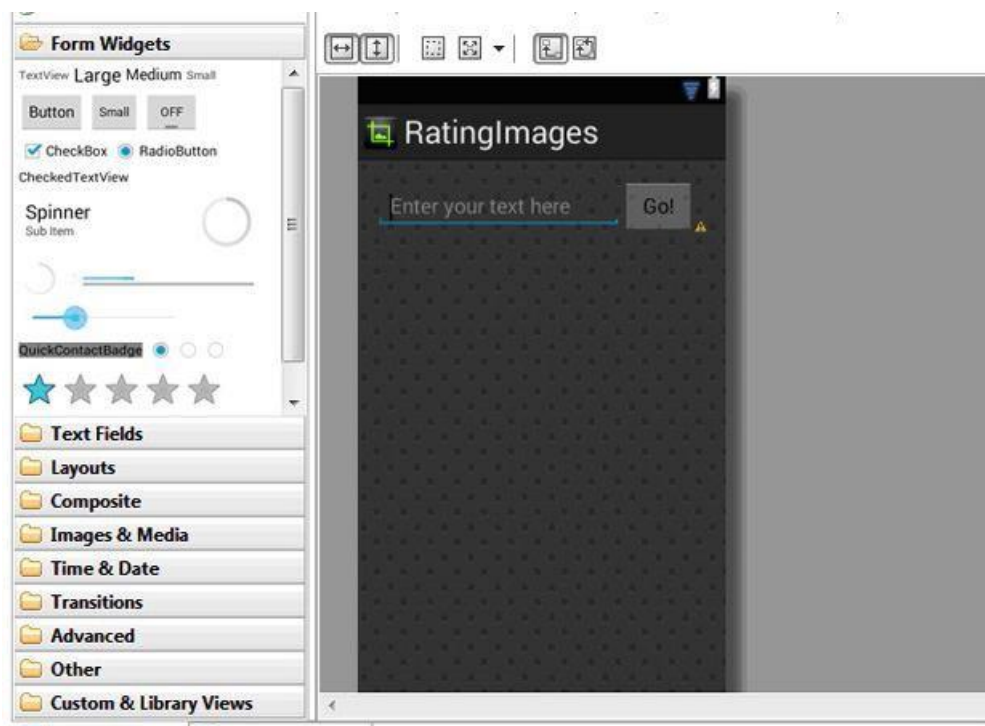


Рисунок 8.31 – Фон из звездочек

В `<RelativeLayout>` стоит добавлять `android:background` только в том случае, если вы хотите неподвижный фон, а в `<ScrollView>` чтобы фон прокручивался вместе с контентом. Если вы выбрали тёмный фон, то стоит поменять цвет текста, вводимого в поле ввода, например на белый.

Для этого в блок `<EditText>` добавим строку

```
android:textColor=«#ffffff»
```


1.5 Область просмотра изображений

Теперь займемся созданием области отображения изображений, которые пользователь будет оценивать.

Добавьте «рамку» <FrameLayout> в <RelativeLayout> (рис. 8.32):

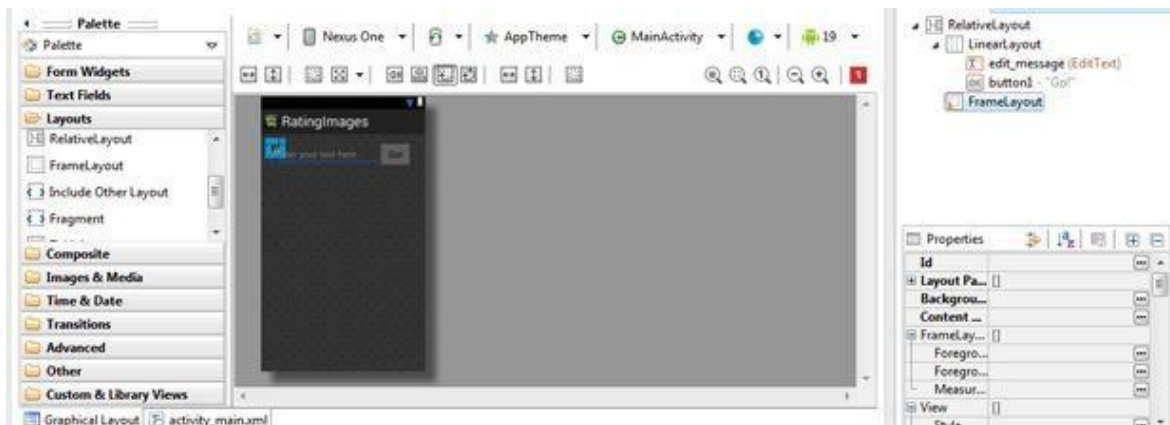


Рисунок 8.32 – Добавление «рамки»

Обратите внимание, что предупреждающий жёлтый треугольник исчез, но появился у нового элемента, и это значит, что всё идёт по плану.

Укажем этому элементу ширину, высоту и id (рис. 8.33). Предупреждение должно пропасть.

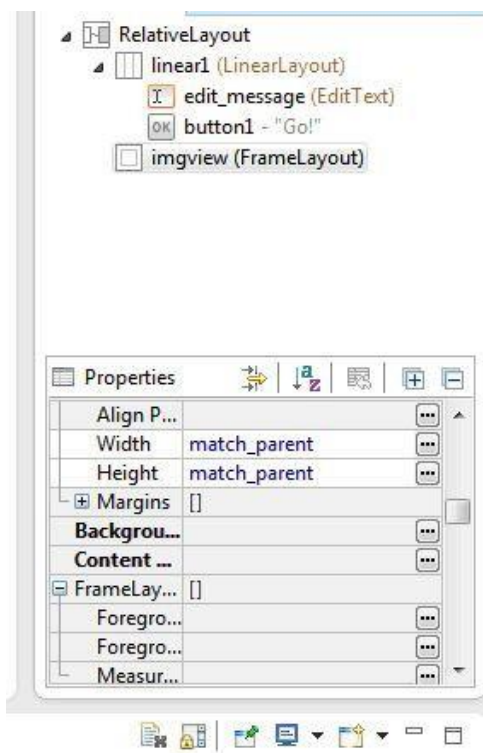


Рисунок 8.33 – Свойства «рамки»

Сейчас **FrameLayout** «заезжает» на поле ввода и кнопку. Чтобы исправить это, нужно задать для «рамки» параметр, который будет следить, чтобы «рамка» находилась ниже, чем поле ввода и кнопка.

Выберите из списка **Outline** «рамку» и кликом правой кнопки мыши вызовите контекстное меню (рис. 8.34): **Other Properties -> Layout Parametrs -> Layout Below...**

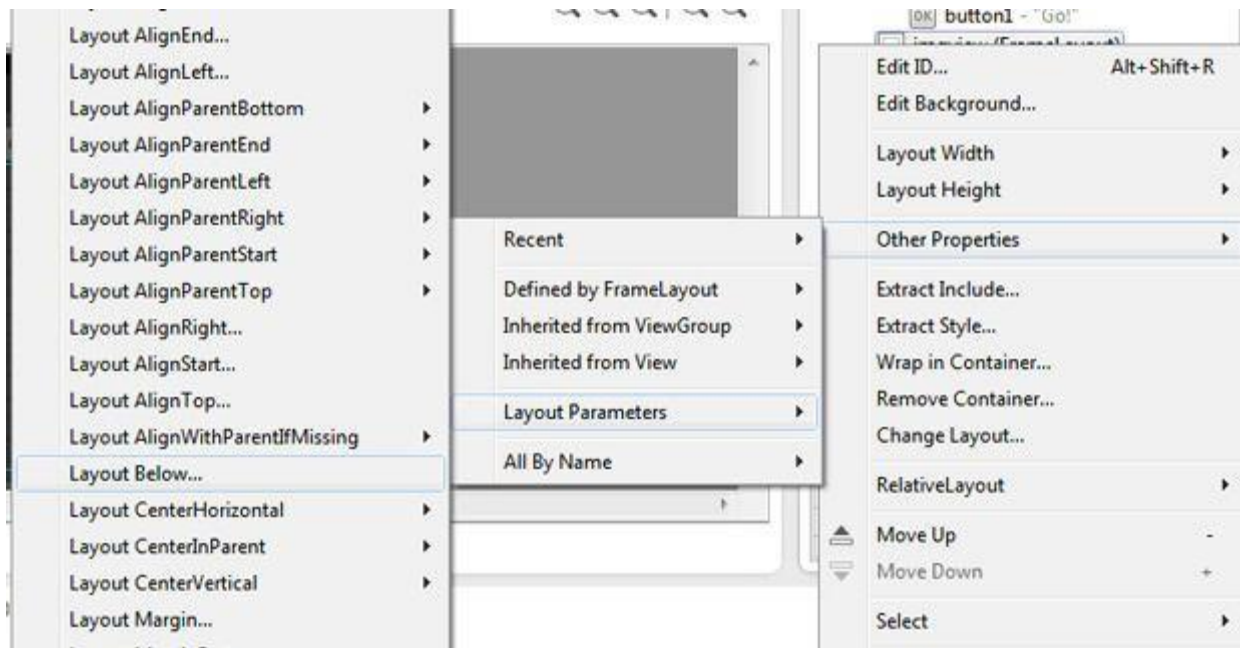


Рисунок 8.34 – Контекстное меню «рамки»

Появится окно. В нём выбираем из списка «ID» имя лейаута, на котором находятся поле ввода и кнопка (рис. 8.35):

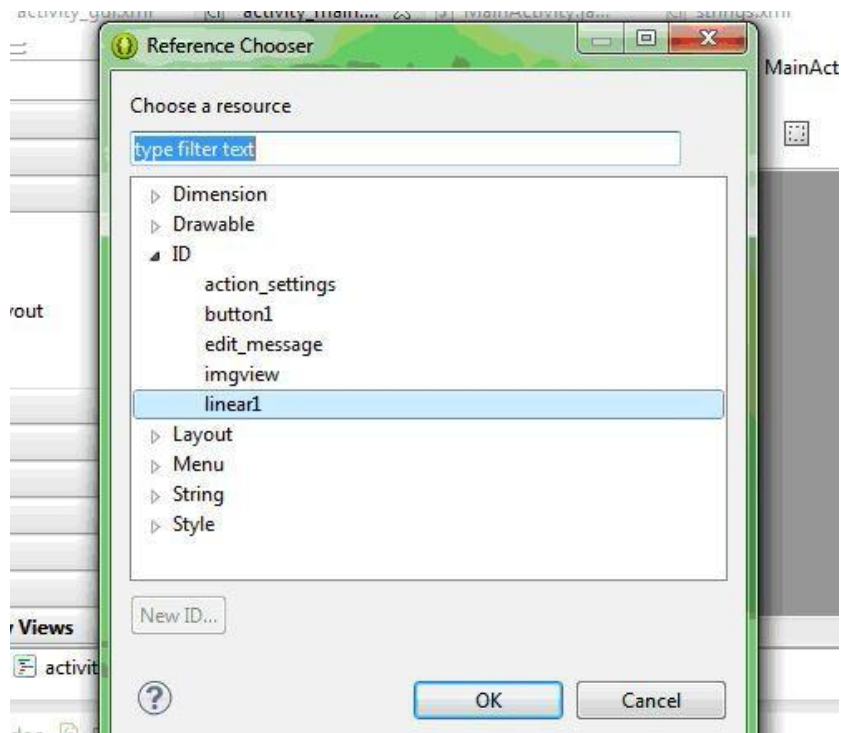


Рисунок 8.35 – Выбор элемента, ниже которого должна быть «рамка»

Жмём **ОК** и «рамка» примет вид (рис. 8.36):

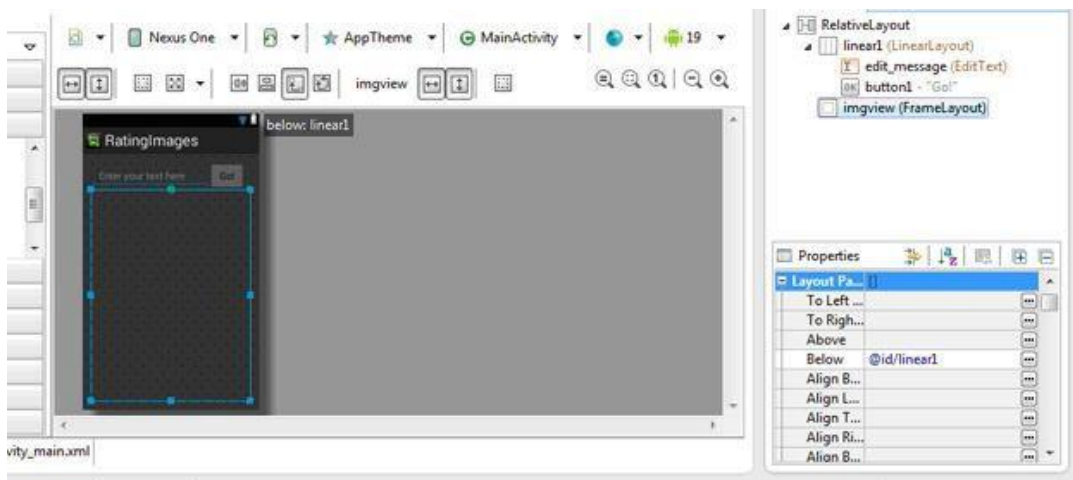


Рисунок 8.36 – Новое расположение «рамки» Теперь для наглядности поместим туда изображение.

Сначала поместим само изображение в папку **res/drawable/** и обновим её (рис. 8.37). Это нужно для того, чтобы новые данные загрузились в проект.

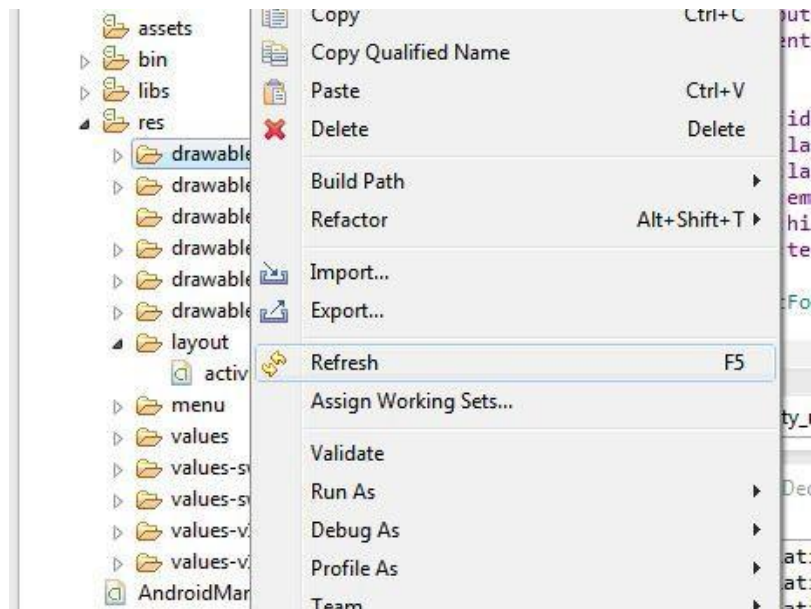


Рисунок 8.37 – Обновление папки

Теперь поместим внутрь «рамки» <ImageView> (рис. 8.38)

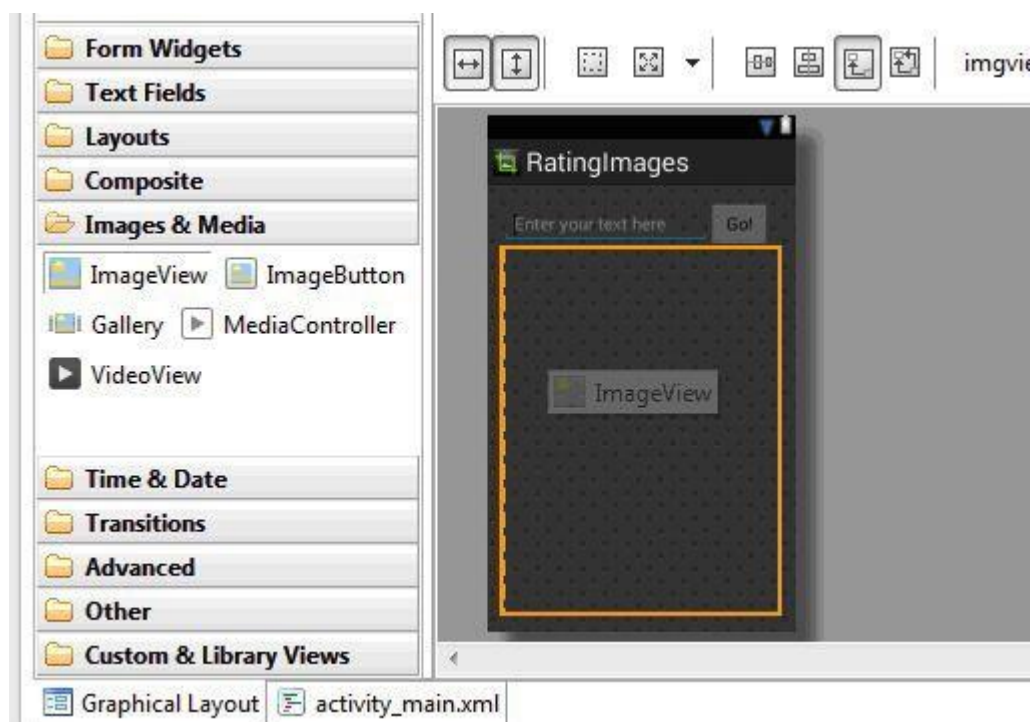


Рисунок 8.38 – Перемещение <ImageView>

В появившемся окне выбираем нужное нам изображение и жмём **ОК** (рис. 8.39).

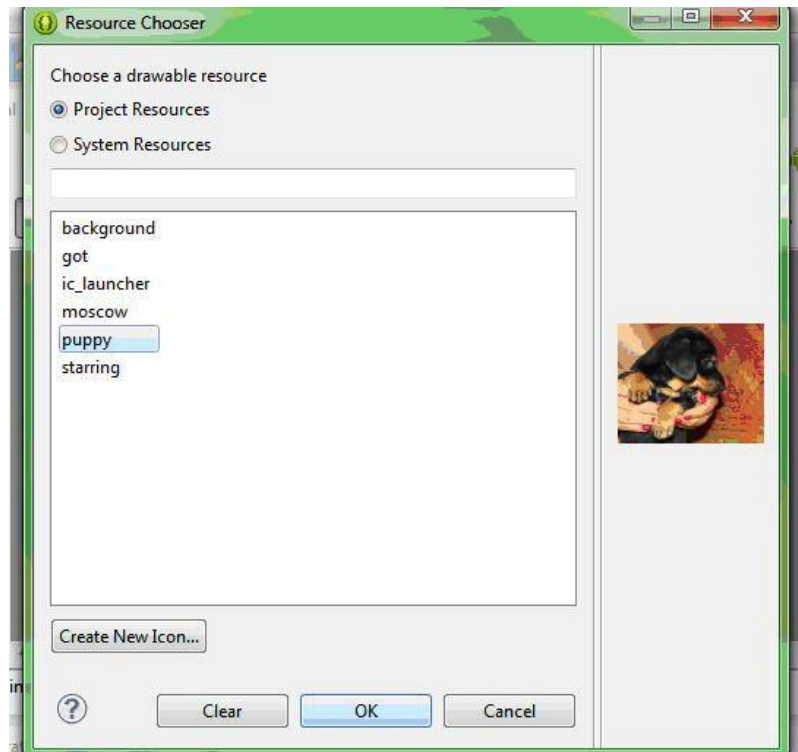


Рисунок 8.39 – Выбор файла

Изображение должно появиться на активности (рис. 8.40):

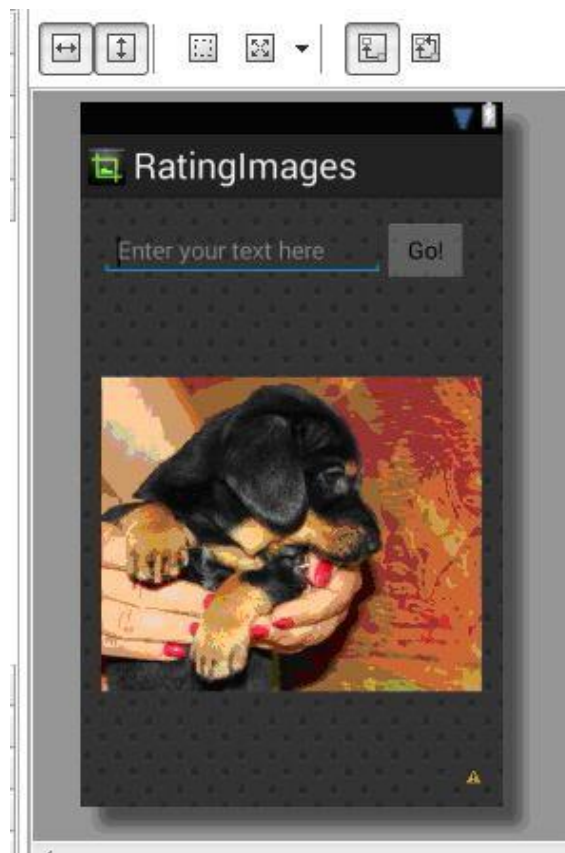


Рисунок 8.40 – Визуализация изображения

Вместе с изображением появился новый предупреждающий знак.

В описании предупреждения сказано: «**[Accessibility] Missing contentDescription attribute on image**». Давайте разберёмся, что это означает.

Атрибут `android:contentDescription` используется в программах, которые поддерживают специальные возможности для людей с нарушениями зрения и слуха. То есть, текст, прописанный в этом атрибуте, не будет показан на экране, но при включенной в «Специальных возможностях» услуге «звуковые подсказки» этот текст будет проговариваться, когда пользователь переходит к этому элементу управления.

Этот атрибут можно задать для `ImageButton`, `ImageView` и `CheckBox`, и задавать его или нет дело каждого.

Свойства у изображения должны быть следующими (рис. 8.41):

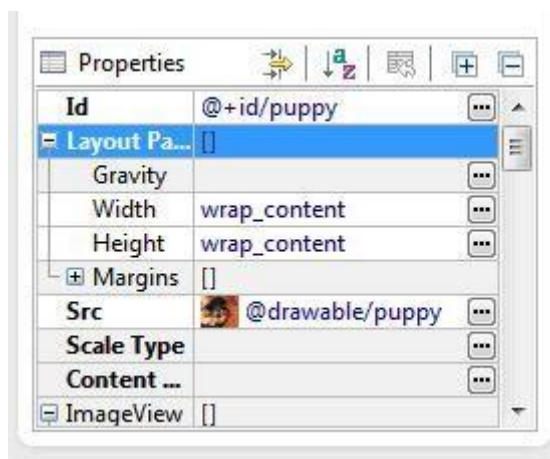


Рисунок 8.41 – Свойства изображения

1.6 Кнопки «like» и «dislike»

Пришло время создать кнопки оценивания.

Для этого добавьте на форму `<RelativeLayout>`, задайте для него ширину и высоту `wrap_content`, и укажите `id`.

Снова в папку **res/drawable/** нужно добавить файлы. Найдите изображения «Палец вверх» и «Палец вниз», и поместите их в эту папку, после чего обновите её.

Добавьте `<ImageButton>`, выберите изображение «Палец вверх», и переместите `<RelativeLayout>` в такое положение (рис. 8.42):

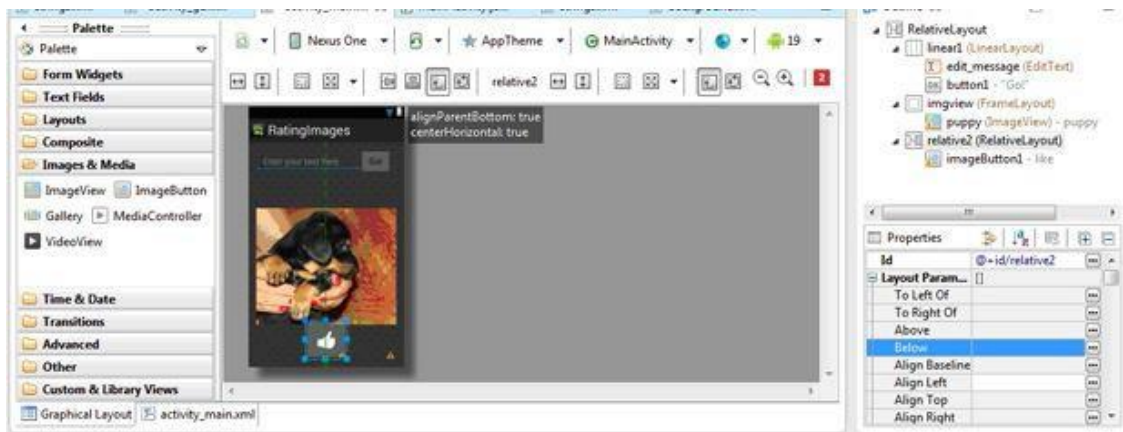


Рисунок 8.42 – Новая кнопка

Укажите «рамке» быть выше, чем поле с кнопкой оценки (рис. 8.43, 8.44):

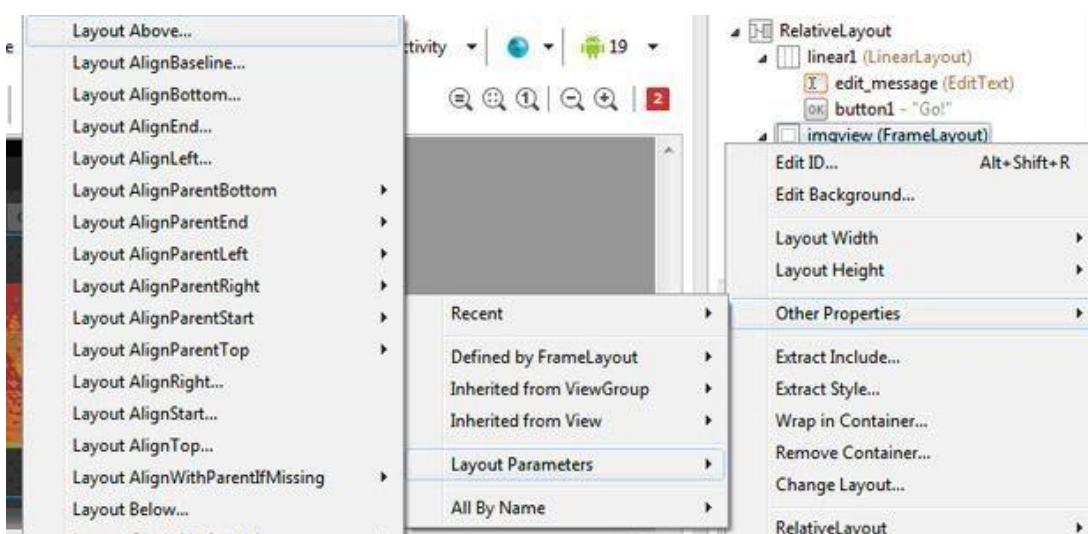


Рисунок 8.43 – Контекстное меню «рамки»

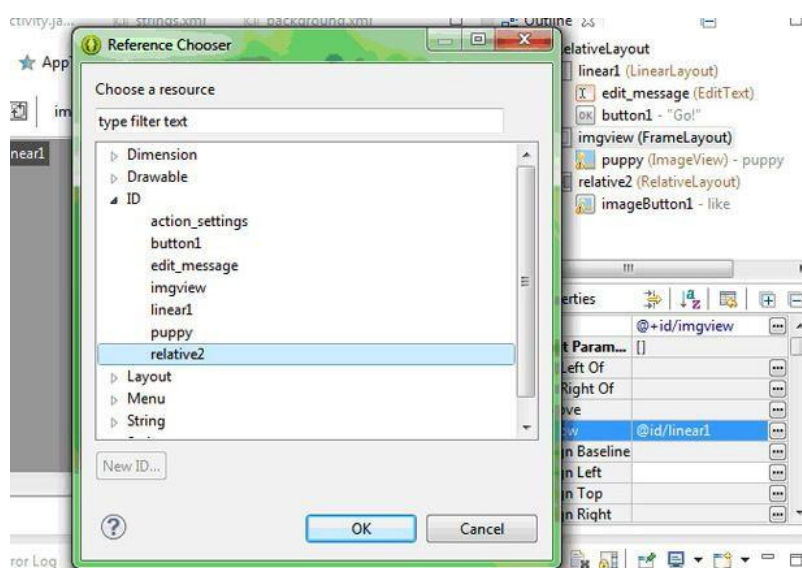


Рисунок 8.44 – Выбор элемента, выше которого должна быть «рамка»

В результате должно получиться следующее (рис. 8.45):

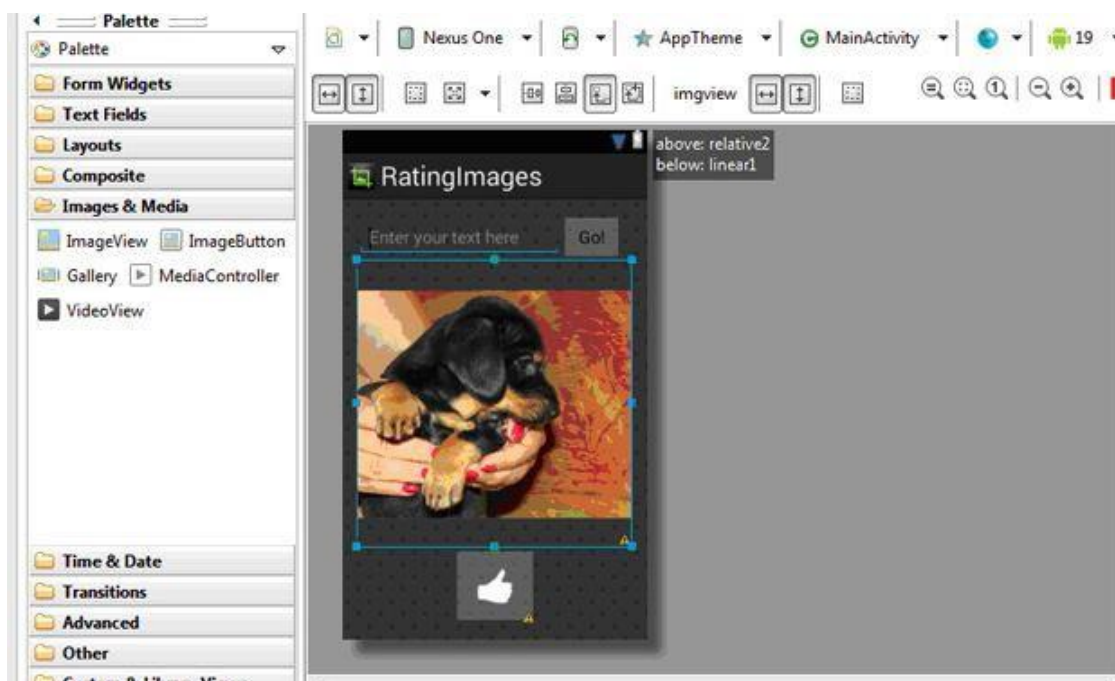


Рисунок 8.45 – Новое расположение элементов

Добавьте еще одну кнопку кнопку «**Палец вниз**». Она «наложилась» на первую кнопку. Чтобы это исправить, сделайте с `<RelativeLayout>` то же самое, что и с `<LinearLayout>`: растяните элемент влево и вправо, до получения такого результата (рис. 8.46):



Рисунок 8.46 – Выравнивание области с кнопками

Теперь расставьте кнопки по краям так, чтобы они «прикрепились» к

краям (рис. 8.47).

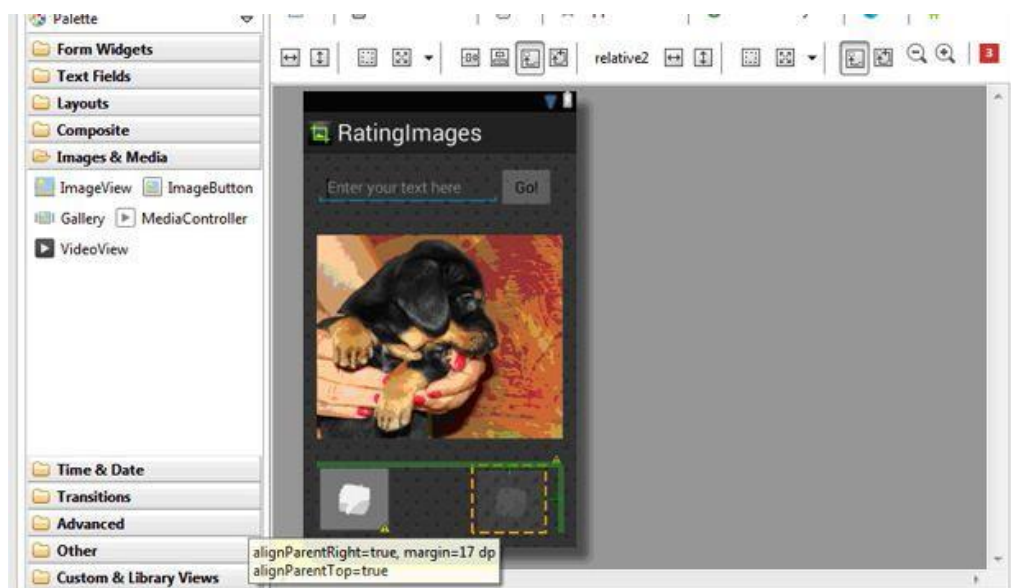


Рисунок 8.47 – Выравнивание кнопок

Готово! Теперь можно запустить эмулятор и посмотреть, что получилось (рис. 8.48).

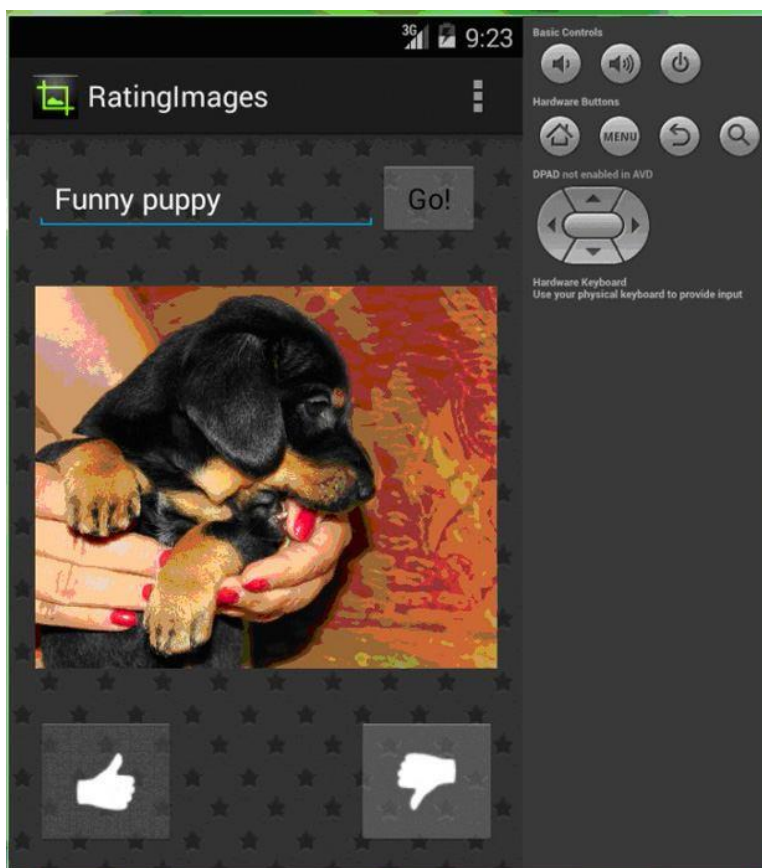


Рисунок 8.48 – Главная активность

1.7 Листинги

1)

```
<RelativeLayout
    xmlns:android=«http://schemas.android.com/apk/res/android»
    xmlns:tools=«http://schemas.android.com/tools»
    android:layout_width=«match_parent»
    android:layout_height=«match_parent»
    android:paddingBottom=«@dimen/activity_vertical_margin»
    android:paddingLeft=«@dimen/activity_horizontal_margin»
    android:paddingRight=«@dimen/activity_horizontal_margin»
    android:paddingTop=«@dimen/activity_vertical_margin»
    tools:context=«.MainActivity»
    android:background=«@drawable/background»>
    <LinearLayout
        android:id=«@+id/linear1»
        android:layout_width=«wrap_content»
        android:layout_height=«wrap_content»
        android:layout_alignParentLeft=«true»
        android:layout_alignParentRight=«true»
        android:orientation=«horizontal» >
        <EditText
            android:id=«@+id/edit_message»
            android:layout_width=«wrap_content»
            android:layout_height=«wrap_content»          android:ems=«10»
            android:hint=«@string/edit_message»
            android:textColor=«#ffffff» >
            <requestFocus />
        </EditText>
        <Button
        />
    </LinearLayout>
```

```

        android:id=«@+id/button1»
    android:layout_width=«wrap_content»
    android:layout_height=«wrap_content»
    android:text=«@string/button_send»

    <FrameLayout
        android:id=«@+id/imgview»
        android:layout_width=«match_parent»
        android:layout_height=«match_parent»
    android:layout_above=«@+id/relative2»
    android:layout_below=«@id/linear1» >
        <ImageView
            android:id=«@+id/puppy»
            android:layout_width=«wrap_content»
            android:layout_height=«match_parent»
            android:src=«@drawable/puppy»
        />
    </FrameLayout>

    <RelativeLayout
        android:id=«@+id/relative2»
        android:layout_width=«wrap_content»
        android:layout_height=«wrap_content»
        android:layout_alignLeft=«@+id/imgview»
    android:layout_alignParentBottom=«true»
    android:layout_alignRight=«@+id/imgview» >
        <ImageButton
            android:id=«@+id/imageButton2»
            android:layout_width=«wrap_content»
            android:layout_height=«wrap_content»
            android:layout_alignParentRight=«true»
            android:layout_alignParentTop=«true»

```



```

android:src=«@drawable/dislike»
    />
    <ImageButton
        android:id=«@+id/imageButton1»
        android:layout_width=«wrap_content»
        android:layout_height=«wrap_content»
        android:layout_alignParentLeft=«true»
        android:layout_alignParentTop=«true»
        android:src=«@drawable/like»
    />
</RelativeLayout>

```

```

</RelativeLayout>

```

2)

```

<?xml version=«1.0» encoding=«utf-8»?>
<resources>
    <string name=«app_name»>RatingImages</string>
    <string name=«action_settings»>Settings</string>
    <string name=«hello_world»>Hello world!</string>
    <string      name=«edit_message»>Enter      your      text
here</string>
    <string name=«button_send»>Go!</string>
</resources>

```

3)

```

<?xml version=«1.0» encoding=«utf-8»?>
    <bitmap
        xmlns:android=«http://schemas.android.com/apk/res/and
roid» android:src=«@drawable/starring»
        android:tileMode=«repeat» >

```

</bitmap>

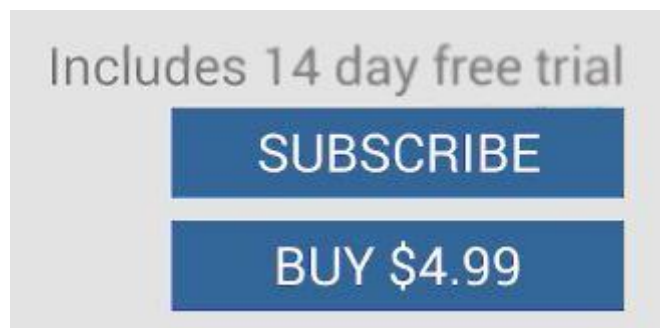
Задание 2. BuildingBlocks или элементы для построения интерфейса

Среда разработки «**AndroidIDE**» открывает широкий выбор готовых к использованию элементов для создания выдающихся приложений.

Кнопка (Button) состоит из текста и/или изображения, которые дают понять пользователю, что произойдёт, если нажать на эту кнопку. Важно помнить, что человек по своей природе привык взаимодействовать с объектами, поэтому фон у кнопки абсолютно не обязателен.



a)



b)

Рисунок 8.49 – Кнопка: а – кнопка-изображение; b – кнопка-текст

Текстовые поля (TextFields)

Текстовые поля позволяют пользователю вводить текст в приложения. Они могут быть однострочными или многострочными. Одно касание текстового поля помещает курсор на поле ввода и автоматически выводит на экран клавиатуру. В дополнение к набору текста на клавиатуре текстовые поля позволяют выделять текст (вырезать, копировать, вставить).

Поиск вариантов завершения слова помогает правописанию слов и упрощает поиск контактов в списке (рис. 8.50, 8.51).

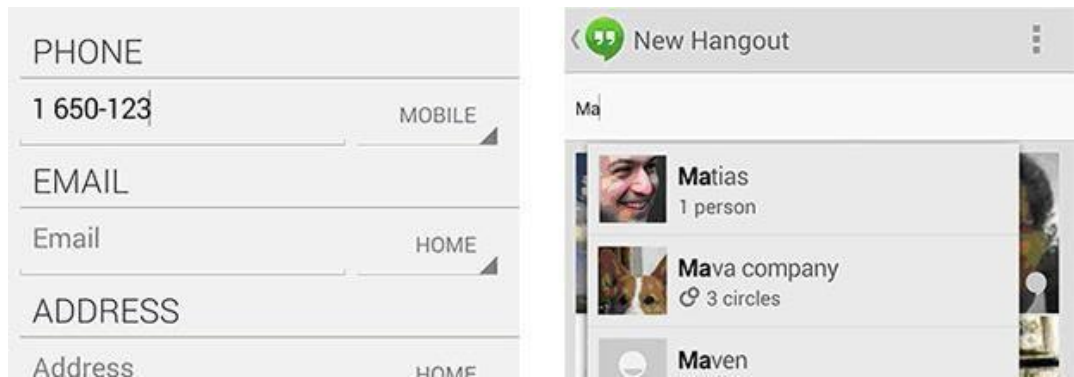


Рисунок 8.50 – Текстовые поля, поиск контактов

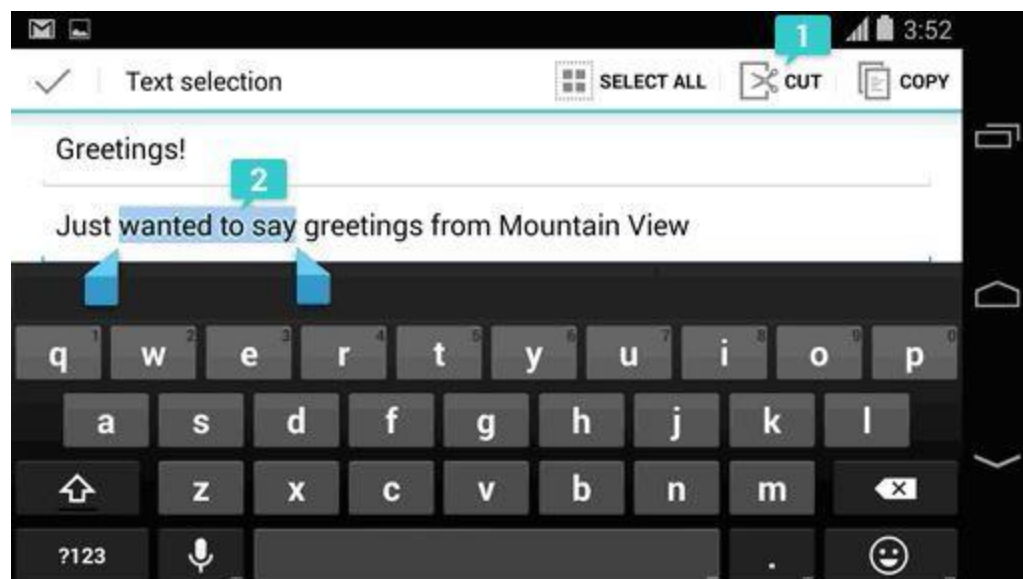


Рисунок 8.51 – Выделение текста

Ползунки и слайдеры (Seek Bars and Sliders)

Интерактивные ползунки позволяют выбирать значение из непрерывного или дискретного диапазона значений путем перемещения ползунка. Наименьшее значение находится слева, наибольшее справа (рис. 8.52).

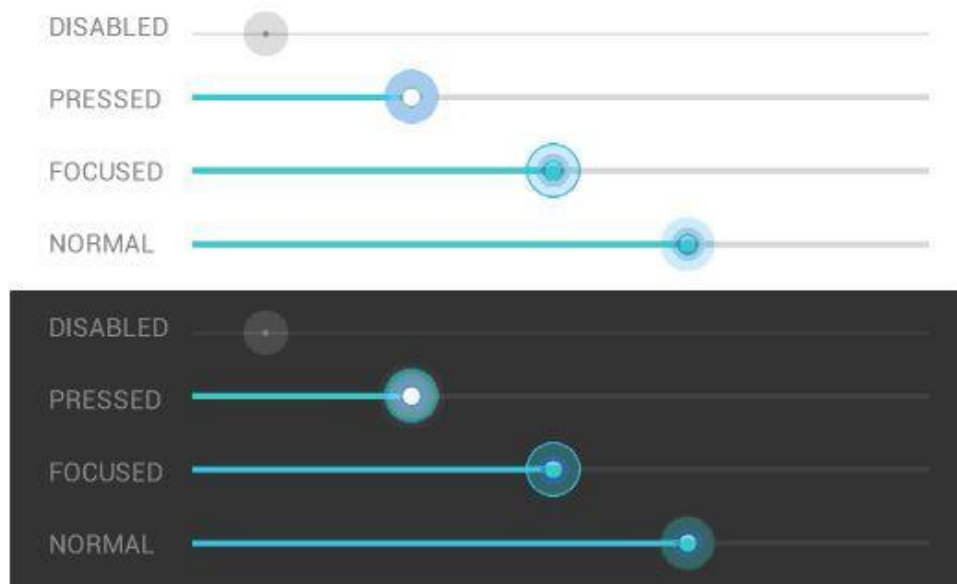


Рисунок 8.52 – Ползунки в светлой и темной темах

Интерактивный характер слайдера делает его удобным для настроек, которые отражают уровни интенсивности, такие как громкость, яркость, или насыщенность цвета (рис. 8.53).

Так, например, уровень громкости можно регулировать кнопками на корпусе устройства или при помощи жеста пальцем по экрану.



Рисунок 8.53 – Настройка звука

Прогресс-бары и активности (Progress&Activity)

Прогресс-бары и показатели активности сигнализируют пользователям о происходящем в данный момент времени длительном действии, что означает для

пользователя подождать завершения процесса некоторое время (рис. 8.54).



Рисунок 8.54 – Процесс загрузки приложения

Переключатели (Switches) позволяют пользователю выбирать параметры.

Есть три вида переключателей: флажки, радио-кнопки, и включение/выключение выключателей.

Флажки используются в том случае, если из предлагаемого списка можно выбрать одновременно несколько вариантов (рис. 8.55).

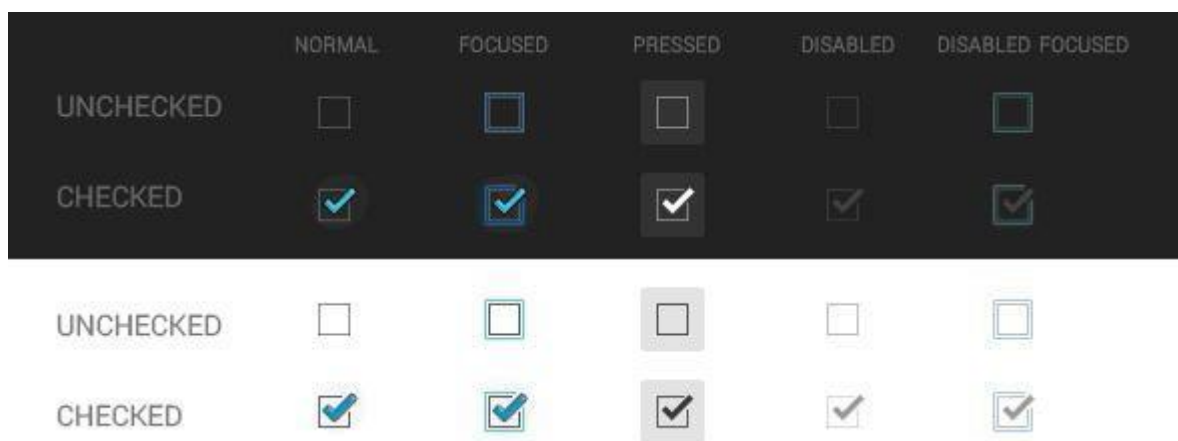


Рисунок 8.55 – Флажки

Радио-кнопки позволяют выбрать только один вариант из списка. Радио-кнопки формируются в группы (рис. 8.56).

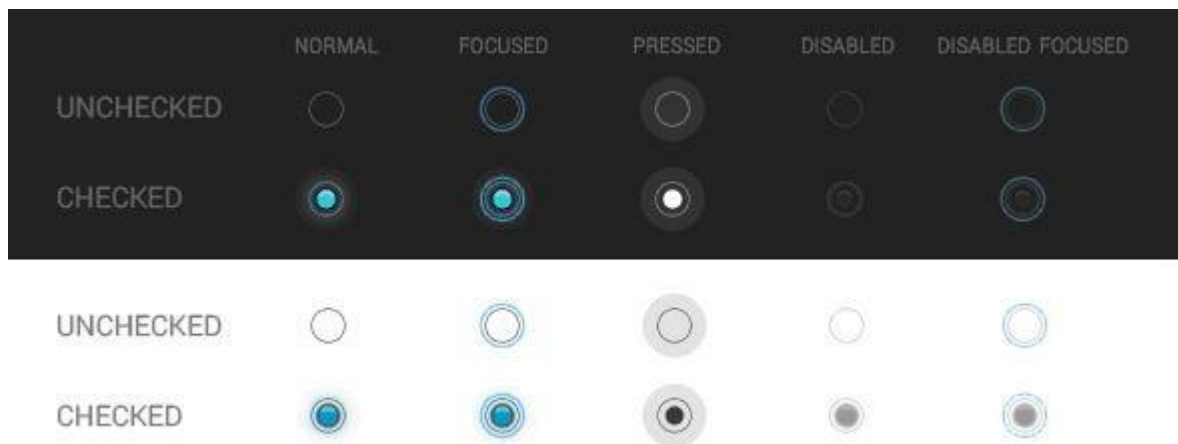


Рисунок 8.56 – Радио-кнопки

Выключатели дают возможность сделать флажок более наглядным, применив в качестве основы кнопку-значок, которая может фиксироваться в нажатом состоянии (рис. 8.57).



Рисунок 8.57 – Выключатели

Диалоговые окна (Dialogs) помогают приложению взаимодействовать с пользователем. Это может быть как банальный вопрос с двумя вариантами ответа (ОК и Отмена), так и сложные окна с множеством настраиваемых пользователем параметров (рис. 8.58).

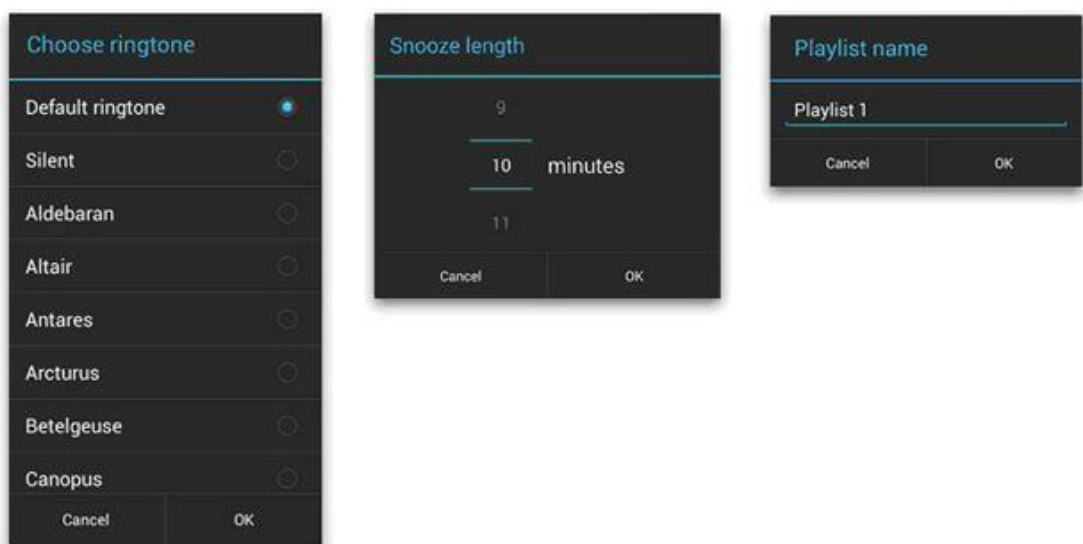


Рисунок 8.58 – Примеры диалоговых окон

Средство выбора (Pickers) это простой способ выбрать одно значение из набора путём последовательного перебора. Удобный перебор предлагаемых значений при помощи клика по стрелочкам или прокрутки, так же можно ввести значение с клавиатуры (рис. 8.59).

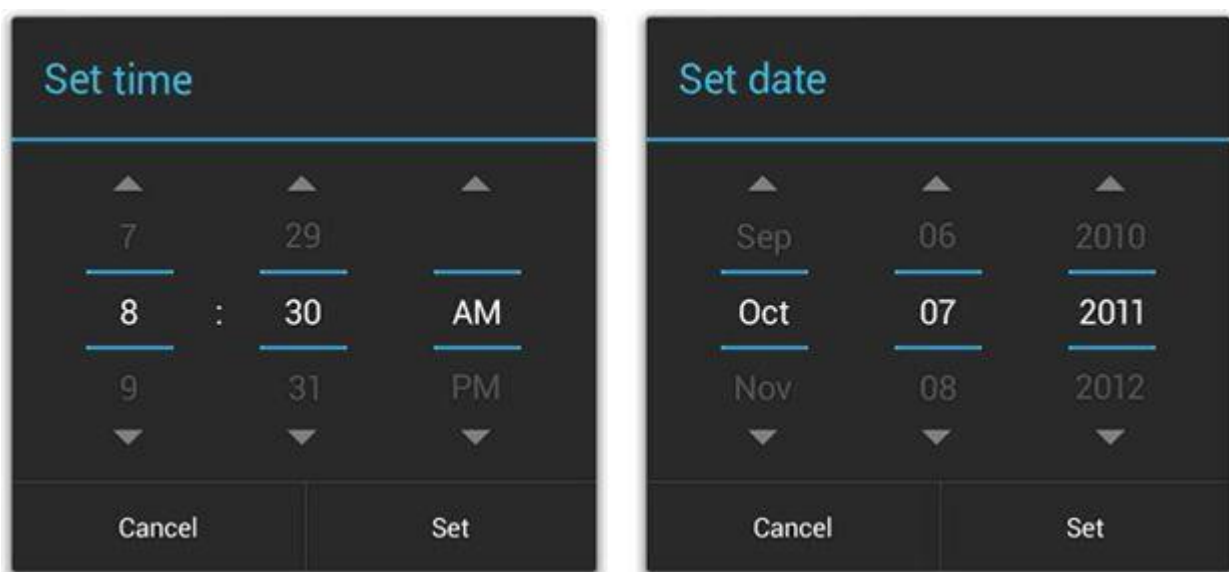


Рисунок 8.59 – Примеры пикеров

Прокрутка (Scrolling), как интуитивно ожидаемый элемент, уже давно плотно сидит в головах у разработчиков и пользователей. Плавное или быстрое перемещение по содержимому экрана и контенту, который «вылез» за рамки экрана, осуществляется прокруткой, достаточно лишь провести пальцем по

экрану в нужную сторону с желаемой скоростью. Для того чтобы индикатор прокрутки не занимал место у рабочей области экрана, после приостановки пользования прокруткой, индикатор исчезает, и появляется снова, стоит только начать «прокручивать» пальцем. Для удобства поиска чего-либо в алфавитном списке, рядом с индикатором прокрутки возникает индекс с указанием на то, в каком разделе находится индикатор (рис. 8.60).

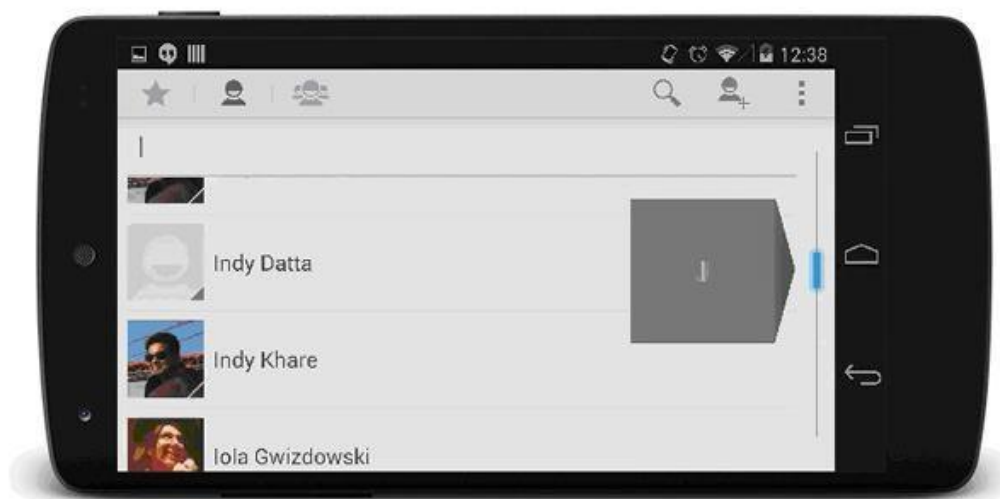


Рисунок 8.60 – Индикатор и индекс прокрутки

Выпадающий список (Spinner) обеспечивает удобный способ выбора одного значения из набора (рис. 8.61).

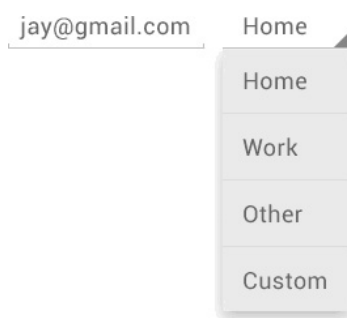


Рисунок 8.61 – Спиннер для выбора типа e-mail адреса

Вкладки (Tabs) позволяют легко просматривать и переключаться между различными окнами или функциональными элементами приложения. Вкладки, прокручиваемые движением пальца (ScrollableTabs), делают этот процесс еще проще и приятнее (рис. 8.62).

Основные вкладки (FixedTabs) удобны при отображении трех и менее вкладок, поскольку ширина вкладок фиксируется по самому длинному названию вкладки.



Рисунок 8.62 – Вкладки

Списки (Lists) удобны для представления нескольких позиций информации в вертикальном расположении (рис. 8.63).

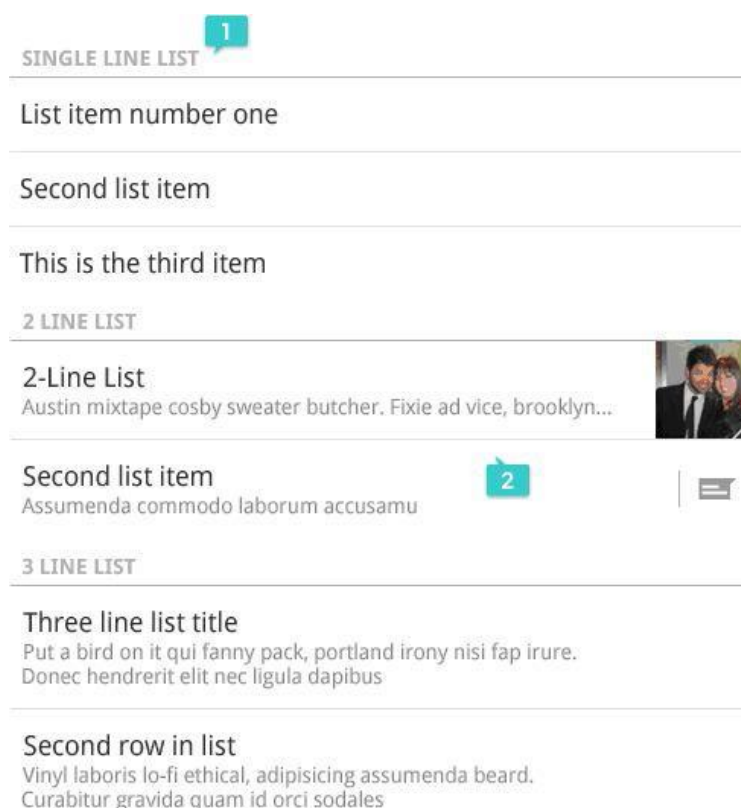


Рисунок 8.63 – Список

Сетки (GridLists) это альтернатива стандартного списка. С помощью сетки можно создавать более удобное представление образных данных, нежели с помощью списка. Ещё одним плюсом сетки является то, что прокрутку можно осуществлять не только в вертикальном, но и в горизонтальном направлении.

Однако нужно стараться избегать создания сетки с возможностью прокрутки в обоих измерениях сразу.

Использование ярлычков для всех панелей сетки и полупрозрачных

«занавесок» для неактивных панелей помогает пользователю быстрее понять, что скрывается под той или иной панелью (рис. 8.64).

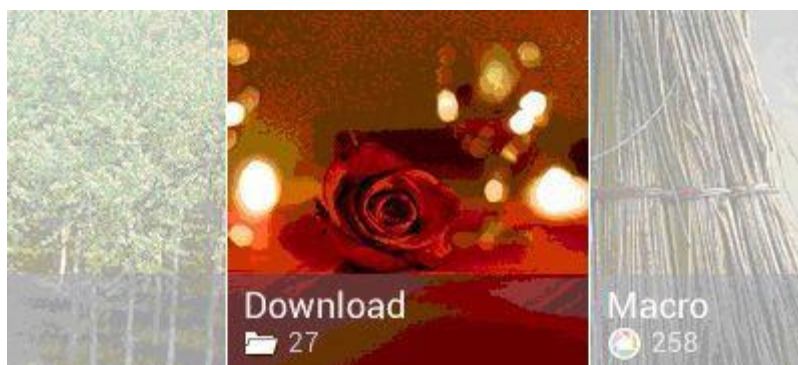


Рисунок 8.64 – Сетка с альбомами

Содержание отчета и его форма

Отчет к лабораторной работе оформляется в виде текстового документа по форме простого реферата и должен включать:

1. Название лабораторной работы.
2. Цель лабораторной работы.
3. Краткие выводы по результатам выполнения лабораторной работы согласно пунктам раздела «Методики выполнения работы».
4. Ответы на Вопросы к лабораторной работе.
5. Формулировку задания для самостоятельной работы и результат его выполнения.

Задания для самостоятельной работы

1. Создайте в приложении на главной активности строку, в которой будет выводиться адрес сайта, откуда загружено изображение, и кнопку для перехода на этот сайт.
2. Подумайте над интерфейсом собственного приложения. Какие элементы управления оно может содержать? Попробуйте воплотить его интерфейс на практике.

Вопросы к лабораторной работе

1. Технология создания прототипа интерфейса.
2. Характеристика элемента <LinearLayout>.

3. Характеристика кнопок «like» и «dislike».
4. Характеристика элементов для построения интерфейса.

Лабораторная работа 9

Создание многоэкранного приложения

Цель работы: научиться создавать приложения, состоящие из нескольких активностей, и диалоговые окна, а также познакомиться с элементами тач-интерфейса

Теоретическое обоснование

Данная лабораторная работа посвящена различным способам разработки многоэкранных приложений. Предлагается разработать три приложения, каждое из которых посвящено отдельному аспекту этой тематики. По завершению работы над приложениями студенты смогут разработать собственные программы с использованием изученных компонентов и технологий.

Задания к лабораторной работе

Задание 1. Создание многоэкранного приложения со списком

1. Создайте проект **MultiScreen**. В java-файле замените Activity на ListActivity. Класс ListActivity разработан таким образом, что на экране есть только прокручиваемый список и ему не нужна дополнительная разметка. Поэтому файл **activity_main.xml** можно удалить. Также следует удалить следующую строчку из класса MultiScreen:

```
setContentView(R.layout.activity_main);
```

т.к. layout-файл мы только что удалили.

2. Теперь нам нужен посредник, который свяжет список и названия элементов этого списка. Для этого в Android есть интерфейс Adapter. Нам потребуется наследник этого класса ArrayAdapter:

```
new ArrayAdapter<Context>(context, int  
textViewResourceId, String[] objects)
```

В качестве аргументов ArrayAdapter потребует текущий контекст, идентификатор ресурса с разметкой для каждой строки, массив строк.

Мы можем указать в качестве текущего контекста `ListActivity` (можно использовать ключевое слово `this`), готовый системный идентификатор ресурса (`android.R.layout.simple_list_item_1`) и созданный массив строк (`String[] islands = { «Канары», «Курилы», «Мальдивы», «Филиппины»};`). А выглядеть это будет так:

```
private ArrayAdapter<String> adapter; adapter = new
ArrayAdapter<String>(this,
    android.R.layout.simple_list_item_1, islands);
```

Обратите внимание на строчку `android.R.layout.simple_list_item_1`. В ней уже содержится необходимая разметка для элементов списка. Если вас не устраивает системная разметка, то можете создать собственную разметку в `xml`-файле и подключить её.

Осталось только подключить адаптер:

```
setListAdapter(adapter);
```

3. Теперь нам нужно подключить обработку нажатия. Для этого необходимо знать, на какой пункт списка осуществляется

нажатие. Существует специальный интерфейс `OnItemClickListener`

с методом `onItemClick()`. Подключаем обработчик:

```
getView().setOnClickListener(itemListener); Набираем все в том же
onCreate OnItemClickListener itemListener = new
OnItemClickListener() {
    }
```

Если подчеркнётся первое слово, то импортируем нужный класс. Далее подведём курсор ко второму слову `new OnItemClickListener`. Нам будет предложено добавить метод (Add unimplemented method). Соглашаемся и получаем заготовку:

```
OnItemClickListener itemListener = new
OnItemClickListener() {
```

```
@Override
```

```
public void onItemClick(AdapterView<?> arg0, View
```

```

arg1, int arg2,
    long arg3) {
    // TODO Auto-generated method stub
}
};

```

Меняем имена переменных `arg` на более привычные и понятные

```

public void onItemClick(AdapterView<?> parent, View v,
int position, long id);

```

4. Осталось описать, что будет происходить при нажатии на элемент.

По нашей задумке каждый элемент будет открывать новое окно с соответствующим содержимым.

Для начала следует создать еще 4 класса: `Canari`, `Curili`, `Maldivi`, `Philippini`, и 4 `xml`-блочка к ним. Можно просто создать и копировать по одному файлу в обеих директориях, меняя только название и содержимое.

Например, создадим файлы **`Canari.java`** и **`canari.xml`** типа `Activity`. Обратите внимание, что как только мы создали ещё один `java`-файл, нужно записать его в файл манифеста, иначе приложение не будет видеть этот новый класс. Файл **`AndroidManifest.xml`** находится сразу под папкой **`res`**. Добавьте код с именем класса в тегах `<activity>``</activity>`, сразу под главной активностью.

```

<activity
    android:name=«com.mypackage.multiscreen.Canari
» android:label=«@string/can» >
</activity>

```

Строку `can` нужно создавать в файле **`strings.xml`**, также как и другие строки.

Перейдем в файл **`canari.xml`** и создадим на экране `TextView`. Экраны будут отличаться

друг от друга картинками. Возьмите 4 любые картинки с вашего компьютера (можете также найти их в интернете) и перетащите из проводника `Windows` в папку **`res\drawable`**. Теперь вы можете поместить элемент `ImageView` на экран и, выбрав нужную картинку из ресурсов проекта, подключить ее.

Остальные экраны создаются аналогично.

5. Теперь перейдем в главный класс и опишем обработку события `onItemClick()`. Создадим переключатель, который зависит от номера элемента.

```
switch (position) {  
    case 0:  
        break; case 1:  
        break; case 2:  
        break; case 3:  
        break;  
}
```

Для запуска нового экрана необходимо создать экземпляр класса `Intent` и указать класс, на который будем переходить (у нас их 4, поэтому для каждого случая выбираем свою). После этого вызывается метод `startActivity()`, который и запускает новый экран.

```
Intent intent = new  
Intent(MultiScreenMainActivity.this, Canari.class);  
startActivity(intent);
```

6. Теперь осталось добавить всплывающее окно **Toast**, которое будет показывать, какой элемент мы выбрали. Этот виджет можно импортировать так же, как и предыдущие. Нам потребуется метод `makeText()`, у которого есть три параметра: контекст приложения, текстовое сообщение и продолжительность времени показа уведомления.

```
Toast.makeText(getApplicationContext(), »Вы  
выбрали«+parent.getItemAtPosition(position).toString(), To  
ast.LENGTH_SHORT).show();
```

Полные листинги файлов проекта, в которых были сделаны изменения, см. ниже.

Листинг 1. Файл `MultiScreenMainActivity.java`

```
package com.mypackage.multiscreen;  
import android.os.Bundle; import android.view.View;  
import android.app.ListActivity;  
import android.content.Intent;
```

```

import android.widget.AdapterView;
import
android.widget.AdapterView.OnItemClickListener;
import android.widget.ArrayAdapter; import
android.widget.Toast;

public class MultiScreenMainActivity extends
ListActivity{
    String[] islands = { «Канары», «Курилы», «Мальдивы»,
«Филиппины»};

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ArrayAdapter<String> adapter = new
ArrayAdapter<String>(this,
android.R.layout.simple_list_item_1, islands);
        setListAdapter(adapter);
        OnItemClickListener itemListener = new
OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View
v, int position, long id) {
                switch (position) { case 0:
                    Intent intent = new
Intent(MultiScreenMainActivity.this, Canari.class);
                    startActivity(intent); break;
                    case 1:
                        Intent intent1 = new
Intent(MultiScreenMainActivity.this, Curili.class);
                        startActivity(intent1); break;
                    case 2:
                        Intent intent2 = new

```

```

Intent (MultiScreenMainActivity.this, Maldivi.class);
    startActivity(intent2); break;
    case 3:
        Intent intent3 = new
Intent (MultiScreenMainActivity.this, Philippini.class);
        startActivity(intent3);
        break;
    }
    Toast.makeText(getApplicationContext(), «Вы выбрали «
+ parent.getItemAtPosition(position).toString(),
    Toast.LENGTH_SHORT).show();
}
};
getListView().setOnClickListener(itemListener);
}
}

```

Листинг 2. Файл Canari.java

```

package          com.mypackage.multiscreen;          import
android.app.Activity; import android.os.Bundle;
public class Canari extends Activity
{
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.canari);
    }
}

```

Листинг 3. Файл canari.xml

```

<?xml version=«1.0» encoding=«utf-8»?>

```



```

        <RelativeLayout
xmlns:android=«http://schemas.android.com/apk/res/android
«
        android:id=«@+id/RelativeLayout1»
android:layout_width=«fill_parent»
android:layout_height=«fill_parent»
android:padding=«10dip» >
        <ImageView
        android:id=«@+id/imageView1»
        android:layout_width=«wrap_content»
        android:layout_height=«wrap_content»
        android:layout_alignParentTop=«true»
        android:layout_centerHorizontal=«true»
android:layout_marginTop=«186dp»
android:src=«@drawable/canari» />
        <TextView
        android:id=«@+id/textView1»
android:layout_width=«wrap_content»
android:layout_height=«wrap_content»
android:layout_alignParentTop=«true»
android:layout_centerHorizontal=«true»
android:layout_marginTop=«80dp»
android:text=«@string/enjoy»
        android:textAppearance=«?android:attr/textAppearanceL
arge» />
</RelativeLayout>

```

Листинг 4. Файл strings.xml

```

<?xml version=«1.0» encoding=«utf-8»?>
<resources>
    <string      name=«app_name»>Куда      бы      поехать      в
отпуск?</string>

```

```
<string name=«action_settings»>Settings</string>
<string name=«enjoy»>Enjoy yourself!</string>
<string name=«can»>Канары</string>
<string name=«phil»>Филиппины</string>
<string name=«cur»>Курилы</string>
<string name=«mal»>Мальдивы</string>
</resources> (рис. 9.1, 9.2)
```

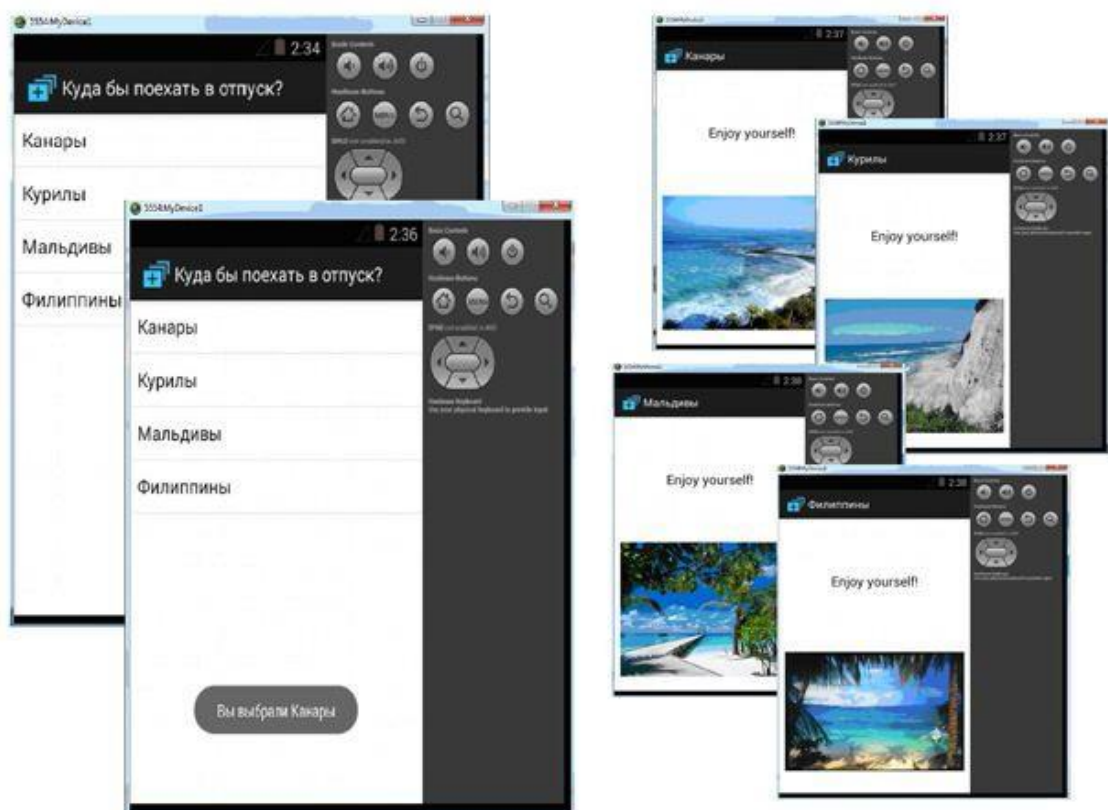


Рисунок 9.1 – Приложение «Куда бы поехать в отпуск?», запущенное на эмуляторе

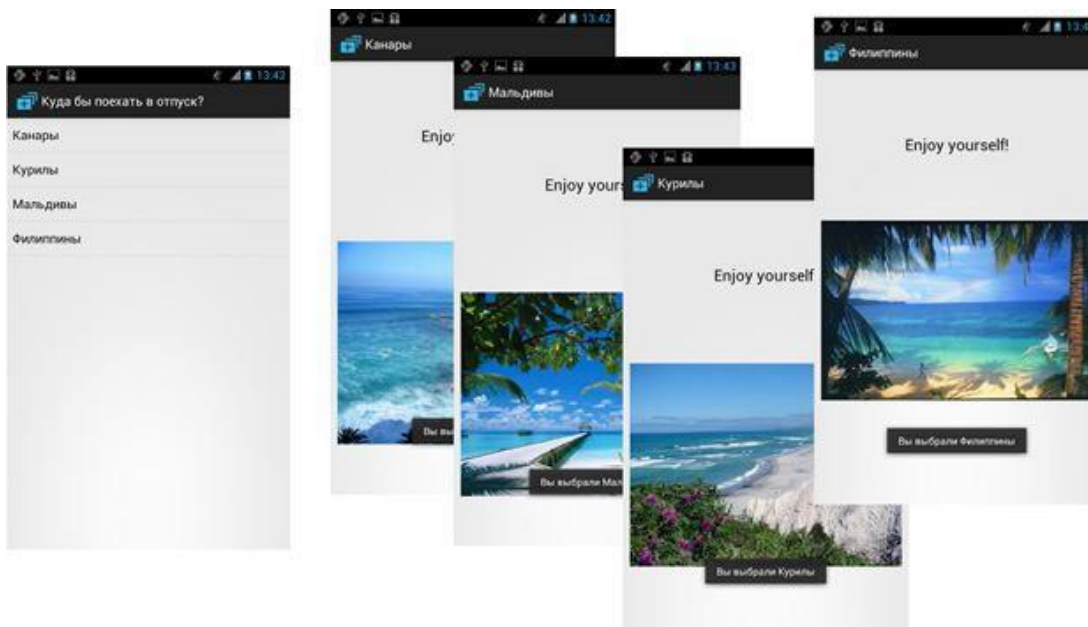


Рисунок 9.2 – Приложение «Куда бы поехать в отпуск?», запущенное на устройстве

Задание 2. Создание диалогового окна

1. Создайте новый проект **Dialog**
2. Создайте кнопку на вашей активности и назовите ее «**Выбрать фон**» (для этого нужно в файле **strings** создать строку соответствующего содержания). Присвойте активности и кнопке **id**.

Листинг 5. Код файла **activity_main.xml**

```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
« android:id="@+id/relative_layout"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >
    <Button
        android:id="@+id/background_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="174dp»
```

```
android:text=«@string/bg»/>
</RelativeLayout>
```

3. Наше приложение меняет фон на выбранный. Значит, нам нужно создать цвета и их названия в файле **strings.xml**. Как вы помните, этот файл находится в папке **values**, которая в свою очередь находится в папке **res**. Также создадим строку **messages**, которая нам понадобится для диалогового окна.

Листинг 6. Файл **strings.xml**

```
<?xml version=«1.0» encoding=«utf-8»?>
<resources>
<string name=«app_name»>Dialog</string>
<string name=«action_settings»>Settings</string>
<string name=«bg»>Выбрать фон</string>
<string name=«message»>Хотите поменять фон?</string>
<color name=«redColor»>#FFFF0000</color>
<color name=«yellowColor»>#FFFF00</color>
<color name=«greenColor»>#FF00FF00</color>
<string name=«red»>Красный</string>
<string name=«yellow»>Жёлтый</string>
<string name=«green»>Зелёный</string>
</resources>
```

4. Перейдем в файл **MainActivity.java**. Создайте следующие переменные:

```
private Button bgButton;
public RelativeLayout relativeLayout;
Context context;
```

Если компилятор подчеркивает тип и сообщает об ошибке, например, подчеркивает **Context**, наведите курсор на подчеркнутое слово: должно появиться контекстное меню, предлагающее варианты, как можно исправить ошибку. Выберите **Import 'Context'**, чтобы импортировать библиотеку.

5. Теперь нужно описать, что будет происходить при нажатии на нашу кнопку.

Для начала свяжем объекты из **activity_main.xml** и переменные в **MainActivity.java** через

id (в onCreate):

```
bgButton =  
(Button)findViewById(R.id.background_button);  
relativeLayout =  
(RelativeLayout)findViewById(R.id.relativelayout);
```

Context это объект, который предоставляет доступ к базовым функциям приложения.

Добавляем в код

```
context = MainActivity.this;
```

6. Теперь нужно добавить событие onClick и навесить на кнопку OnClickListener. Добавляем в заголовок класса MainActivity implements OnClickListener. Чтобы связать кнопку и Listener в onCreate пишем

```
bgButton.setOnClickListener(this);
```

Создадим теперь событие onClick

```
@Override  
public void onClick(View v){  
}
```

В этом объекте создаем собственно диалог и называем его:

```
AlertDialog.Builder builder = new  
AlertDialog.Builder(this);  
builder.setTitle(R.string.message);  
AlertDialog alert = builder.create(); alert.show();
```

Мы будем создавать диалоговое окно, предоставляющее пользователю выбор из списка. Для этого потребуется ещё одна переменная, которая сформирует список из имеющихся строк.

```
final CharSequence[] items = {  
    getText(R.string.red),      getText(R.string.yellow),  
    getText(R.string.green)  
};
```

7. Сформируем собственно наш список и зададим еще один Listener, который будет менять цвет фона на выбранный:

```
builder.setItems(items, new
DialogInterface.OnClickListener() { public void
onClick(DialogInterface dialog, int item) {
    switch (item) {
        case 0: {
relativeLayout.setBackgroundResource(R.color.redColor);
        break;}
        case 1:
{relativeLayout.setBackgroundResource(R.color.yellowColor
);
        break;}
        case 2:
{relativeLayout.setBackgroundResource(R.color.greenCo
lor);
        break;}
    }
}
```

8. Осталось добавить в каждый case всплывающие окна Toast, и приложение полностью готово!

Листинг 7. Код файла MainActivity.java

```
Toast.makeText(context, R.string.green,
Toast.LENGTH_LONG).show(); package com.mypackage.dialog;
import android.app.Activity; import
android.app.AlertDialog; import android.content.Context;
import android.content.DialogInterface;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener; import
android.widget.Button;
import android.widget.RelativeLayout; import
```

```

android.widget.Toast;

    public class MainActivity extends Activity implements
OnClickListener {
    private Button bgButton;
    public RelativeLayout relativeLayout;
    Context context;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        bgButton = (Button)
findViewById(R.id.background_button);
        bgButton.setOnClickListener(this);        context =
MainActivity.this;
        relativeLayout =
(RelativeLayout)findViewById(R.id.relativelayout);
    }

    @Override
    public void onClick(View v) {
        final CharSequence[] items = {getText(R.string.red) ,
getText(R.string.yellow),getText(R.string.green)
        };

        AlertDialog.Builder builder = new
AlertDialog.Builder(this);
        builder.setTitle(R.string.message);
        builder.setItems(items, new
DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog, int item)
{
                switch (item) {
                    case 0: {

```

```

relativeLayout.setBackgroundResource(R.color.redColor);
    Toast.makeText(context, R.string.red,
        Toast.LENGTH_LONG).show();
    break;}

    case 1:
{relativeLayout.setBackgroundResource(R.color.yellowColor
);
    Toast.makeText(context, R.string.yellow,
        Toast.LENGTH_LONG).show();
    break;}

    case
2:{relativeLayout.setBackgroundResource(R.color.greenColor
);
    Toast.makeText(context, R.string.green,
        Toast.LENGTH_LONG).show();
    }
    });
break;}

    }

    AlertDialog alert = builder.create(); alert.show();
}
}

```

Скриншоты работающего приложения (рис. 9.3 - 9.6)

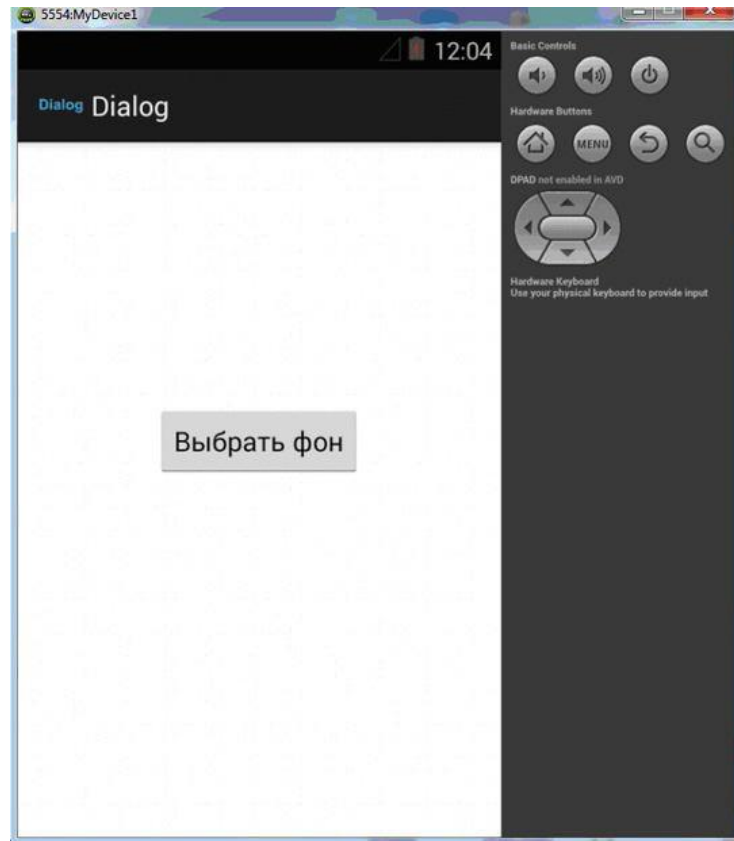


Рисунок 9.3 – Приложение «Dialog», запущенное на эмуляторе

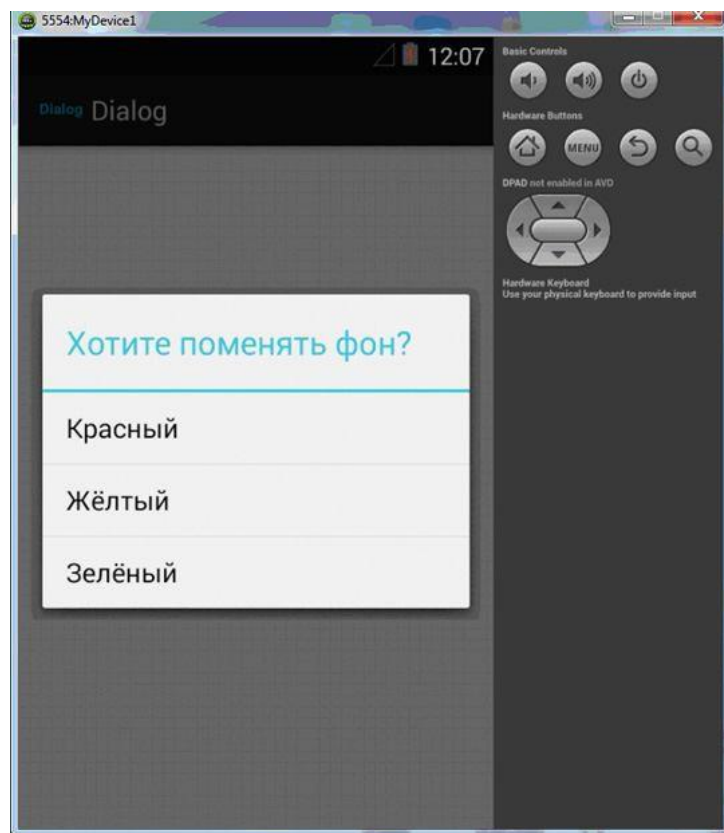


Рисунок 9.4 – Диалоговое окно

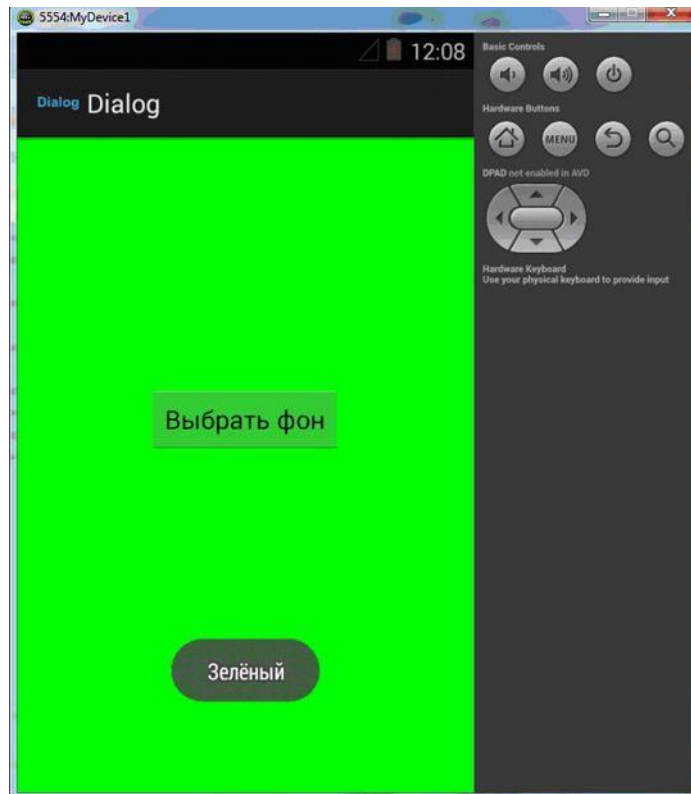


Рисунок 9.5 – Выбран зеленый фон

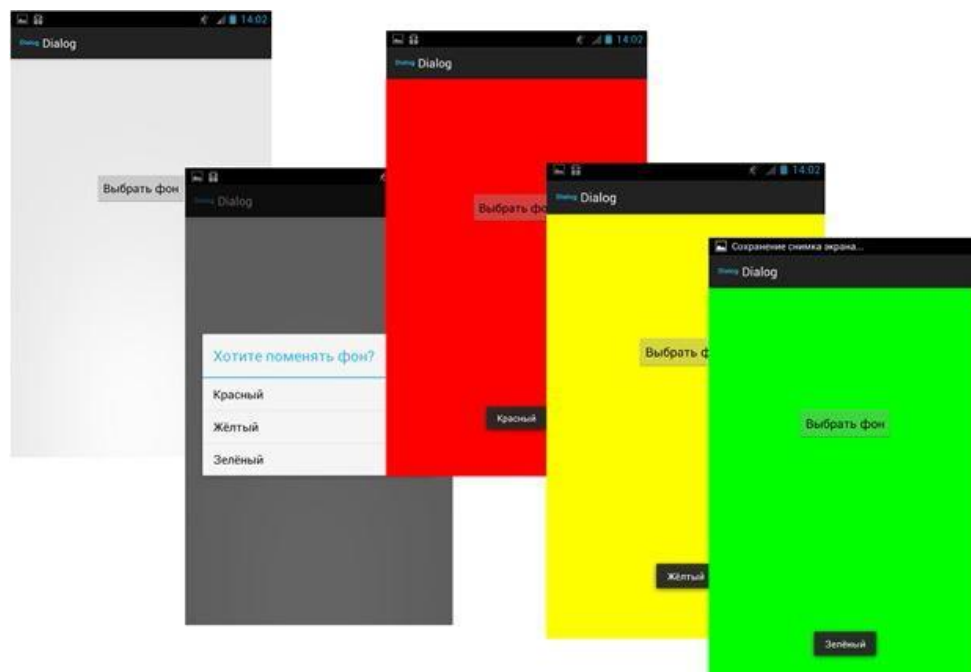


Рисунок 9.6 – Приложение «Dialog», запущенное на устройстве

Задание 3. Создание приложения со слайдингом из шаблона

1. Создайте проект **TabsAndSwipe**. Обратите внимание: чтобы использовать стандартный шаблон активности **Fixed Tabs + Swipe**, вам

необходимо при создании проекта указать

Minimum Required SDK не меньше, чем **API11**, т.к. в более ранних версиях этот шаблон не поддерживается (рис. 9.7).

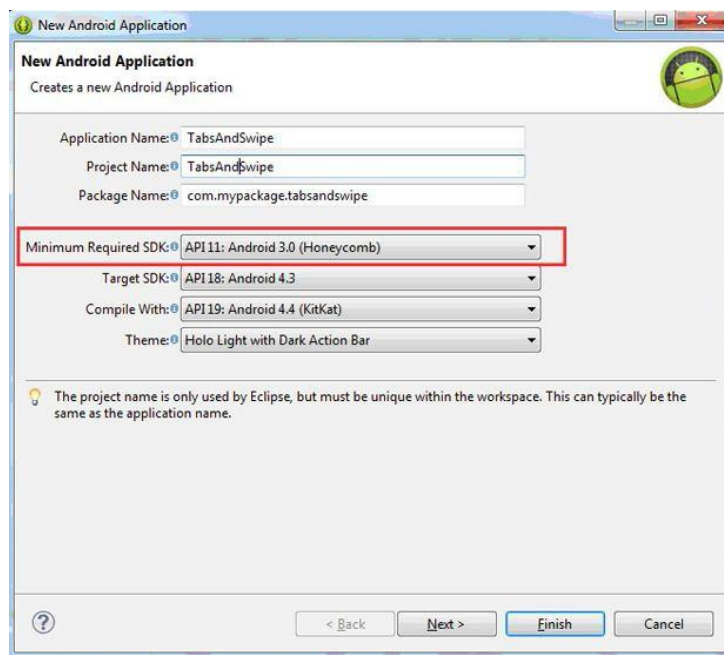


Рисунок 9.7 – Создание приложения TabsAndSwipe Следующие три окна оставляем без изменений

Далее в окне **Blank Activity** выбираем в графе **Navigation Type** (рис. 9.8).

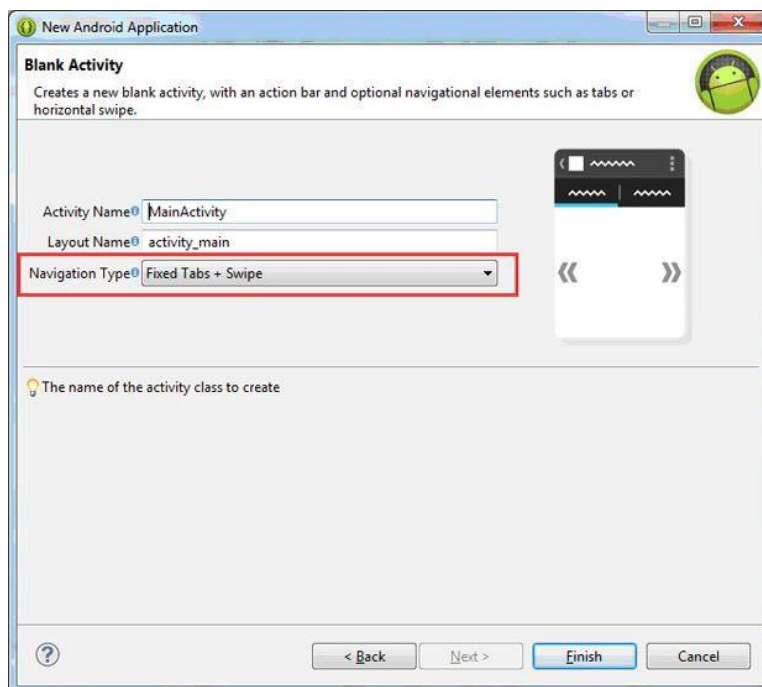


Рисунок 9.8 – Выбор шаблона Navigation Type

2. Посмотрите на структуру проекта: у вас появились два xml-файла в папке **res/layout activity_main.xml** и **fragment_main_dummy.xml**.

3. Запустите приложение, чтобы убедиться, что все компилируется правильно.

4. Создайте три копии файла **fragment_main_dummy.xml**. Назовите их соответственно **first_tab**, **second_tab** и **third_tab**. Присвойте элементу TextView в каждом файле уникальный id. Можете поместить на экраны какие-нибудь элементы, например картинки или надписи.

5. Теперь переходим к файлу **MainActivity.java**. Нас интересует класс

```
public static class DummySectionFragment extends
Fragment
```

Делаем три копии данного класса со всем содержимым, называя их соответственно

FirstActivity, SecondActivity, ThirdActivity.

6. Поменяйте в них следующие строки

```
View rootView =
    inflater.inflate(R.layout.fragment_main_dummy,
        container, false);

TextView dummyTextView =
    (TextView)
    rootView.findViewById(R.id.section_label);
```

Замените **R.layout.fragment_main_dummy** на **R.layout.first** и **R.id.section_label** на **R.id.section_label1** соответственно.

7. Поменяйте строки-названия секций в файле **strings.xml**.

```
<string name=«title_section1»>Лента</string>
<string name=«title_section2»>Фото</string>
<string name=«title_section3»>Карта</string>
```

8. Теперь переходим к классу **SectionsPagerAdapter()**. Нас интересует его метод **Fragment getItem()**, именно его мы будем изменять. Чтобы при

перелистывании менялось не только содержимое TextView, но и всего фрагмента, замените код этого метода на следующий:

```
public Fragment getItem(int position) {
    // getItem is called to instantiate the fragment for
the given page.
    1);
    1);
    1);
    // Return a DummySectionFragment (defined as a static
inner class
    // below) with the page number as its lone argument.
    Fragment fragment=null;
    Bundle args;
    switch (position) { case 0:
        fragment = new FirstFragment(); args = new Bundle();
        args.putInt(FirstFragment.ARG_SECTION_NUMBER,
position +
        fragment.setArguments(args); break;
        case 1:
            fragment = new SecondFragment();
            args = new Bundle();
args.putInt(SecondFragment.ARG_SECTION_NUMBER, position +
            fragment.setArguments(args); break;
            case 2:
                fragment = new ThirdFragment();
                args = new Bundle();
                args.putInt(ThirdFragment.ARG_SECTION_NUMBER,
position +
                fragment.setArguments(args); break;
            }
    return fragment;
```

}

9. Приложение готово, можно запускать (рис. 9.9).

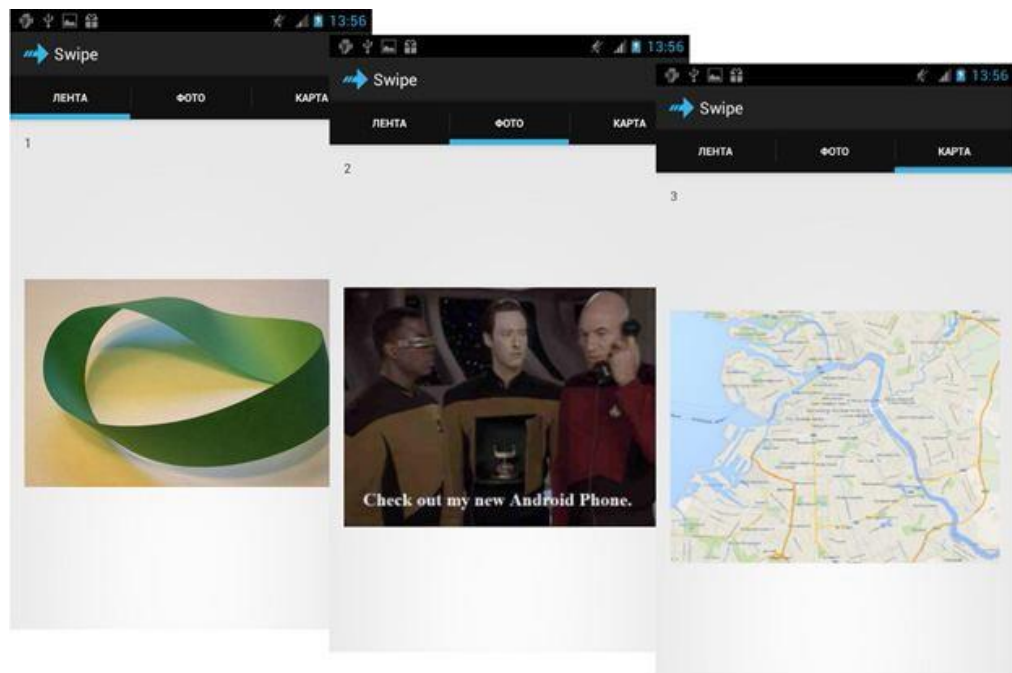


Рисунок 9.9 – Приложение TabsAndSwipe, запущенное на устройстве

Содержание отчета и его форма

Отчет к лабораторной работе оформляется в виде текстового документа по форме простого реферата и должен включать:

1. Название лабораторной работы.
2. Цель лабораторной работы.
3. Краткие выводы по результатам выполнения лабораторной работы согласно пунктам раздела «Методики выполнения работы».
4. Ответы на Вопросы к лабораторной работе.
5. Формулировку задания для самостоятельной работы и результат его выполнения.

Задания для самостоятельной работы

1. Подумайте над собственным приложением, сочетающим различные возможности проектирования многооконных приложений, рассмотренные выше.
2. Создайте прототип этого приложения и настройте его пользовательский интерфейс.

Вопросы к лабораторной работе

1. Технология создания многоэкранного приложения со списком.
2. Технология создания диалогового окна.
3. Технология создания приложения со слайдингом из шаблона.

Лабораторная работа 10

Демонстрации распознавания стандартных жестов

Цель работы: разработать простейшие приложения для демонстрации распознавания стандартных жестов.

Теоретическое обоснование

Для работы со стандартными жестами Android предоставляет класс `GestureDetector`. Этот класс содержит два вложенных интерфейса-слушателя: `OnGestureListener` и `OnDoubleTapListener`, эти интерфейсы задают методы, отслеживающие стандартные жесты. А также `GestureDetector` содержит вложенный класс `SimpleOnGestureListener`, который содержит пустые реализации, возвращающие значение `false`, где это необходимо, всех методов интерфейсов: `OnGestureListener` и `OnDoubleTapListener`.

В лабораторной работе рассмотрим две возможности распознавания жестов:

- случай распознавания всех поддерживаемых жестов, для этого реализуем в классе активности оба интерфейса;
- случай распознавания только некоторого набора поддерживаемых жестов, для этого в классе активности объявим внутренний класс-наследник класса `GestureDetector.SimpleOnGestureListener`.

Задания к лабораторной работе

Разработаем приложение, в котором продемонстрируем распознавание всех поддерживаемых жестов. Приложение содержит одну активность, одно информационное поле для вывода информации о распознанном жесте. Приложение работает следующим образом: пользователь выполняет один из поддерживаемых сенсорных жестов, в информационном поле отображается

информация о распознанном жесте.

1. Создадим простое приложение и добавим на форму TextView для вывода информации.

2. Настроим логику приложения. В java класс, соответствующий активности внесем следующие дополнения.

3. Класс активности должен реализовывать интерфейсы: GestureDetector.OnGestureListener и GestureDetector.OnDoubleTapListener, для этого в объявление класса добавим конструкцию:

```
implements GestureDetector.OnGestureListener,  
GestureDetector.OnDoubleTapListener
```

4. Нам понадобится экземпляр класса GestureDetectorCompat поэтому в качестве поля класса активности объявим следующую переменную:

```
GestureDetectorCompat mDetector;
```

В методе onCreate() класса активности, создадим экземпляр класса GestureDetectorCompat и присвоим его переменной mDetector:

```
mDetector = new GestureDetectorCompat(this, this);
```

одним из параметров конструктора является класс, который реализует интерфейс GestureDetector.OnGestureListener, в нашем случае использовано слово this, т. е. параметром является сам класс активности. Этот интерфейс уведомляет пользователей когда появляется определенное сенсорное событие.

В методе onCreate() класса активности, следующая строка:

```
mDetector.setOnDoubleTapListener(this);
```

устанавливает слушатель событий, связанных с двойным касанием, это должен быть класс, реализующий интерфейс GestureDetector.OnDoubleTapListener. В нашем случае использовано слово this, т.е. слушателем будет опять сам класс активности.

1. Чтобы позволить вашему объекту GestureDetector получать события, необходимо переопределить метод onTouchEvent() для активности или элемента GUI. И передавать в экземпляр детектора все обнаруженные события.

```
public boolean onTouchEvent(MotionEvent event) {  
this.mDetector.onTouchEvent(event);
```



```
// Be sure to call the superclass implementation return
super.onTouchEvent(event);
}
```

2. После проведенной подготовки пришло время реализовать все методы, объявленные в интерфейсах, отвечающих за прослушивание сенсорных событий.

Методы интерфейса `GestureDetector.OnGestureListener`:

<code>onDown()</code>	отслеживает появление касания, т. е. палец прижат к экрану;
<code>onFling()</code>	отслеживает появление жеста смахивания;
<code>onLongPress()</code>	отслеживает удерживание пальца прижатым к экрану длительное время;
<code>onScroll()</code>	отслеживает появление жеста прокрутки (пролистывания);
<code>onShowPress()</code>	отслеживает, что произошло событие касания и больше никаких событий не происходит короткое время;
<code>onSingleTapUp()</code>	отслеживает появление жеста одиночного нажатия (клик).

Методы интерфейса `GestureDetector.OnDoubleTapListener`:

<code>onDoubleTap()</code>	отслеживает появление жеста двойного нажатия («двойной клик»);
<code>onDoubleTapEvent()</code>	отслеживает появление события во время выполнения жеста двойного нажатия, включая касание, перемещение, подъем пальца.

onSingleTapConfirmed()	отслеживает появление жеста одиночного нажатия (клик).

В листинге 18.1 представлен код приложения, в котором распознаются все поддерживаемые жесты, информация о появившемся и распознанном жесте выдается в информационное поле (TextView).

В качестве практики предлагается воспроизвести данное приложение и проверить, как система распознает тот или иной жест. Очень полезно для понимания, как выполняются основные жесты.

Задание 2. Распознавание только части поддерживаемых жестов

Разработаем приложение, в котором продемонстрируем распознавание только некоторой части поддерживаемых жестов по выбору программиста. Мы рассмотрим распознавание жеста смахивания (fling). Приложение содержит одну активность, одно информационное поле для вывода информации о распознанном жесте. Приложение работает следующим образом: пользователь выполняет один из поддерживаемых сенсорных жестов, в информационном поле отображается информация о распознанном жесте.

1. Создадим простое приложение и добавим на форму TextView для вывода информации.

2. Настроим логику приложения. В java класс, соответствующий активности внесем следующие дополнения.

Нам понадобится экземпляр класса GestureDetectorCompat поэтому в качестве поля класса активности объявим следующую переменную:

```
GestureDetectorCompat mDetector;
```

В методе onCreate() класса активности, создадим экземпляр класса GestureDetectorCompat и присвоим его переменной mDetector:

```
mDetector=new GestureDetectorCompat(this, new MyGestListener());
```

в конструкторе аргументом, отвечающим за отслеживание сенсорных

событий, служит экземпляр класса `MyGestListener()` внутренний класс, который является наследником класса `GestureDetector.SimpleOnGestureListener`.

Имеет смысл рассмотреть класс `GestureDetector.SimpleOnGestureListener`. Этот

класс реализует интерфейсы `GestureDetector.OnGestureListener` и `GestureDetector.OnDoubleTapListener`, все методы заявленные в интерфейсах в этом классе имеют пустую реализацию и те, которые должны возвращать значение, возвращают `false`. Поэтому для распознавания какого-то события или некоторого подмножества событий достаточно написать реализацию соответствующих методов, в классе наследнике.

В листинге 18.2 представлен код приложения, в котором распознается только жест смахивания, т. е. реализован метод `onFling()`, информация о появившемся и распознанном жесте выдается в информационное поле (`TextView`). В качестве слушателя используется экземпляр класса `MyGestListener()`, являющийся наследником класса `GestureDetector.SimpleOnGestureListener`.

Получить больше информации о распознавании жестов можно по ссылке:

<http://developer.android.com/training/gestures/index.html>

Листинг 1. Распознавание поддерживаемых жестов с помощью реализации интерфейсов

```
package com.example.lab5_1_gestall; import
android.os.Bundle;
import android.app.Activity;
import android.support.v4.view.GestureDetectorCompat;
import android.view.*;
import android.widget.*;
public class MainActivity extends Activity implements
GestureDetector.OnGestureListener,
GestureDetector.OnDoubleTapListener
{
```

```

    TextView tvOutput; GestureDetectorCompat mDetector;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        tvOutput = (TextView)findViewById(R.id.textView1);
        mDetector = new GestureDetectorCompat(this,this);
        mDetector.setOnDoubleTapListener(this);
    }
    public boolean onTouchEvent(MotionEvent event){
        this.mDetector.onTouchEvent(event);
        // Be sure to call the superclass implementation return
        super.onTouchEvent(event);
    }
    @Override
    public boolean onDown(MotionEvent event) {
        tvOutput.setText(«onDown: « + event.toString());
        return false;
    }
    @Override
    public boolean onFling(MotionEvent event1, MotionEvent
event2,
        float velocityX, float velocityY) {
        tvOutput.setText(«onFling: « +
event1.toString()+event2.toString());
        return true;
    }
    @Override
    public void onLongPress(MotionEvent event) {
        tvOutput.setText(«onLongPress: « + event.toString());
    }

```

```

    }

    @Override
    public boolean onScroll(MotionEvent e1, MotionEvent
e2, float distanceX,
    float distanceY) {
        tvOutput.setText («onScroll:          «          +
e1.toString()+e2.toString());
        return true;
    }

    @Override
    public void onShowPress(MotionEvent event) {
        tvOutput.setText («onShowPress: « + event.toString());
    }

    @Override
    public boolean onSingleTapUp(MotionEvent event) {
        tvOutput.setText («onSingleTapUp:          «          +
event.toString());
        return true;
    }

    @Override
    public boolean onDoubleTap(MotionEvent event) {
        tvOutput.setText («onDoubleTap: « + event.toString());
        return true;
    }

    @Override
    public boolean onDoubleTapEvent(MotionEvent event) {
        tvOutput.setText («onDoubleTapEvent:          «          +
event.toString());
        return true;
    }

    @Override

```

```

        public boolean onSingleTapConfirmed(MotionEvent event)
        {
            tvOutput.setText («onSingleTapConfirmed:      «      +
event.toString());
            return true;
        }
    }
}

```

**Листинг 2. Распознавание жестов с использованием класса
GestureDetector.SimpleOnGestureListener.**

```

package      com.example.lab5_1_gestsubset;      import
android.os.Bundle;

import android.app.Activity;
import android.support.v4.view.*;
import android.view.*;
import android.widget.*;

public class SubsetGestActivity extends Activity {
private GestureDetectorCompat mDetector;
private TextView tvOut;
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_subset_gest);
    mDetector = new GestureDetectorCompat(this, new
MyGestListener());
    tvOut = (TextView)findViewById(R.id.textView1);
}
@Override
public boolean onTouchEvent(MotionEvent event){
    this.mDetector.onTouchEvent(event);
    return super.onTouchEvent(event);
}
}

```

```

    }

    class MyGestListener extends
GestureDetector.SimpleOnGestureListener {

    @Override

    public boolean onFling(MotionEvent event1, MotionEvent
event2, float velocityX, float velocityY) {

        tvOut.setText («onFling:          «          +
event1.toString()+event2.toString()); return true;

    }

    }

    }

```

Содержание отчета и его форма

Отчет к лабораторной работе оформляется в виде текстового документа по форме простого реферата и должен включать:

1. Название лабораторной работы.
2. Цель лабораторной работы.
3. Краткие выводы по результатам выполнения лабораторной работы согласно пунктам раздела «Методики выполнения работы».
4. Ответы на Вопросы к лабораторной работе.
5. Формулировку задания для самостоятельной работы и результат его выполнения.

Задания для самостоятельной работы

1. Проанализировать распознавание всех поддерживаемых жестов;
2. Проанализировать распознавание только части поддерживаемых жестов.

Вопросы к лабораторной работе

1. Характеристика класса GestureDetector.
Технология распознавания поддерживаемых жестов с помощью реализации интерфейсов.
2. Технология распознавания жестов с использованием класса GestureDetector.SimpleOnGestureListener.

Лабораторная работа 11

Принципы работы с жестами вводимыми пользователями

Цель работы: разработка приложения, помогающего понять принципы работы с жестами вводимыми пользователями.

Теоретическое обоснование

Начиная с версии 1.6, Android предоставляет API для работы с жестами, который располагается в пакете `android.gesture` и позволяет сохранять, загружать, создавать и распознавать жесты.

В данной работе рассмотрим процесс создания набора жестов, для создания жестов будем

использовать приложение `Gesture Builder`, которое предустановлено на виртуальные устройства Android (AVD), начиная с версии 1.6.

После этого разработаем приложение, в котором предполагается распознавание и использование созданных жестов.

Задания к лабораторной работе

Задание 1. Создание набора жестов

Для начала создадим новое приложение.

Далее запустим эмулятор и используем приложение `Gesture Builder` для создания жестов «1», «2» и «S». Жест всегда связан с именем, но имя не обязательно должно быть уникальным, в действительности, для повышения точности в распознавании жеста рекомендуется сохранять несколько жестов с одним и тем же именем.

На рисунке 11.1 можно увидеть приложение `Gesture Builder` в работе, чтобы добавить жест, необходимо нажать на кнопку **Add gesture**, в свободном пространстве изобразить жест (обычно он рисуется желтым цветом), в поле ввода задать имя жеста. Результат последовательного добавления жестов можно увидеть на рисунке 11.2.

Жесты сохраняются на SD карте эмулятора, чтобы использовать их в приложении необходимо импортировать файл жестов в проект.

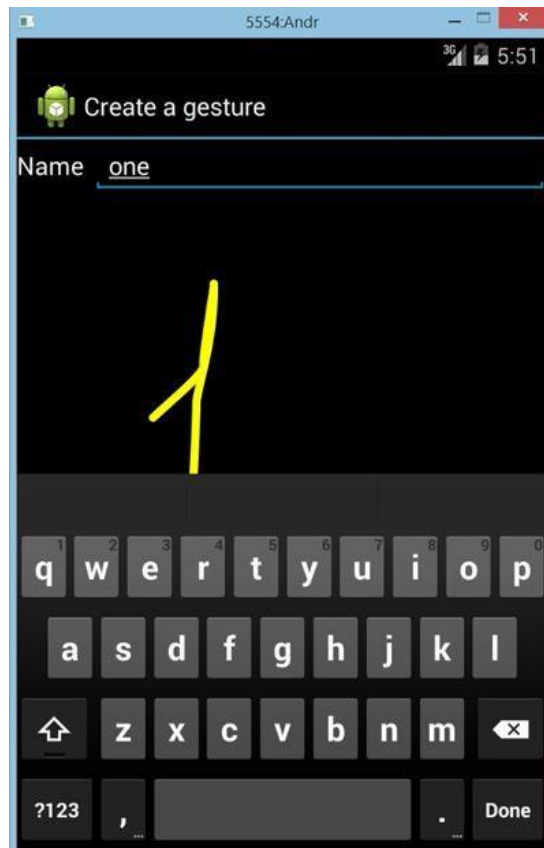


Рисунок 11.1 – Создание жестов с помощью приложения Gesture Builder

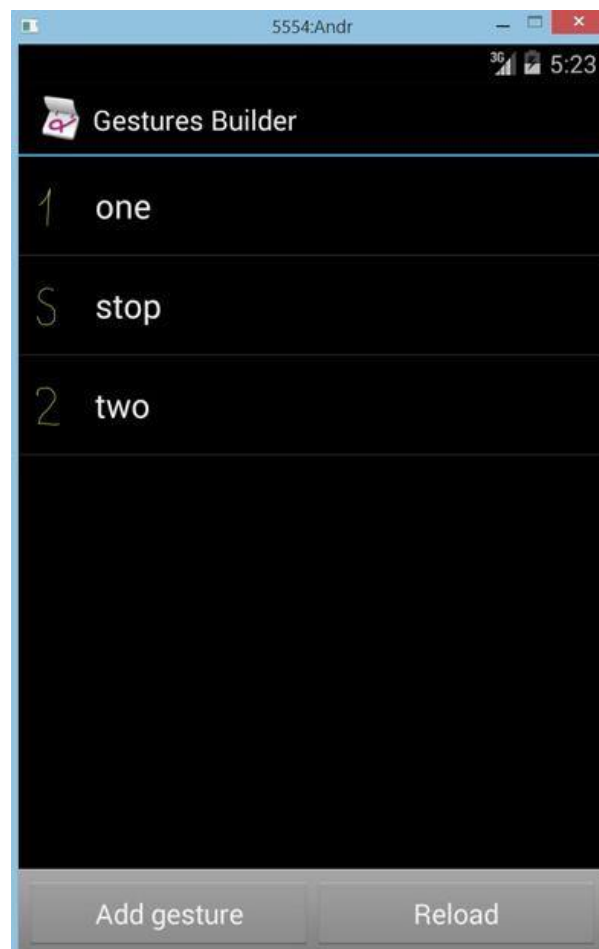


Рисунок 11.2 – Набор жестов, созданных в приложении Gesture Builder

Примечание: Gesture Builder может сообщить, что ему некуда сохранять жесты, в этом случае необходимо запустить эмулятор с образом SD карты.

Сначала образ нужно создать с помощью утилиты **mksdcard** (расположена в папке

<AndrSDK>/sdk/tools, где <AndrSDK> путь, куда установлен Android SDK). В командной строке напишем:

```
mksdcard -l mySdCard 64M gesture.img
```

Следующим шагом будет создать и запустить эмулятор с образом gesture.img, для этого зайдём в папку <AndrSDK>/sdk/tools и выполним команду:

```
emulator -avd nameEmulator -sdcard gestures.img
```

nameEmulator имя, которое присвоено эмулятору при создании.

Каждый раз, когда мы создаем или редактируем жесты с помощью Gesture Builder, создается файл gestures на SD карте эмулятора. Необходимо импортировать этот файл в директорию **res/raw/**, созданного проекта, в котором планируем использовать жесты (рис. 11.3).

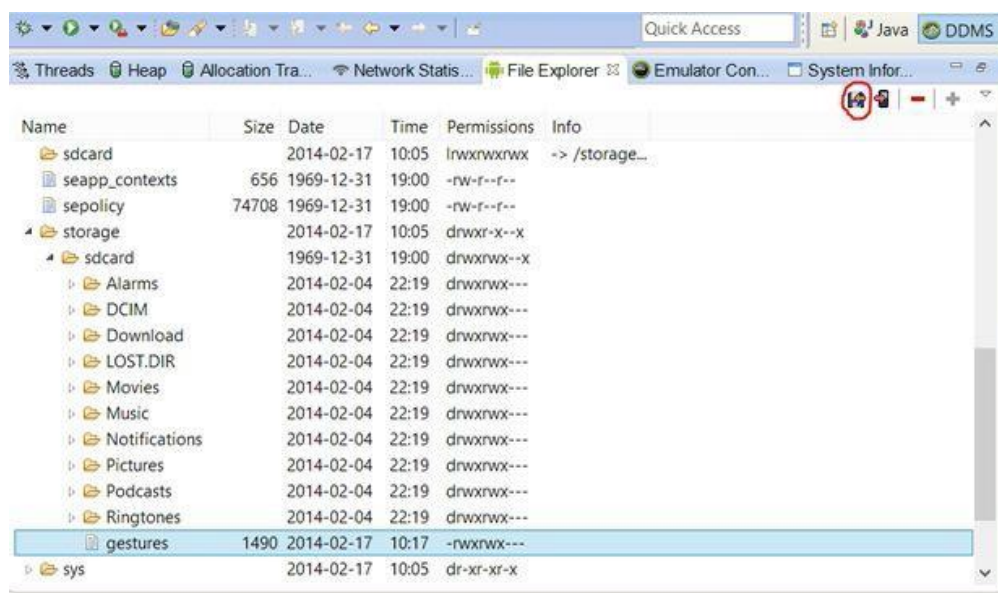


Рисунок 11.3 – Расположение файла gestures

Самый простой способ импортировать жесты в проект заключается в использовании вкладки **File Explorer** в компоновке (perspective) DDMS. (Если

компоновки DDMS нет найти ее можно следующим образом: **Window->Open Perspective->Other...->DDMS**. Если вкладки **File Explorer** нет, можно добавить: **Window-> Show View-> File Explorer**). На вкладке **File Explorer** найти директорию **sdcard/** (в нашем случае оказалась директория **storage/sdcard/**, имеет смысл при создании жестов обратить внимание в какую директорию Gesture Builder их сохраняет). На рис. 3 показана вкладка **File Explorer** в компоновке DDMS.

Чтобы скопировать файл жестов с эмулятора в проект, необходимо выбрать его и нажать кнопку «**Pull a file from the device**», выделенную на рис. 3 красным подобием окружности. Откроется диалог с предложением выбрать папку, в которую необходимо скопировать жесты, здесь надо найти папку проекта, в ней папку **res/raw/** (если папки **raw/** нет, ее необходимо создать) и нажать кнопку **Сохранить**. Теперь жесты есть в нашем проекте и их можно использовать.

Задание 2. Использование созданных жестов в приложении

Для распознавания жестов необходимо добавить элемент **GestureOverlayView** в XML файл активности. И этот файл может выглядеть, например, как показано на рисунке 11.4:

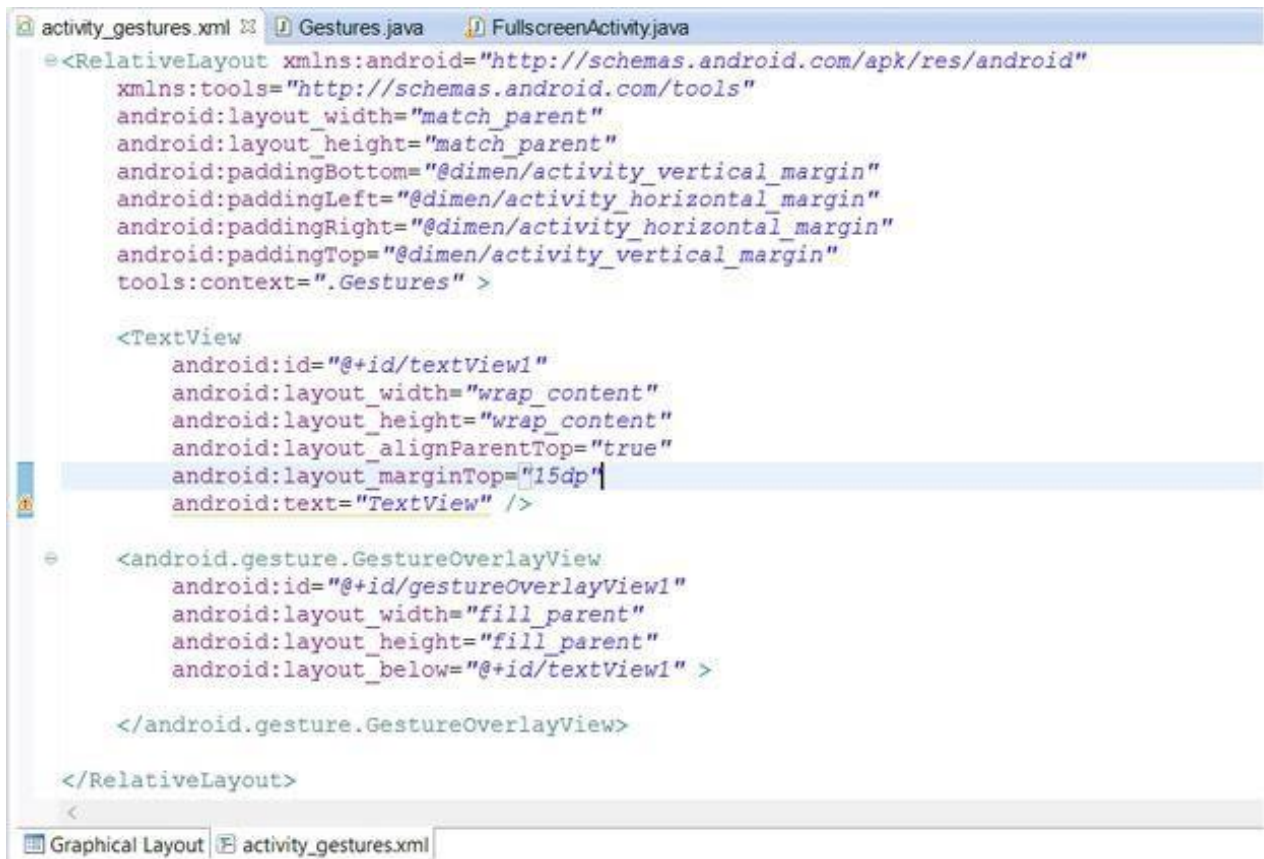


Рисунок 11.4 – XML файл активности приложения, элемент GestureOverlayView обычный компонент интерфейса пользователя

Можно добавить элемент GestureOverlayView поверх всех компонентов, как прозрачный слой, в этом случае XML файл активности может выглядеть так, как показано на рисунке 11.5.

Далее необходимо обработать ввод жеста пользователя, сравнить с загруженными жестами, и либо определить жест, либо сообщить пользователю, что такого жеста нет. Теперь вся работа будет выполняться в java файле, описывающем главную (и единственную) активность приложения. Внесем в этот класс следующие дополнения:

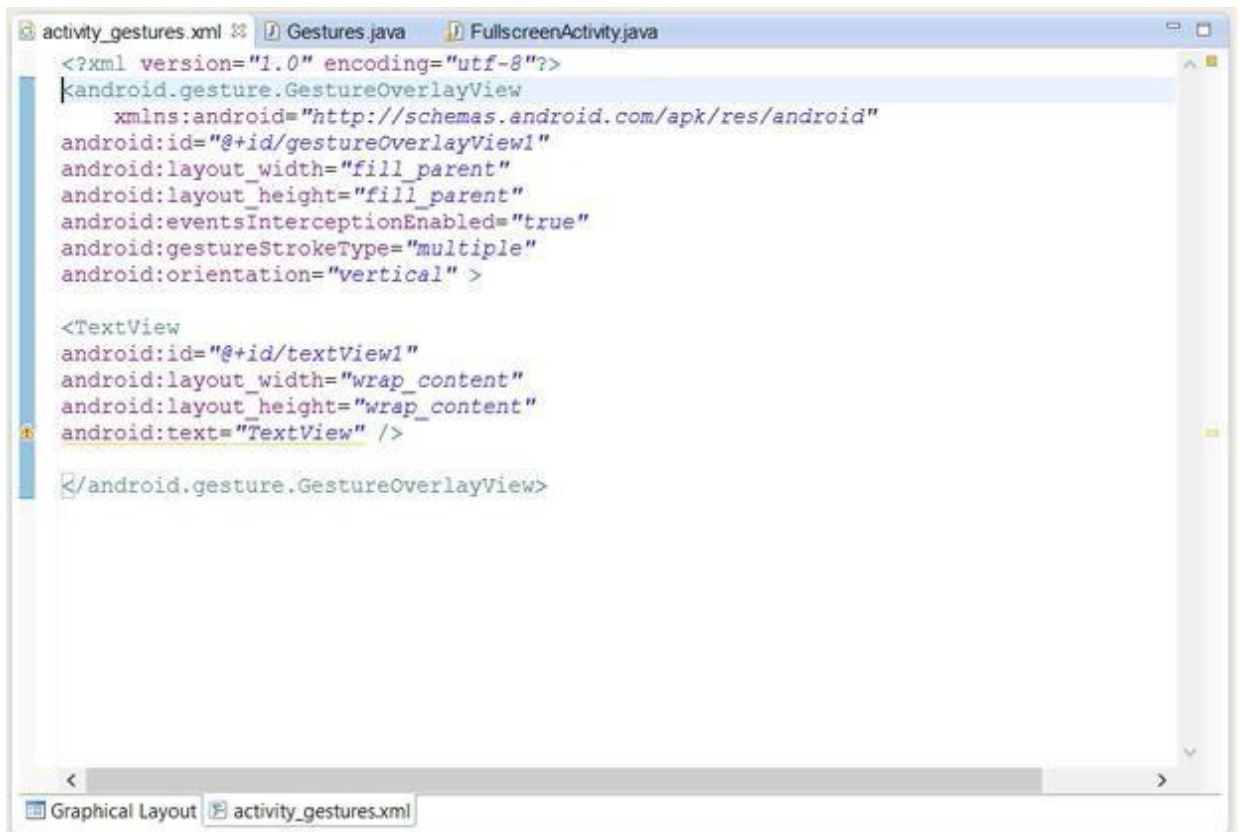


Рисунок 11.5 – XML файл активности приложения, элемент GestureOverlayView поверх всех компонентов интерфейса пользователя

Класс активности должен реализовывать интерфейс OnGesturePerformedListener, для этого в объявление класса добавим конструкцию:

```
implements OnGesturePerformedListener;
```

Нам понадобятся экземпляры классов GestureLibrary и GestureOverlayView, поэтому в качестве полей класса активности объявим следующие переменные:

```
GestureLibrary gLib; GestureOverlayView gestures;
```

В методе onCreate() выполним следующие действия:

```
gLib = GestureLibraries.fromRawResource(this,
R.raw.gestures); if (!gLib.load()) {
    finish();
}
```

В первой строке выполнена инициализация переменной gLib жестами,

загруженными из файла gestures папки **res/raw/**.

Оператор `if` выполняет проверку загружены ли жесты, если нет, выполняется выход из приложения.

```
Добавим в метод onCreate() еще две строчки: gestures =
(GestureOverlayView)findViewById(R.id.gestureOverlayView1
); gestures.addOnGesturePerformedListener(this);
```

Для инициализации переменной `gesture` и подключения к ней слушателя событий появления жеста.

И наконец, напомним реализацию метода `OnGesturePerformed()`, который и будет вызываться при появлении события, соответствующего какому-либо жесту.

Листинг 1. Распознавание жестов загруженных в файл `res/raw/gestures`

```
public void
    onGesturePerformed(GestureOverlayView overlay,
        Gesture gesture) {
    //Создаёт ArrayList с загруженными из gestures жестами
    ArrayList<Prediction> predictions =
    gLib.recognize(gesture); if (predictions.size() > 0) {
        //если загружен хотябы один жест из gestures Prediction
    prediction = predictions.get(0);
        if (prediction.score > 1.0) {
            if (prediction.name.equals(«one»))
                tvOut.setText(«1»);
            else if (prediction.name.equals(«stop»))
                tvOut.setText(«stop»);
            else if (prediction.name.equals(«two»))
                tvOut.setText(«2»);
            }else{
                tvOut.setText(«Жест неизвестен»);
            }
        }
    }
```

В приложении всего лишь распознаются жесты и в информационное поле выводится информация о том, что за жест был использован. В листинге 1 представлен возможный код приложения.

Содержание отчета и его форма

Отчет к лабораторной работе оформляется в виде текстового документа по форме простого реферата и должен включать:

1. Название лабораторной работы.
2. Цель лабораторной работы.
3. Краткие выводы по результатам выполнения лабораторной работы согласно пунктам раздела «Методики выполнения работы».
4. Ответы на Вопросы к лабораторной работе.
5. Формулировку задания для самостоятельной работы и результат его выполнения.

Задания для самостоятельной работы

1) В качестве практического задания предлагаем реализовать жестами ввод чисел в приложении «Угадайка», разработанном в лабораторной работе второй темы. Создать жесты «0», «1», «2», «3», «4», «5», «6», «7», «8», «9» для ввода цифр и жест «S» для остановки ввода числа. В приложение добавить распознавание этих жестов, преобразование их в число и сравнение полученного числа с загаданным.

2) разработать простой калькулятор с жестовым вводом чисел и операций;

3) разработать блокнотик для заметок с рукописным вводом текста.

```
package com.example.lab5_2_gestures;      import
java.util.ArrayList;

import android.os.Bundle;
import android.app.Activity;
import android.gesture.Gesture;
import android.gesture.GestureLibraries;
import android.gesture.GestureLibrary;
import android.gesture.GestureOverlayView;
```

```

import
android.gesture.GestureOverlayView.OnGesturePerformedList
ener;

import          android.gesture.Prediction;          import
android.view.Menu;

import android.widget.TextView;

public class Gestures extends Activity implements
OnGesturePerformedListener{ GestureLibrary gLib;
GestureOverlayView gestures;
TextView tvOut;

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_gestures);
tvOut=(TextView)findViewById(R.id.textView1);
//Загрузка жестов (gestures) из res/raw/gestures
gLib      =      GestureLibraries.fromRawResource(this,
R.raw.gestures);
if (!gLib.load()) {
//Если жесты не загружены, то выход из приложения
finish();
}

gestures              =              (GestureOverlayView)
findViewById(R.id.gestureOverlayView1);
gestures.addOnGesturePerformedListener(this);
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
// Inflate the menu; this adds items to the action bar
if              it              is              present.
getMenuInflater().inflate(R.menu.gestures, menu);

```



```

        return true;
    }

    public void onGesturePerformed(GestureOverlayView
overlay, Gesture gesture)
    {
        //Создаёт ArrayList с загруженными из gestures жестами
        ArrayList<Prediction> predictions =
gLib.recognize(gesture); if (predictions.size() > 0) {
            //если загружен хотя бы один жест из gestures
            Prediction prediction = predictions.get(0); if
(prediction.score > 1.0) {
                if (prediction.name.equals («one»))
tvOut.setText («1»);
                else if (prediction.name.equals («stop»))
tvOut.setText («stop»);
                else if (prediction.name.equals («two»))
tvOut.setText («2»);
                }else{
                    tvOut.setText («Жест неизвестен»);
                }
            }
        }
    }
}

```

Вопросы к лабораторной работе

1. Технология создания набора жестов.
2. Технология использования созданных жестов в приложении.
3. Технология распознавания жестов загруженных в файл
res/raw/gestures.

Лабораторная работа 12

Многооконное приложение

Цель работы: разработка многооконного приложения, предоставляющего возможности: воспроизведения аудио и видео файлов, создания и отображения фотоснимков.

Теоретическое обоснование

Для достижения цели, поставленной в лабораторной работе, сформулируем требования к разрабатываемому приложению, назовем его «Media».

Приложение предоставляет пользователю возможность выбора рода деятельности:

- 1) работа с камерой для создания снимков;
- 2) воспроизведение аудио и видео;
- 3) просмотр изображений.

В приложении предполагается реализовать четыре активности:

- 4) главная активность, предназначена для выбора рода деятельности, содержит три кнопки, нажатие на каждую кнопку вызывает к жизни соответствующую активность;
- 5) активность для работы с камерой и создания снимков;
- 6) активность для воспроизведения аудио и видео;
- 7) активность для просмотра изображений.

Задания к лабораторной работе

Задание 1. Создание приложения

Создадим новое приложение, активность, полученную при создании проекта, оставим с именем **MainActivity**, она будет главной активностью приложения. Добавим в приложение еще три активности: **New->Other...->Android Activity**.

В нашем приложении созданные активности имеют следующие имена:

CameraActivity для работы с камерой и создания снимков;

MediaActivity для воспроизведения видео и аудио;

GalleryActivity для просмотра изображений.

Далее продолжим работу с этими активностями, настроим интерфейс и реализуем логику для каждой из них. Начнем с активностей, отвечающих за тот или иной вид деятельности, главную активность приведем в порядок в самом конце работы.

Задание 2. Настройка интерфейса и реализация логики активности для работы с камерой

Настроим интерфейс активности для работы с камерой. Нам понадобится окно предварительного просмотра, добавим в окно активности элемент **SurfaceView**. А так же нам понадобится кнопка для выполнения снимков, добавим в окно активности элемент **ImageButton**. Далее предлагаем настроить интерфейс самостоятельно. В нашем случае активность выглядит так, как показано на рисунке 12.1.

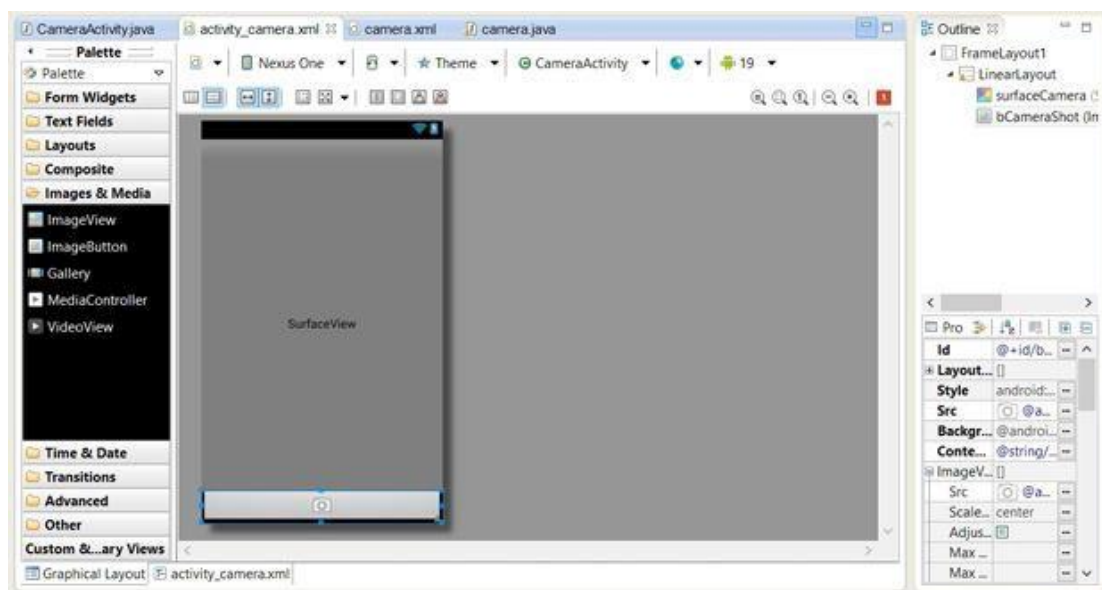


Рисунок 12.1– Интерфейс активности для работы с камерой

Реализуем логику активности, в данном случае необходимо при нажатии на кнопку выполнить снимок. Работаем с **java** файлом, описывающим соответствующий класс активности, **CameraActivity.java**.

Объявим следующие поля класса активности:

```
private Camera camera; //для проведения всех операций  
с камерой private SurfaceHolder surfaceHolder; //для
```

задания preview

```
private SurfaceView preview; //для отображения окна  
предпросмотра
```

```
private View shotBtn; //для выполнения снимка (кнопка)
```

Для начала работы с камерой необходимо ее инициализировать, сделать это лучше в методе onResume() класса активности, для инициализации используется следующая конструкция:

```
camera = Camera.open();
```

После завершения работы с камерой, необходимо ее освободить для других приложений, сделать это лучше в методе onPause() класса активности, для освобождения камеры используется следующая конструкция:

```
Camera.release();
```

Для активности, работающей с камерой, имеет смысл сразу задать расположение экрана следующим образом
setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);

в противном случае, приходится, например, в методе surfaceCreated() проверять расположение экрана и поворачивать окно предварительного просмотра, что не очень удобно, так как поворот экрана занимает некоторое время.

Выполним настройку окна предварительного просмотра:

```
preview = (SurfaceView)  
findViewById(R.id.surfaceCamera); surfaceHolder =  
preview.getHolder(); surfaceHolder.addCallback(new  
MyCallback());
```

метод addCallback() используется для отслеживания изменений окна предпросмотра. Параметром этого метода служит экземпляр класса реализующего интерфейс SurfaceHolder.Callback, в нашем случае для реализации этого интерфейса создан внутренний класс активности MyCallback.

В этом классе необходимо реализовать методы:

```
public void surfaceCreated(SurfaceHolder holder);
```

```

    public void surfaceChanged(SurfaceHolder holder, int
format,
    int width, int height);
    public void surfaceDestroyed(SurfaceHolder holder);

```

Мы реализуем метод `surfaceCreated()`, оставшиеся два метода запишем с пустой реализацией.

В методе `surfaceCreated()` зададим окно предварительного просмотра:

```
camera.setPreviewDisplay(holder);
```

Выполним необходимые настройки окна предпросмотра (см. Листинг 21-22.1). Запустим отображение окна предварительного просмотра:

```
camera.startPreview();
```

Пришло время обсудить и реализовать выполнение фотосъемки. Для того, чтобы сделать снимок необходимо вызвать метод `takePicture()`. Вызов этого метода можно выполнить из метода-обработчика события нажатия кнопки, тогда при нажатии на кнопку сразу будет получена фотография, а можно «растянуть удовольствие» и воспользоваться предварительной автофокусировкой.

Сначала инициализируем кнопку и добавим слушателя события нажатия в методе `onCreate()` активности:

```

shotBtn          =          findViewById(R.id.bCameraShot);
shotBtn.setOnClickListener(new MyViewListener());

```

В данном случае слушателем является экземпляр внутреннего класса `MyViewListener`, который реализует интерфейс `View.OnClickListener`.

```

class MyViewListener implements View.OnClickListener{
    @Override
    public void onClick(View v){
        if (v == shotBtn){
            camera.autoFocus(new MyAutoFocusCallback());
        }
    }
}

```

В методе `onClick()` вызываем обработчик автофокуса, слушателем события автофокусировки в данном случае является экземпляр внутреннего класса `MyAutoFocusCallback`, который реализует интерфейс `Camera.AutoFocusCallback`.

```
class MyAutoFocusCallback implements
Camera.AutoFocusCallback{
    @Override
    public void onAutoFocus(boolean paramBoolean, Camera
paramCamera){
        if (paramBoolean){
            // если удалось сфокусироваться, делаем снимок
            paramCamera.takePicture(null, null, null,
            new MyPictureCallback());
        }
    }
}
```

Метод `takePicture()` имеет четыре параметра:

Camera.ShutterCallback Shutter	вызывается в момент получения изображения с матрицы;
Camera.PictureCallback Raw	программе передаются для обработки raw данные (если поддерживается аппаратно);
Camera.PictureCallback Postview	программе передаются полностью обработанные данные (если поддерживается аппаратно);
Camera.PictureCallback jpg	программе передается изображение в формате jpg. Здесь может быть организована запись изображения на карту памяти.

В нашем случае последним параметром является экземпляр класса

MyPictureCallback(), который реализует интерфейс Camera.PictureCallback. В этом классе реализован единственный метод (см. Листинг 21-22.1):

```
public void onPictureTaken(byte[] paramArrayOfByte,
Camera paramCamera){...}
```

В листинге 1 представлен код активности, реализующей работу с камерой и позволяющей выполнять снимки.

Чтобы разрешить приложению работать с камерой и сохранять фотографии на карте памяти, в манифест необходимо добавить следующие разрешения:

```
<uses-permission
android:name=«android.permission.CAMERA» />
<uses-permission
android:name=«android.permission.WRITE_EXTERNAL_STORAGE»
/>
```

Задание 3. Настройка интерфейса и реализация логики активности для воспроизведения аудио и видео

Настроим интерфейс активности для воспроизведения аудио и видео. Нам понадобится окно предварительного просмотра, добавим в окно активности элемент SurfaceView. Для работы приложения необходимо поле для задания расположения медиа контента для проигрывания, добавим элемент EditText, а также потребуются несколько кнопок, добавим в окно активности следующие кнопки:

b_Start	для запуска воспроизведения контента с начала, зададим свойству On Click этой кнопки значение onClickStart;
b_Pause	для приостановки воспроизведения, зададим свойству On Click этой кнопки значение onClick;

b_Resume	для продолжения воспроизведения с места приостановки, зададим свойству On Click этой кнопки значение onClick;

b_Stop для полной остановки воспроизведения, зададим свойству On Click этой кнопки значение onClick.

Для возможности выбора зацикленного воспроизведения медиа контента добавим в окно активности элемент CheckBox.

Далее предлагаем настроить интерфейс самостоятельно. В нашем случае активность выглядит так, как показано на рисунке 12.2.

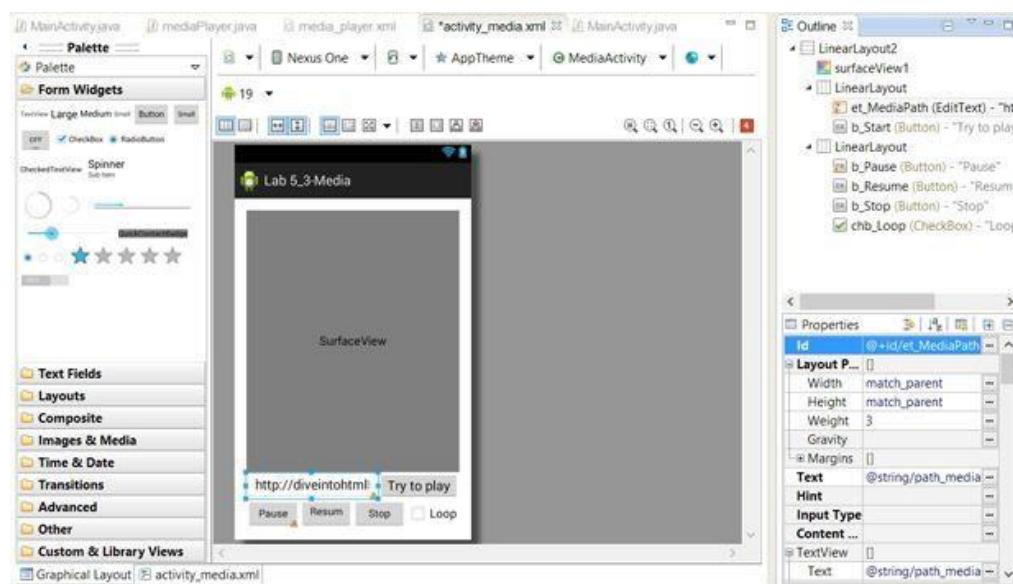


Рисунок 12.2 – Интерфейс активности для воспроизведения аудио и видео
Реализуем логику активности, в данном случае необходимо выполнять следующие действия:

1. при нажатии на кнопку **Try to play**, запускается воспроизведение с начала контента, расположенного по адресу, указанному в поле ввода, расположенном слева от кнопки;
2. при нажатии на кнопку **Pause**, воспроизведение контента приостанавливается;
3. при нажатии на кнопку **Resume**, воспроизведение контента продолжается с места приостановки;

4. при нажатии на кнопку **Stop**, воспроизведение контента останавливается и может быть возобновлено только с начала, т. е. необходимо снова нажать **Try to play**.

Элемент Loop управляет возможностью повтора воспроизведения.

Работать будем с файлом **MediaActivity.java**, описывающим соответствующий класс активности.

Объявим следующие поля класса активности:

```
MediaPlayer mediaPlayer; //управляет воспроизведением
CheckBox chbLoop; //управляет режимом повтора
```

В методе onCreate() активности настроим чек-бокс так, чтобы он включал/выключал режим повтора для плеера.

```
chbLoop = (CheckBox) findViewById(R.id.chb_Loop);
chbLoop.setOnCheckedChangeListener(new
    OnCheckedChangeListener() {
    @Override
    public void onCheckedChanged(CompoundButton
buttonView,
    boolean isChecked) {
    if (mediaPlayer != null)

    });
mediaPlayer.setLooping(isChecked);
```

Метод onClickStart() используется для обработки нажатий на кнопку **Try to play**. В этом метода сначала освобождаем ресурсы текущего проигрывателя, используя вызов метода: `releaseMP()`;

Этот метод рекомендуется вызывать по окончании использования плеера, а также при onPause/onStop, если нет острой необходимости держать объект.

Реализация метода releaseMP():

```
private void releaseMP() {
    if (mediaPlayer != null) {
        try {
            mediaPlayer.release();
            mediaPlayer = null;
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Продолжим рассмотрение метода `onClickStart()`. В нем после освобождения ресурсов, занимаемся подготовкой плеера к воспроизведению.

```
mediaPlayer = new MediaPlayer();
mediaPlayer.setDataSource(DATA);
mediaPlayer.setDisplay(((SurfaceView)
    findViewById(R.id.surfaceView1)).getHolder());
mediaPlayer.setOnPreparedListener(this);
mediaPlayer.prepareAsync();
```

Создаем новый объект класса `MediaPlayer`.

Используем метод `setDataSource()` для задания источника данных, в качестве параметра передаем строку из поля ввода, сохраненную в переменной `DATA`.

Используя метод `setDisplay()`, задаем экран для воспроизведения, в нашем случае элемент

`SurfaceView`, добавленный в окно активности на этапе формирования интерфейса.

Метод `prepareAsync()` выполняет асинхронную подготовку плеера к воспроизведению и когда подготовка будет завершена сообщает об этом слушателю, указанному в методе `setPreparedListener()`. В нашем случае слушателем является сам класс активности, для этого он объявлен, как класс реализующий интерфейс `OnPreparedListener`. В случае готовности плеера

вызывается метод `onPrepared(MediaPlayer mp)`, объявленный в указанном интерфейсе и реализованный в классе активности, в этом методе выполняется запуск воспроизведения:

```
mp.start();
```

Существует еще метод `prepare()`, он также выполняет подготовку плеера, но в синхронном режиме. Для прослушивания файлов из интернета, необходимо использовать асинхронный режим, что мы и сделали.

Последние две строки метода `onClickStart()`:

```
mediaPlayer.setLooping(chbLoop.isChecked());  
mediaPlayer.setOnCompletionListener(this);
```

В первой из них определяется возможность циклического воспроизведения в зависимости от значения чек-бокса.

Во второй задается слушатель для получения сообщения о достижении конца воспроизводимого контента. В нашем случае этим слушателем будет сам класс активности для этого он объявлен, как класс реализующий интерфейс `OnCompletionListener`. Полную версию метода `onClickStart()` можно найти в листинге 2.

Метод `onClick()` вызывается при нажатии на любую кнопку: **Pause**, **Resume**, **Stop**. В этом методе выполняется проверка какая кнопка была нажата и после этого выполняются соответствующие действия.

Если нажата кнопка **Pause**, выполняется следующая конструкция:

```
if (mediaPlayer.isPlaying()) mediaPlayer.pause();
```

Если нажата кнопка **Release**, выполняется следующая конструкция:

```
if (!mediaPlayer.isPlaying()) mediaPlayer.start();
```

Если нажата кнопка **Stop**, выполняется следующая конструкция:

```
mediaPlayer.stop();
```

Имеет смысл обратить внимание на метод активности:

```
protected void onDestroy() { super.onDestroy();  
releaseMP();  
}
```

В этом методе обязательно необходимо освободить ресурсы, что и

выполняется вызовом метода `releaseMP()`.

Полный код класса `MediaActivity` представлен в листинге 2.

Задание 4. Настройка интерфейса и реализация логики активности для просмотра изображений

Настроим интерфейс активности для просмотра изображений. Нам потребуется элемент для просмотра изображений, добавим в окно активности элемент `ImageView`. Добавим информационное поле, т. е. элемент `TextView`, для отображения информации об общем количестве изображений в папке и номере просматриваемого изображения. Добавим две кнопки, для перемещения от одного изображения к другому вперед и назад.

Предлагаем настроить интерфейс самостоятельно. В нашем случае активность выглядит так, как показано на рисунке 12.3. Но это, разумеется, не единственно возможный вариант.

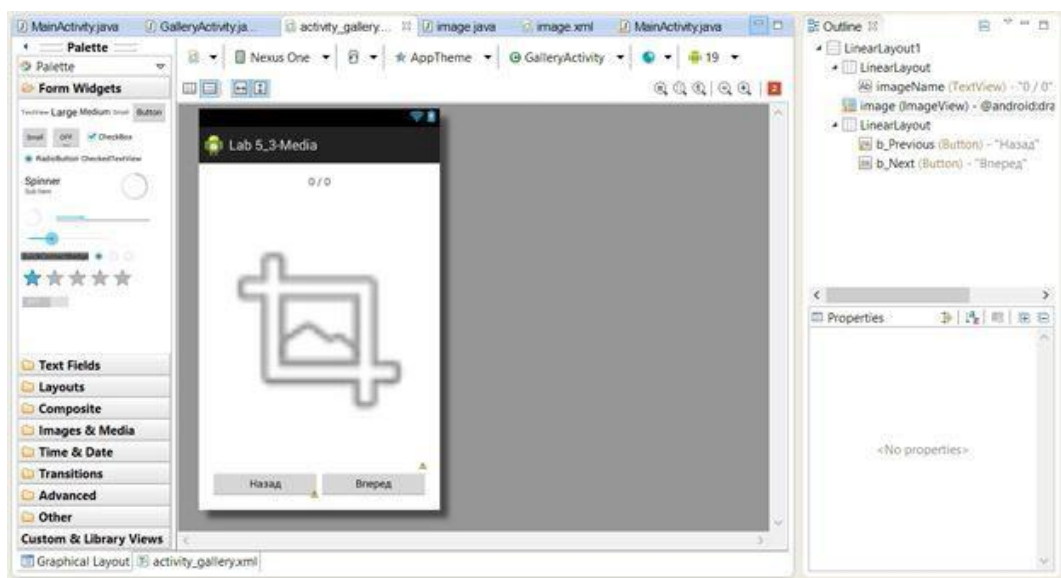


Рисунок 12.3 – Интерфейс активности для просмотра изображений

Реализуем логику активности, данная активность ориентирована на просмотр снимков, сделанных в этом же приложении и сохраненных в папке: `/sdcard/TrainingMedia/`. Кнопка **Назад** выводит в окно просмотра предыдущий снимок, по отношению к уже отображенному, зададим свойству `On Click` этой кнопки значение `onPrevious`. Кнопка **Вперед** выводит в окно просмотра следующий снимок, по отношению к уже отображенному, зададим свойству `On Click` этой кнопки значение `onNext`.

Работать будем с файлом **GalleryActivity.java**, описывающим класс соответствующей активности.

Объявим поля класса активности:

```
int    currentImage=0;    ArrayList<String>    images;
ImageView imageView; TextView nameView;
```

Настройку основных элементов для вывода изображений на экран выполним в методе `onResume()` активности, этот метод вызывается каждый раз, перед выводом активности на передний план (см. жизненный цикл активности в «Виды приложений и их структура»). `images=new ArrayList<String>()`;

```
imageView=((ImageView)findViewById(R.id.image)); try{
    File    imagesDirectory=new
File(«/sdcard/TrainingMedia/»);
images=searchImage(imagesDirectory);
updatePhoto(Uri.parse(images.get(currentImage)) );
} catch (Exception e) {
    nameView.setText(«Ошибка: Папка
'/sdcard/TrainingMedia/' не
найдена»);
}
```

Для получения списка изображений в переменную `images` используется метод `searchImage()`, этот метод, используя переданный в него в качестве параметра адрес директории с изображениями, находит файл с расширением `.jpg`, `.png` или `.jpeg` и добавляет его к списку изображений. Метод возвращает список доступных файлов с изображениями. Код метода представлен в листинге 3.

Метод `updatePhoto()` выполняет обновление счетчика фотографий в информационном поле и выводит на экран изображение, соответствующее переданному в метод URI. Код метода представлен в листинге 3.

В методе `onPause()` активности выполняется очистка списка изображений и освобождение памяти. Этот метод вызывается каждый раз, как активность

теряет фокус ввода. Код метода представлен в листинге 3.

Осталось рассмотреть еще два метода: `onPrevious()` и `onNext()`.

Первый метод вызывается, когда нажата кнопка **Назад**. В нем уменьшается номер текущего изображения и вызывается метод `updatePhoto()`.

Второй метод вызывается, когда нажата кнопка **Вперед**. В нем увеличивается номер, текущего изображения и вызывается метод `updatePhoto()`.

Полный код класса `GalleryActivity` представлен в листинге 3.

Задание 5. Настройка интерфейса и реализация логики главной активности приложения

Настроим интерфейс главной активности приложения. Эта активность содержит три кнопки (**Image Button**):

bCamera	для вызова активности, предоставляющей возможности работы с камерой;
bGallery	для вызова активности, предоставляющей возможности просмотра изображений;
bMusic	для вызова активности, предоставляющей возможности воспроизведения аудио и видео.

Предлагаем настроить интерфейс самостоятельно. В нашем случае активность выглядит так, как показано на рисунке 12.4. Но это, разумеется, не единственно возможный вариант.

Реализуем логику главной активности. В данном случае достаточно настроить обработку событий нажатия на кнопки, таким образом, чтобы при нажатии на кнопку запускалась соответствующая активность. Работаем с `java` файлом, описывающим класс активности.

Создадим переменную `btnClick`, как реализацию интерфейса-слушателя `OnClickListener`:

```
OnClickListener btnClick=new OnClickListener() {  
    @Override  
    public void onClick(View v) {
```

```
Click(v.getId());
}
};
```

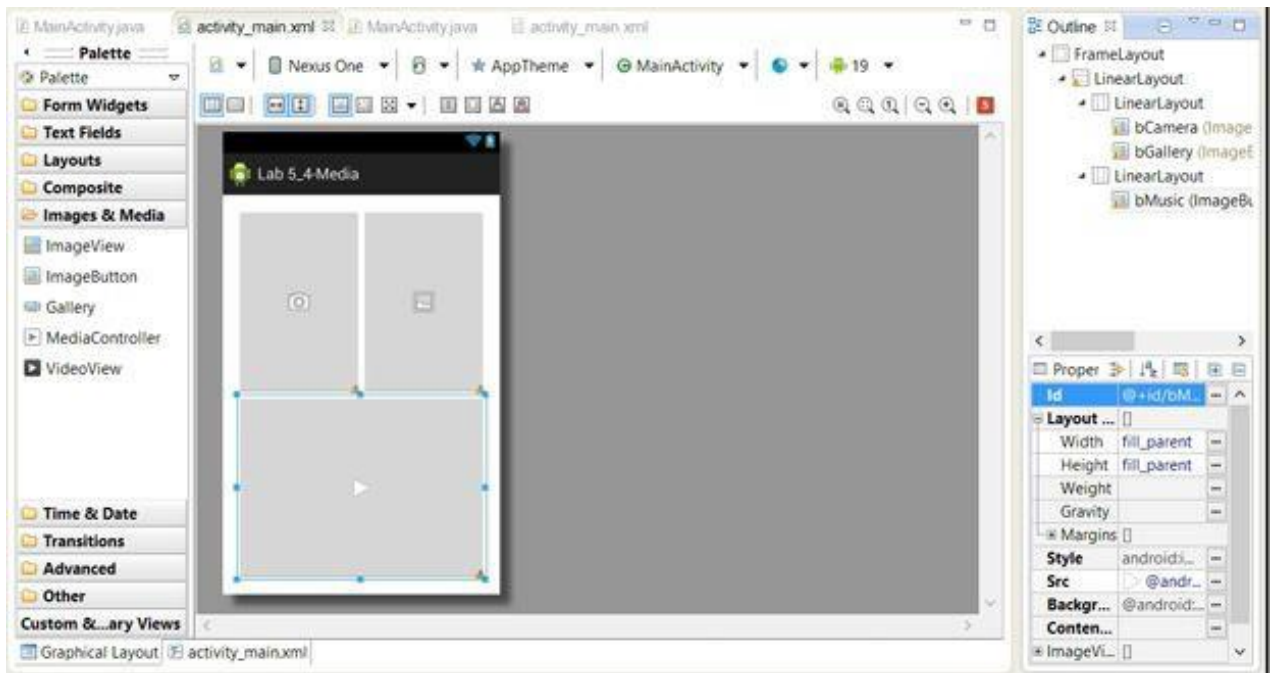


Рисунок 12.4 – Интерфейс главной активности приложения

Добавим слушателя события нажатия к каждой кнопке активности:

```
((ImageButton)findViewById(R.id.bMusic)).setOnClickListener(
btnClick);
((ImageButton)findViewById(R.id.bCamera)).setOnClickListener(
btnClick);
((ImageButton)findViewById(R.id.bGallery)).setOnClickListener(
btnClick);
```

В методе `onClick()` слушателя вызываем метод `Click()`, в котором выполняется запуск соответствующей активности:

```
protected void Click(int view){
    Intent intent=null; switch (view){
        case R.id.bMusic: intent=new
Intent(this,mediaPlayer.class);
        break;
        break; break;
```

```

    }

    case R.id.bGallery: intent=new
Intent(this,GalleryActivity.class); case R.id.bCamera:
intent=new Intent(this,CameraActivity.class); default:
break;

    if(intent!=null){
startActivity(intent);
    }
}

```

В листингах 1 - 4 представлен код главной активности.

Листинг 1. Класс CameraActivity для работы с камерой

```

package com.example.lab5_4_media;
import java.io.File;
import java.io.FileOutputStream;
import          java.io.IOException;          import
android.hardware.Camera;
import          android.hardware.Camera.Size;      import
android.os.Bundle;
import android.app.Activity;
import          android.content.pm.ActivityInfo;      import
android.content.res.Configuration;          import
android.util.Log;
import          android.view.SurfaceHolder;          import
android.view.SurfaceView;
import android.view.View; import android.view.Window;
import android.view.WindowManager;
import android.view.ViewGroup.LayoutParams;
public class CameraActivity extends Activity { private
Camera camera;
private SurfaceHolder surfaceHolder;
private SurfaceView preview; private View shotBtn;

```



```

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // если хотим, чтобы приложение постоянно имело
портретную                                     ориентацию
    setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_L
ANDSCAPE);
    // если хотим, чтобы приложение было полноэкранным
getWindow().addFlags(WindowManager.LayoutParams.FLAG_FULL
SCREEN);
    //                                     и                                     без                                     заголовка
requestWindowFeature(Window.FEATURE_NO_TITLE);

    setContentView(R.layout.activity_camera);
    preview                                     =                                     (SurfaceView)
findViewById(R.id.surfaceCamera);    surfaceHolder =
preview.getHolder();    surfaceHolder.addCallback(new
MyCallback(this));
    surfaceHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH
_BUFFERS);
    shotBtn = findViewById(R.id.bCameraShot);
shotBtn.setOnClickListener(new MyViewListener());
}
@Override
protected void onResume() {
    super.onResume();
    camera = Camera.open();
}
@Override
protected void onPause() {
    super.onPause();
}

```

```

        if (camera != null){
            camera.setPreviewCallback(null);
camera.stopPreview(); camera.release();
            camera = null;
        }
    }

    class MyCallback implements SurfaceHolder.Callback{
Activity host;
        MyCallback(Activity act){
            host=act;
        }

        @Override
        public void surfaceChanged(SurfaceHolder holder, int
format, int width,
            int height){}

        @Override
        public void surfaceCreated(SurfaceHolder holder){ try
{
            camera.setPreviewDisplay(holder);
camera.setPreviewCallback(new MyPreviewCallback());
        }

        catch (IOException e){ Log.d(«myLogs»,»Ошибка
камеры»); e.printStackTrace();
        }

        Size previewSize =
camera.getParameters().getPreviewSize();
        float aspect = (float) previewSize.width /
previewSize.height;

        int previewSurfaceWidth = preview.getWidth(); int
previewSurfaceHeight = preview.getHeight();
        LayoutParams lp = preview.getLayoutParams();

```

```

        // здесь корректируем размер отображаемого preview,
чтобы не было иска
        if
(host.getResources().getConfiguration().orientation    !=
Configuration.ORIENTATION_LANDSCAPE) {
        // портретный вид camera.setDisplayOrientation(90);
lp.height = previewSurfaceHeight;
        lp.width = (int) (previewSurfaceHeight / aspect);
        }
        else {
        // ландшафтный camera.setDisplayOrientation(0);
lp.width = previewSurfaceWidth;
        lp.height = (int) (previewSurfaceWidth / aspect);
        }
        preview.setLayoutParams(lp);
        camera.startPreview();
        }
@Override
public void surfaceDestroyed(SurfaceHolder holder){}
}
class MyViewListener implements View.OnClickListener{
@Override
public void onClick(View v) { if (v == shotBtn) {
// либо делаем снимок непосредственно здесь
// либо включаем обработчик автофокуса
//camera.takePicture(null, null, null, this);
camera.autoFocus(new MyAutoFocusCallback());
}
}
}
class MyAutoFocusCallback implements

```

```

Camera.AutoFocusCallback{
    @Override
    public void onAutoFocus(boolean paramBoolean, Camera
paramCamera){ if (paramBoolean){
        // если удалось сфокусироваться, делаем снимок
paramCamera.takePicture(null, null, null, new
MyPictureCallback());
    }
}

class MyPictureCallback implements
Camera.PictureCallback{
    @Override
    public void onPictureTaken(byte[] paramArrayOfByte,
Camera paramCamera){
        // сохраняем полученные jpg в папке
/sdcard/CameraExample/
        // имя файла System.currentTimeMillis()
        try {
            File saveDir = new File («/sdcard/CameraExample/»); if
(!saveDir.exists()) {
                saveDir.mkdirs();
            }
            FileOutputStream os = new
FileOutputStream(String.format («/sdcard/CameraExample/%d.
jpg», System.currentTimeMillis()));
            os.write(paramArrayOfByte); os.close();
        }
        catch (Exception e) {}
        // после того, как снимок сделан, показ превью
отключается.

```

```

        // Необходимо включить его
        paramCamera.startPreview();
    }
}

class MyPreviewCallback implements
Camera.PreviewCallback{
    @Override
    public void onPreviewFrame(byte[] paramArrayOfByte,
Camera paramCamera) {
        // здесь можно обрабатывать изображение, показываемое
в preview
    }
}
}

```

Листинг 2. Класс MainActivity

```

package com.example.lab5_4_media; import
android.media.MediaPlayer;
import
android.media.MediaPlayer.OnCompletionListener;
import android.media.MediaPlayer.OnPreparedListener;
import android.os.Bundle;
import android.view.Gravity; import
android.view.SurfaceView; import android.view.View;
import android.widget.CheckBox; import
android.widget.CompoundButton;
import android.widget.EditText; import
android.widget.Toast;
import
android.widget.CompoundButton.OnCheckedChangeListener;
import android.app.Activity;
    public class MainActivity extends Activity implements

```

```

OnPreparedListener, OnCompletionListener{
    MediaPlayer mediaPlayer; CheckBox chbLoop;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_media);
        chbLoop = (CheckBox) findViewById(R.id.chb_Loop);
        chbLoop.setOnCheckedChangeListener(new
        OnCheckedChangeListener() {
            @Override
            public void onCheckedChanged(CompoundButton
            buttonView, boolean isChecked) {
                if (mediaPlayer != null)
                    mediaPlayer.setLooping(isChecked);
            }
        });
        public void onClickStart(View view) { releaseMP();
        String
        DATA=((EditText)findViewById(R.id.et_MediaPath)).getText(
        ).toString();
        try {
            mediaPlayer = new MediaPlayer();
            mediaPlayer.setDataSource(DATA);
            mediaPlayer.setDisplay(((SurfaceView)
            findViewById(R.id.surfaceView1)).getHolder());
            //mediaPlayer.setAudioStreamType(AudioManager.STREAM_
            MUSIC);
            mediaPlayer.setOnPreparedListener(this);
            mediaPlayer.prepareAsync();
        } catch (Exception e) { showMessage («Ошибка

```

```

воспроизведения»);
    }
    if (mediaPlayer == null) return;
    mediaPlayer.setLooping(chbLoop.isChecked());
mediaPlayer.setOnCompletionListener(this);
    }
    private void showMessage(String text){
        Toast toast = Toast.makeText(getApplicationContext(),
text,
        Toast.LENGTH_SHORT);
        toast.setGravity(Gravity.CENTER, 0, 0);
        toast.show();
    }
    private void releaseMP() { if (mediaPlayer != null) {
        try {
            mediaPlayer.release();
            mediaPlayer = null;
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

    public void onClick(View view) { if (mediaPlayer ==
null)
        return;
        switch (view.getId()) {
        case R.id.b_Pause:
            if (mediaPlayer.isPlaying())
                mediaPlayer.pause(); break;
        case R.id.b_Resume:
            if (!mediaPlayer.isPlaying())

```

```

mediaPlayer.start();
break;
case R.id.b_Stop:
mediaPlayer.stop();
break;
}
}
@Override
public void onPrepared(MediaPlayer mp) {
mp.start();
}
@Override
public void onCompletion(MediaPlayer mp) {}
@Override
protected void onDestroy() {
super.onDestroy();
releaseMP();
}

}

```

Листинг 3. Класс GalleryActivity

```

package com.example.lab5_4_media; import java.io.File;
import java.util.ArrayList;
import android.net.Uri;
import android.os.Bundle; import android.app.Activity;
import android.util.Log; import android.view.View;
import android.widget.ImageView;
import android.widget.TextView;
public class GalleryActivity extends Activity { int
currentImage=0;
    ArrayList<String>    images;    ImageView    imageView;

```



```

TextView nameView;

@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_gallery);
}

@Override
public void onResume() {
super.onResume();
currentImage=0;
Log.d(«myLogs»,»onResume cI=«+currentImage);
nameView=( (TextView) findViewById(R.id.imageName));
images=new ArrayList<String>();

imageView=( (ImageView) findViewById(R.id.image)); try{
File imagesDirectory=new
File(«/sdcard/TrainingMedia/»);
images=searchImage(imagesDirectory);
updatePhoto(Uri.parse(images.get(currentImage)));
}catch(Exception e){
nameView.setText(«Ошибка: Папка
'/sdcard/TrainingMedia/' не найдена»);
Log.d(«myLogs»,»Ошибка»);
}
}

@Override
protected void onPause()
{
super.onPause();
images.clear();
Log.d(«myLogs»,»onPause cI=«+currentImage);

```

```

    }

    private ArrayList<String> searchImage(File dir){
ArrayList<String> imagesFinded=new ArrayList<String>();
for(File f:dir.listFiles()){
    if(!f.isDirectory()){
        String fileExt=getFileExt(f.getAbsolutePath());
        if(fileExt.equals(«png») || fileExt.equals(«jpg») ||
fileExt.equals(«jpeg»)){
            Log.d(«myLogs»,»Файл найден «+f.getAbsolutePath());
            imagesFinded.add(f.getAbsolutePath());
        }
    }
}
return imagesFinded;
}

public static String getFileExt(String filename){
    return filename.substring(filename.lastIndexOf(«.») +
1);
}

public void updatePhoto(Uri uri){ try{
    nameView.setText((currentImage+1)+»/»+images.size());
imageView.setImageURI(uri);
} catch (Exception e){ nameView.setText(«Ошибка загрузки
файла»);
}
}

public void onNext(View v){
if(currentImage+1<images.size() && images.size()>0){
    currentImage++;
updatePhoto(Uri.parse(images.get(currentImage)));
}
}

```

```

    }

    public void onPrevious(View v){ if(currentImage>0 &&
images.size())>0){
        currentImage--;
updatePhoto(Uri.parse(images.get(currentImage)));
    }
}
}
}

```

Листинг 4. Класс MainActivity

```

package          com.example.lab5_4_media;          import
android.os.Bundle;

import android.app.Activity;
import android.content.Intent;
import android.util.Log;
import android.view.Menu; import android.view.View;
import      android.view.View.OnClickListener;      import
android.widget.ImageButton;

public class MainActivity extends Activity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        OnClickListener btnClick=new OnClickListener() {
            @Override
            public void onClick(View v) {
                Log.d(«myLogs»,v.getId()+»«»); Click(v.getId());
            }
        };

        ((ImageButton)findViewById(R.id.bMusic)).setOnClickLi
stener(btnClick);

        ((ImageButton)findViewById(R.id.bCamera)).setOnClickL

```

```

istener(btnClick);
        ((ImageButton)findViewById(R.id.bGallery)).setOnClickListener
Listener(btnClick);
    }
    protected void Click(int view){
        Intent intent=null;
        Log.d(«myLogs»,view+»«);
        switch (view){
            case R.id.bMusic: intent=new
Intent(this,MediaActivity.class);
            break;
            case R.id.bGallery: intent=new
Intent(this,GalleryActivity.class);
            break;
            case R.id.bCamera: intent=new
Intent(this,CameraActivity.class); break;
            default: break;
        }
        if(intent!=null){
            Log.d(«myLogs»,»Интент = «+intent.toString());
            startActivity(intent);
        }
    }
}

```

Содержание отчета и его форма

Отчет к лабораторной работе оформляется в виде текстового документа по форме простого реферата и должен включать:

1. Название лабораторной работы.
2. Цель лабораторной работы.
3. Краткие выводы по результатам выполнения лабораторной работы

согласно пунктам раздела «Методики выполнения работы».

4. Ответы на Вопросы к лабораторной работе.
5. Формулировку задания для самостоятельной работы и результат его выполнения.

Задания для самостоятельной работы

1. Подумайте, какие особенности мобильного устройства могут повлиять на появление дополнительных возможностей в работе вашего приложения.
2. Попробуйте реализовать эти возможности.

Вопросы к лабораторной работе

1. Технология создания многооконного приложения.
2. Технология настройки интерфейса и реализация логики активности для работы с камерой.
3. Технология настройки интерфейса и реализация логики активности для воспроизведения аудио и видео.
4. Технология настройки интерфейса и реализация логики активности для просмотра изображений.
5. Технология настройки интерфейса и реализация логики главной активности приложения.

Лабораторная работа 13

Геолокационные возможности

Цель работы: разработка приложения, демонстрирующего геолокационные возможности.

Теоретическое обоснование Взаимодействие с системами позиционирования

Системы позиционирования позволяют определить местоположение в некоторой системе координат, обычно определяются широта и долгота.

Так как смартфон является мобильным телефоном, ему доступны методы, обычно используемые мобильными телефонами для определения своего местоположения.

1. Во-первых, смартфон постоянно связывается с сотовой вышкой, в зоне действия которой он находится. Каждая сотовая вышка в мире имеет уникальный идентификатор, называемый идентификатором соты (Cell ID), а также для нее точно известны широта и долгота ее расположения. В связи с этим, смартфон, зная идентификатор соты, в которой он находится, может получить географические координаты центра этой соты.

Радиусы сот варьируются в зависимости от того, насколько активный сетевой трафик ожидается в конкретном районе. Разумеется, такой способ позиционирования даст очень приблизительные результаты, что называется: «плюс-минус трамвайная остановка».

Во-вторых, чаще всего смартфон оказывается в зоне действия более, чем одной сотовой вышки. В современных мобильных технологиях, начиная с поколения 2G, сотовая вышка может определить, с какого направления приходит сигнал. В случае, когда телефон находится в зоне действия двух или трех сотовых вышек, они могут выполнять триангуляцию его местоположения. Телефон может запросить у сети информацию о том, где он находится. Такая техника определения местоположения может быть очень точной и не требует установки дополнительного оборудования.

Дополнительно к возможностям определения местоположения, доступным обычным мобильным телефонам, большинство смартфонов

укомплектованы спутниковыми системами глобального позиционирования (Global Positioning System, GPS). В настоящее время наиболее распространенными в мире системами глобального спутникового позиционирования являются: GPS, разработанная и реализованная в США, и система ГЛОНАСС (Глобальная навигационная спутниковая система), советская, а позже российская спутниковая система навигации. Многие смартфоны могут использовать сигналы сразу от двух навигационных систем, что позволяет серьезно увеличить надежность и точность определения координат, прежде всего, в городских условиях.

В дополнение к вышеперечисленным методам позиционирования, добавляется возможность использования сигналов WiFi, Bluetooth и NFC, а также внутреннего сенсора для более точной геолокации, особенно внутри помещений.

В этом разделе нас, в первую очередь, будет интересовать возможность добавления в приложения способностей определять координаты устройства и работать с картами. При создании приложений, учитывающих текущее местоположение, под Android можно воспользоваться GPS и определением местоположения в сети (с помощью Network Location Provider). Несмотря на то, что GPS дает более точные результаты, он не очень хорошо работает в помещениях (чаще не работает), он сильно расходует заряд батареи и скорость определения координат не всегда соответствует ожиданиям пользователя. Network Location Provider определяет координаты, используя сигналы сотовых вышек и WiFi, может работать как на улице, так и внутри помещений, более экономно расходует заряд батареи и работает быстрее по сравнению с GPS. Для получения координат в приложении можно использовать оба способа или один из них на выбор.

Android предоставляет приложениям доступ к геолокационным возможностям мобильного устройства, через классы пакета `android.location`. Центральным классом этого пакета является класс `LocationManager`, который предоставляет доступ к системным сервисам для определения координат устройства.

В приложения можно добавлять карты, используя Google Maps Android API, которое автоматически управляет доступом к серверам Google Maps, загрузкой данных, отображением карт и сенсорными жестами на карте. Также можно использовать вызовы API для добавления маркеров, многоугольников и внешних прозрачных слоев, а также для изменения пользовательского представления отдельных участков карты.

Ключевым классом в Google Maps Android API является класс `MapView`, который отображает карту с данными полученными из сервиса Google Maps. Когда `MapView` имеет фокус, он может перехватывать нажатия клавиш и сенсорные жесты для выполнения автоматического перемещения и изменения масштаба карты, а также может управлять сетевыми запросами для получения дополнительных фрагментов карты. Этот класс так же предоставляет все элементы пользовательского интерфейса, необходимые для управления картой.

Google Maps Android API не является частью платформы Android, но доступен на любом устройстве с Google Play Store, работающем, начиная с Android 2.2, через Google Play services. Чтобы обеспечить возможность интеграции Google Maps в приложения, в Android SDK необходимо установить библиотеку Google Play services.

Подробнее вопросы добавления в приложения геолокационных возможностей и использование карт (Google Maps) рассмотрены в четвертой части лабораторной работы к данной теме.

Другие сенсоры и датчики

Большинство устройств, работающих под управлением Android, укомплектованы встроенными сенсорами, которые предоставляют исходные данные высокой точности. Сенсоры

могут быть полезны в том случае, если необходимо регистрировать положение и перемещения, повороты устройства в трехмерном пространстве, а также изменения параметров окружающей среды.

Платформа Android поддерживает три категории сенсоров:

Датчики движения	Эти сенсоры измеряют силы ускорения и вращательные силы по трем осям. Эта категория включает акселерометры, гироскопы, датчики вектора вращения и сенсоры силы тяжести.
Датчики окружающей среды	Эти сенсоры измеряют различные параметры окружающей среды, такие как температура воздуха и давление, освещенность и влажность. Эта категория включает барометры, термометры и датчики освещенности.
Датчики положения	Эти сенсоры измеряют физическое положение устройства. Эта категория включает магнитометры и датчики ориентации устройства в пространстве.

Сенсоры могут быть реализованы аппаратно или программно. Аппаратно-реализованные датчики являются физическими элементами встроенными в мобильное устройство, они получают данные путем прямых измерений некоторых свойств, таких как ускорение, сила геомагнитного поля или изменение углов. Программно-реализованные датчики получают свои данные с одного или нескольких физических датчиков и вычисляют значение, которое от них ожидается.

Какие типы датчиков поддерживаются Android можно узнать по ссылке:

http://developer.android.com/guide/topics/sensors/sensors_overview.html.

Android предоставляет набор классов и интерфейсов для работы с сенсорами. Эти классы и интерфейсы являются частью пакета android.hardware и позволяют выполнять следующие задачи:

- определять какие сенсоры доступны на устройстве;
- определять индивидуальные возможности сенсоров, такие как

максимальное значение,

- производитель, требования к потребляемой энергии и разрешения;
- собирать данные с сенсоров и определять минимальную частоту, с которой выполняется сбор данных;
- подключать и отключать слушателей событий от датчиков, события состоят в изменении значений датчиков.

Для работы с датчиками Android предоставляет следующие классы и интерфейсы: `SensorManager` Этот класс может использоваться для создания экземпляра сервиса, связанного с сенсором. Также он предоставляет различные методы для доступа и составления списка сенсоров, подключения и отклю

	чения слушателей событий от сенсоров, сбора информации. Этот класс содержит константы, которые используются для задания точности сенсора, частоты получения данных и настройки датчиков.
<code>Sensor</code>	Этот класс используется для создания экземпляра датчика, предоставляет методы, позволяющие определить свойства сенсора.
<code>SensorEvent</code>	Система использует этот класс для создания объекта, соответствующего событию датчика и предоставляющего следующую информацию: данные сенсора; тип сенсора, который породил событие, точность данных и время появления события.
<code>SensorEventListener</code>	Данный интерфейс может использоваться для реализации двух методов, получающих уведомления (события датчиков), когда меняется значение сенсора или когда меняется точность сенсора.

Использование в приложении полученных от сенсоров данных будет рассмотрено в лабораторной работе темы 7. Подробнее об использовании

сенсоров можно узнать по ссылке:

http://developer.android.com/guide/topics/sensors/sensors_overview.html.

Задания к лабораторной работе

Задание 1. Разработка приложения, получающего координаты устройства и отслеживающего их изменение

Создадим приложение.

Далее будем править java файл, определяющий класс активности приложения. Внесем в этот класс следующие дополнения:

Нам потребуется экземпляр класса `LocationManager`, поэтому в методе `onCreate()` запишем следующую конструкцию:

```
LocationManager mlocManager =  
    (LocationManager)  
getSystemService(Context.LOCATION_SERVICE);
```

экземпляр класса `LocationManager`, как и большинство системных сервисов, создается с помощью вызова метода `getSystemService()`.

Укажем, что в приложении необходимо получать обновления координат, сделаем это с помощью вызова метода `requestLocationUpdates()` класса `LocationManager`:

```
mlocManager.requestLocationUpdates(LocationManager.GP  
S_PROVIDER, 0, 0, mlocListener);
```

первый параметр метода указывает способ получения координат, для этого используются константы класса `LocationManager`:

GPS_PROVIDER	сообщает, что координаты определяются с помощью GPS;
NETWORK_PROVIDER	сообщает, что координаты определяются с использованием сигналов сотовых вышек и WiFi.

В случае, когда необходимо получать координаты и с GPS, и от сотовых вышек необходимо вызвать метод `requestLocationUpdates()` дважды: один раз первый параметр должен быть равен `GPS_PROVIDER`, второй раз `NETWORK_PROVIDER`.

Второй и третий параметры метода управляют частотой обновлений.

Второй определяет минимальное время между извещениями об обновлениях, третий минимальное изменение расстояния между извещениями об обновлениях. Если оба эти параметра имеют нулевое значение, то извещения об обновлениях появляются настолько часто, насколько это возможно.

Последний параметр указывает на слушателя, который получает вызовы при обновлениях координат. В нашем случае слушателем является переменная `mlocListener` реализация интерфейса `LocationListener`.

- Теперь необходимо объявить переменную `mlocListener`:

```
LocationListener mlocListener = new
LocationListener() { public void
onLocationChanged(Location location) {
    tvLat.append(« «+location.getLatitude());
    tvLon.append(« «+location.getLongitude());
}
    public void onStatusChanged(String provider, int
status,
    Bundle extras) {}
    public void onProviderEnabled(String provider) {}
    public void onProviderDisabled(String provider) {}
};
```

Переменная `mlocListener` инициализируется реализацией интерфейса `LocationListener`. Этот интерфейс содержит 4 метода, каждый из которых должен быть определен в реализации интерфейса. Нас интересует метод `onLocationChanged()`, который вызывается каждый раз при изменении показаний GPS датчика, в этом методе всего лишь выполняется вывод полученных координат в информационные поля.

Чтобы разрешить получать обновления координат с помощью сигналов от сотовых вышек (`NETWORK_PROVIDER`) или с GPS датчика (`GPS_PROVIDER`), необходимо в файле манифеста добавить строку:

```
<uses-permission
android:name=«android.permission.ACCESS_COARSE_LOCATION»
```

```
    />  
    (для работы с сигналами сотовых вышек) или  
    <uses-permission  
android:name=«android.permission.ACCESS_FINE_LOCATION»  
    />
```

(для работы GPS датчиком).

Без этих полномочий приложение «сломается» во время выполнения, когда попытается запросить обновление координат. Если в приложении в качестве источника координат используются и NETWORK_PROVIDER, и GPS_PROVIDER, в манифесте достаточно запросить только полномочия на ACCESS_FINE_LOCATION. Тогда как ACCESS_COARSE_LOCATION позволяет работать только с NETWORK_PROVIDER.

В листинге 1 представлен код приложения, позволяющего получать координаты устройства и отслеживать их изменения, используя GPS приемник.

После создания приложения, разумеется, необходимо протестировать его работу. Проще всего это сделать, используя реальное устройство под управлением Android, но если

устройства под рукой нет, можно использовать виртуальное устройство. Однако при этом необходимо решить вопрос, каким образом имитировать передачу данных о местоположении на эмулятор. Проще всего воспользоваться для этого DDMS.

В Eclipse откройте компоновку (Perspective) DDMS, в этой компоновке выберите вкладку **Emulator Control**. С помощью инструмента DDMS, можно имитировать обновление данных о местоположении несколькими способами:

- вручную передать значения широты и долготы на виртуальное устройство;
- использовать GPX файл, описывающий маршрут для считывания эмулятором;
- использовать KML файл, описывающий отдельные метки местности для последовательной передачи на виртуальное устройство.

В этой части работы рассмотрели вопросы разработки приложений,

способных получать координаты устройства и отслеживать их изменения. Разработанное приложение реагирует на изменения координат устройства и выдает новые координаты в соответствующие информационные поля.

Листинг 1. Получение геолокационных данных

```
package      com.example.lab5_4_geolocation;      import
android.location.Location;

import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle; import android.app.Activity;
import      android.content.Context;      import
android.widget.TextView;

    public class MainActivity extends Activity { TextView
tvOut;

    TextView tvLon;
    TextView tvLat;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        tvOut = (TextView)findViewById(R.id.textView1); tvLon
=
        (TextView)findViewById(R.id.longitude);      tvLat      =
        (TextView)findViewById(R.id.latitude);

        //Получаем сервис
        LocationManager mlocManager = (LocationManager)
        getSystemService(Context.LOCATION_SERVICE);
        LocationListener      mlocListener      =      new
        LocationListener(){ public void onLocationChanged(Location
        location) {

            //Called when a new location is found by the network
            location      provider.      tvLat.append(«
            «+location.getLatitude());
```

```

        tvLon.append(« «+location.getLongitude());
    }

    public void onStatusChanged(String provider, int
status, Bundle extras)
    {}

    public void onProviderEnabled(String provider) {}
public void onProviderDisabled(String provider) {}

    };

    //Подписываемся на изменения в показаниях датчика
mlocManager.requestLocationUpdates(LocationManager.GPS_PR
OVIDER, 0, 0,
    mlocListener);

    //Если gps включен, то ... , иначе вывести «GPS is not
turned on...» if
(mlocManager.isProviderEnabled(LocationManager.GPS_PROVID
ER)) {
    tvOut.setText(«GPS is turned on...»);
    } else {
    tvOut.setText(«GPS is not turned on...»);
    }
    }
    }
    }

```

Содержание отчета и его форма

Отчет к лабораторной работе оформляется в виде текстового документа по форме простого реферата и должен включать:

1. Название лабораторной работы.
2. Цель лабораторной работы.
3. Краткие выводы по результатам выполнения лабораторной работы согласно пунктам раздела «Методики выполнения работы».
4. Ответы на Вопросы к лабораторной работе.

5. Формулировку задания для самостоятельной работы и результат его выполнения.

Задания для самостоятельной работы

1. Подумайте, какие особенности мобильного устройства могут повлиять на появление дополнительных возможностей в работе вашего приложения.

2. Разработать приложение, получающее координаты устройства и отслеживающее их изменения.

Вопросы к лабораторной работе

1. Характеристика систем позиционирования.
2. Категории сенсоров, поддерживаемые платформой Android.
3. Технология разработки приложения, получающего координаты устройства и отслеживающего их изменение.

Лабораторная работа 14

Использование сторонних библиотек

Цель работы: научиться писать приложения с использованием сторонних библиотек

Теоретическое обоснование

1. Библиотеки

1.1. Использование библиотек

Библиотека (от англ. library) в программировании сборник подпрограмм или объектов, используемых для разработки программного обеспечения (ПО). Для ОС Android существует большое количество подключаемых библиотек. Их можно классифицировать в зависимости от их предназначения. Выделим следующие группы:

1. Библиотеки совместимости. Они позволяют использовать возможности, появившиеся в какой-то версии ОС Android, на более ранних версиях платформы. Дело в том, что новые версии API выходят гораздо быстрее, чем в широком использовании оказываются устройства, поддерживающие эту версию. Разработчик с одной стороны должен ориентироваться на новые возможности и уметь их использовать, а с другой стараться сделать так, чтобы приложение работало на максимальном количестве устройств. Библиотеки совместимости позволяют сделать это противоречие менее жестким.

Библиотеки специального назначения. Используются для разработки игр, работы с социальными сетями, сбора статистики и в других случаях.

2. Библиотеки, предоставляющие дополнительные возможности. В эту категорию можно

отнести большое количество самых разных библиотек. Сюда можно отнести библиотеки рисования графиков, работы с изображениями, модифицированные элементы управления и многое другое.

а. Подключение библиотек

Библиотеки могут поставляться как в собранном и уже готовом к использованию виде (jar-файлы), так и в исходниках. Подключить библиотеку

(файл *.jar) очень просто. Достаточно создать папку **libs** в проекте (на том же уровне, что и папки **src** и **res**) и копировать туда файл библиотеки (можно просто перетащить). Далее необходимо добавить ее в

проект через меню **Project -> Properties**.

Если библиотека представлена в виде исходного кода, необходимо ее предварительно собрать. Необходимо щелкнуть правой кнопкой по корневой папке проекта > **Export: -> Java -> Runnable JAR file -> Указать класс для запуска -> Указать место сборки -> Finish** (рис. 14.1).

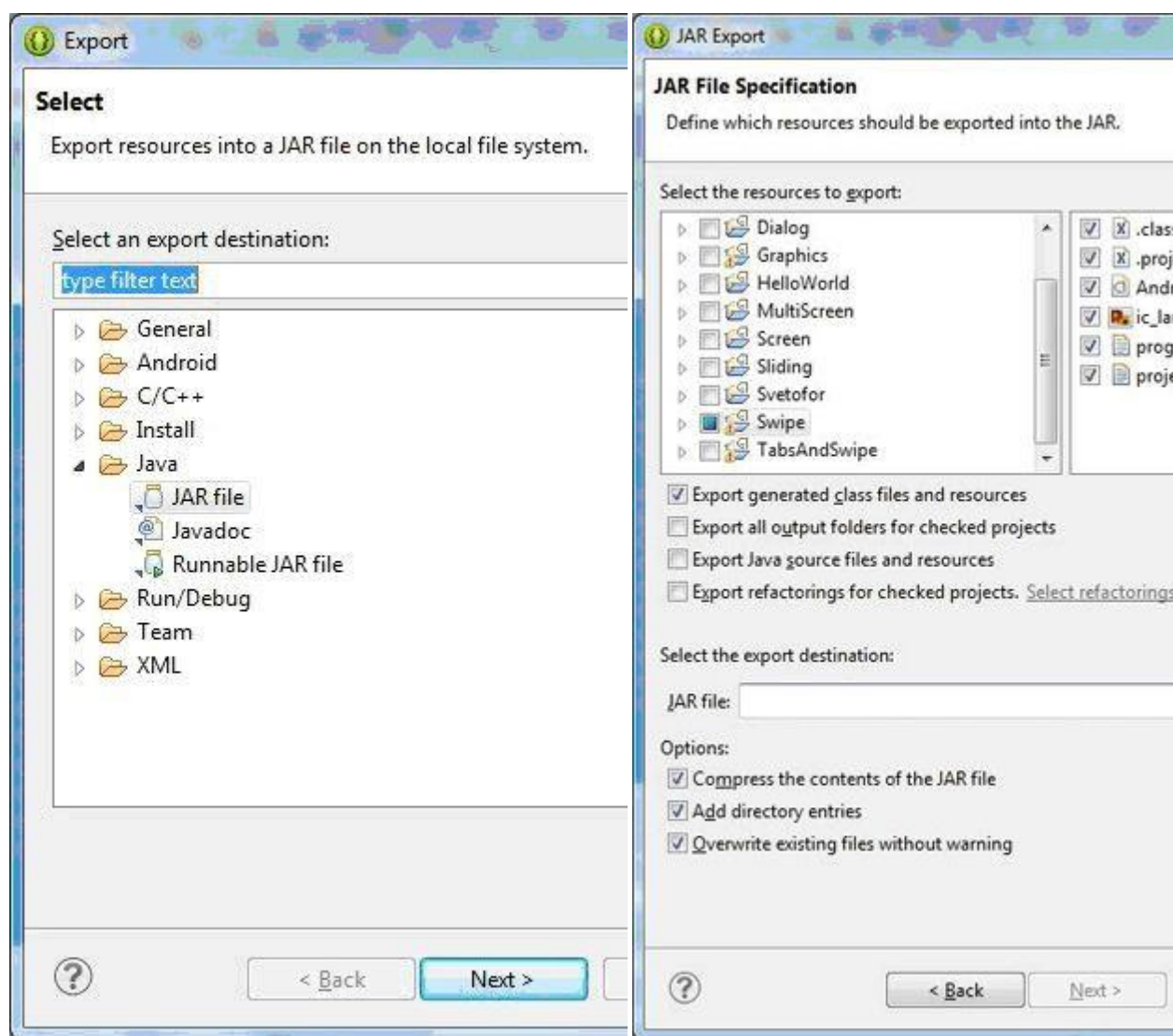


Рисунок 14.1 – Сборка библиотеки из исходного кода

3. Обзор популярных библиотек

б. Android Support Library

Android Support Library это набор библиотек, которые обеспечивают обратную совместимость новых API на более старых версиях платформы. Каждая библиотека из этого набора обладает обратной совместимостью к конкретному уровню Android API. Это означает, что ваши приложения смогут использовать возможности библиотеки и быть запущены на устройствах Android 1.6 (API level 4) и выше.

Подключение библиотек поддержки в Android является хорошим тоном в разработке приложений, зависящих от версии и возможностей платформы. Использование возможностей Support Library поможет вам распространить ваше приложение для большего числа пользователей. Если вы используете примеры Android-приложений, вы можете заметить, что все они содержат по умолчанию одну или несколько библиотек поддержки.

О возможностях различных версий Android Support Library можно узнать на официальном сайте. Скачать и установить эти библиотеки можно с помощью Android SDK Manager, выбрав в разделе Extras нужные пункты (рис. 14.2).

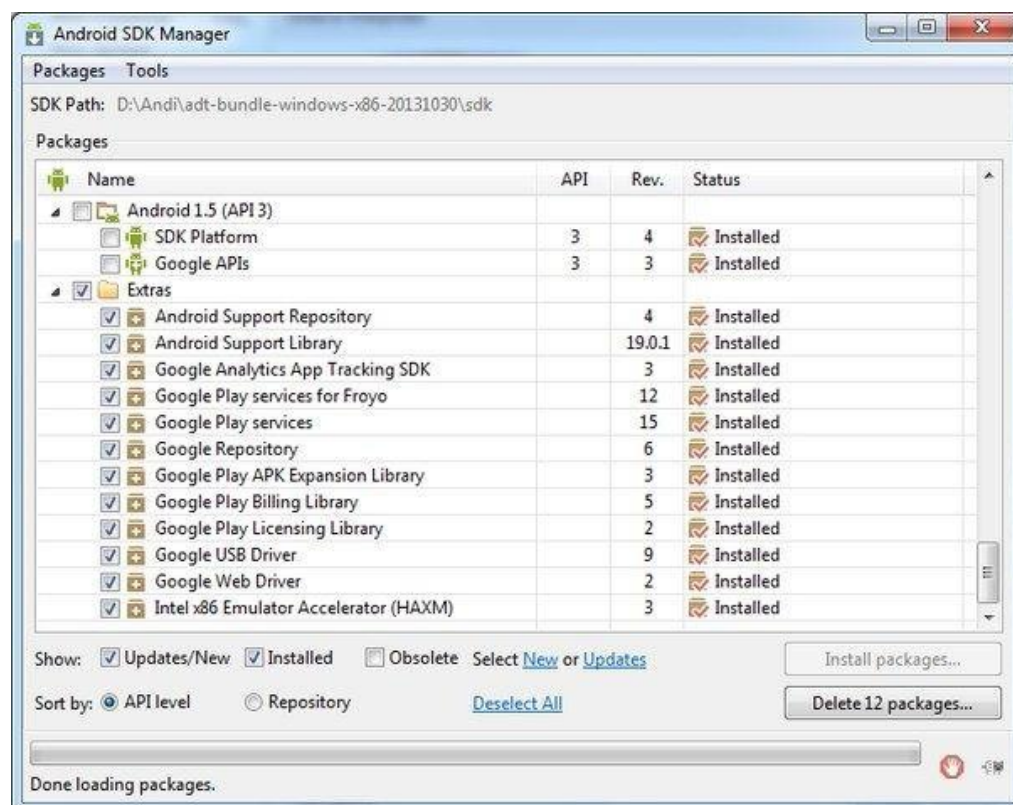


Рисунок 14.2 – Подключение Android Support Library

При настройке обратной совместимости необходимо отредактировать файл манифеста, указав в нем минимальную версию Android SDK, которая необходима для запуска приложения, и основную (целевую) версию:

```
<uses-sdk  
    android:minSdkVersion=«7»  
    android:targetSdkVersion=«17» />
```

с. Сторонние библиотеки

Помимо официальных и поддерживаемых Google библиотек совместимости, существуют

аналогичные решения от сторонних разработчиков. Вы можете их использовать на свой страх и риск, но в некоторых случаях они предпочтительнее, так как обладают некоторыми дополнительными возможностями.

NineOldAndroids один из примеров таких библиотек. Она предназначена для использования анимации, которая стала доступна только с версии Honeycomb (Android 3.0), на всех более ранних платформах. Она поддерживает различные возможности анимации и очень удобна в использовании (рис. 14.3). Главным преимуществом этой библиотеки является то, что она работает для всех версий Android, начиная с 1.0.

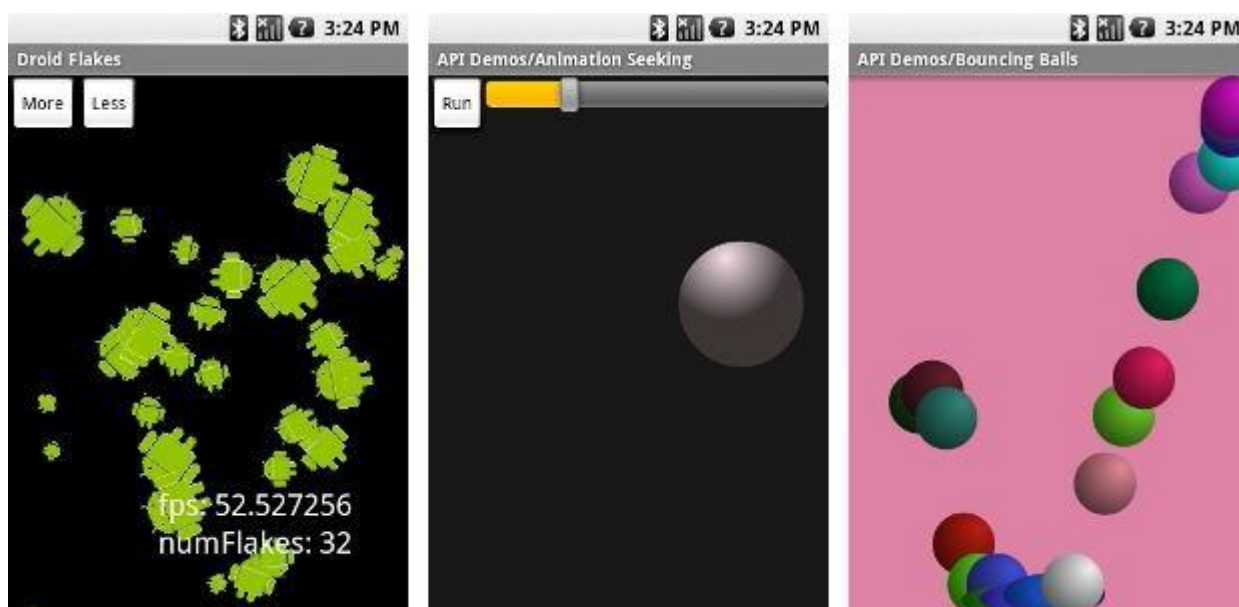


Рисунок 14.3 – Использование библиотеки NineOldAndroids

Другим примером подобного решения является автономная библиотека **ActionBarSherlock**. С ее помощью можно использовать нативный компонент ActionBar, появившийся только в версии Android 4.0, в более ранних (2.x и выше). На рисунке 14.4 представлена работа приложения, использующего библиотеку ActionBarSherlock, на устройствах со старыми версиями.

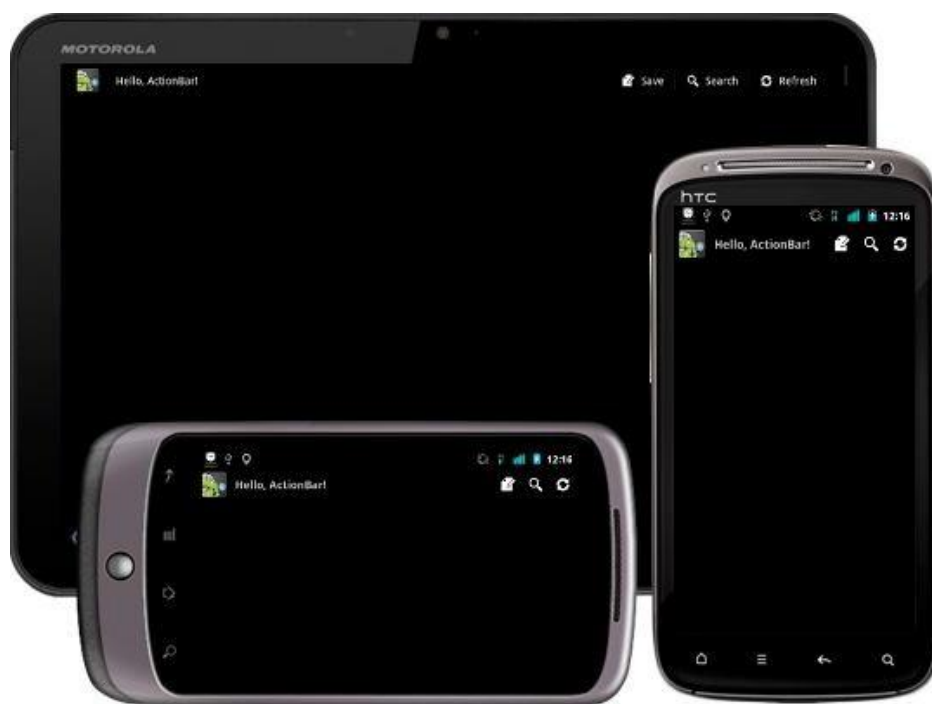


Рисунок 14.4 – Использование библиотеки ActionBarSherlock

d. Библиотеки специального назначения

Yandex.Metrica for Apps набор библиотек для сбора статистики использования мобильного приложения. Метрика показывает актуальную статистику об использовании приложения. Сервис позволяет отвечать на вопросы об аудитории и выделять любые ее сегменты. Инструменты помогают понять, как люди пользуются приложением. SDK позволяет отслеживать следующие данные:

- 1) информация об устройстве;
- 2) информация о сессиях;
- 3) информация об источнике перехода пользователя на страницу скачивания приложения;

- 4) действия, выполненные пользователем в приложении;
- 5) местоположение пользователя;
- 6) ошибки, возникающие во время использования приложения;
- 7) собственные события;
- 8) другие данные (например, количество пользователей, установивших приложение).

Facebook SDK for Android официальная библиотека Facebook для Android. Позволяет писать сообщения на стену, читать и менять статусы, смотреть ленту друзей и многое другое (рис. 14.5).

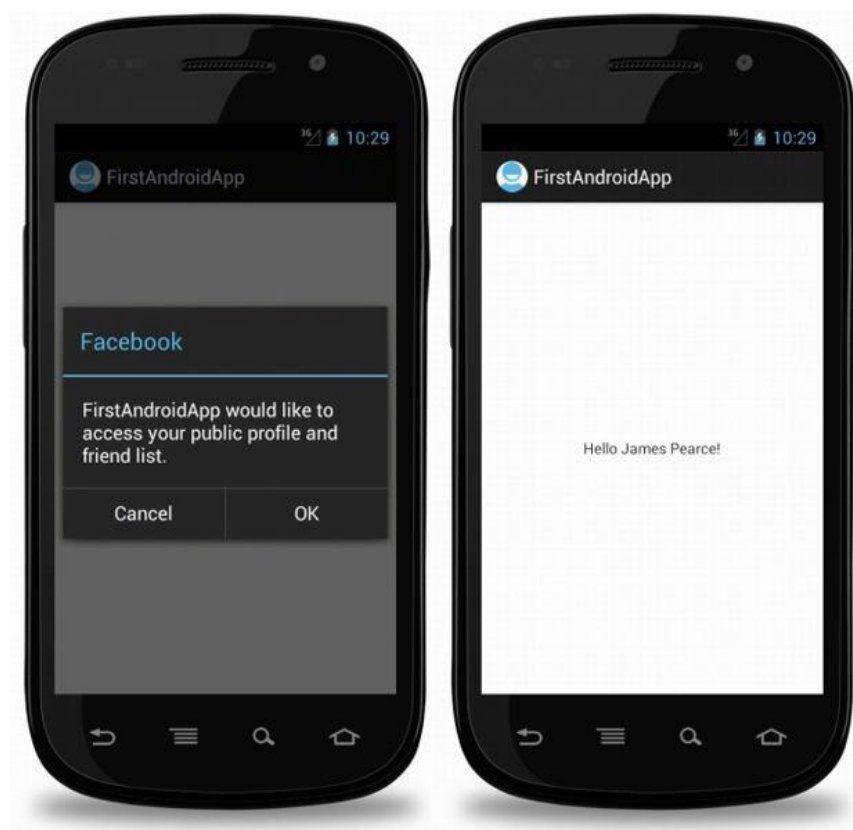


Рисунок 14.5 – Пример приложения, подготовленного с использованием Facebook SDK for Android

а. Прикладные библиотеки

К этой категории можно отнести различные библиотеки, предоставляющие дополнительные возможности.

Universal Image Loader for Android мощная и гибкая библиотека,

предназначенная для загрузки, кеширования и отображения картинок в Android (рис. 14.6).

Возможности:

- i. Многопоточная загрузка изображений.
- ii. Широкие возможности настройки и конфигурирования.
- iii. Кеширование загруженных изображений как в оперативной памяти, так и на карте.
- iv. Поддержка виджетов.
- v. Поддерживает Android 2.0 и выше.

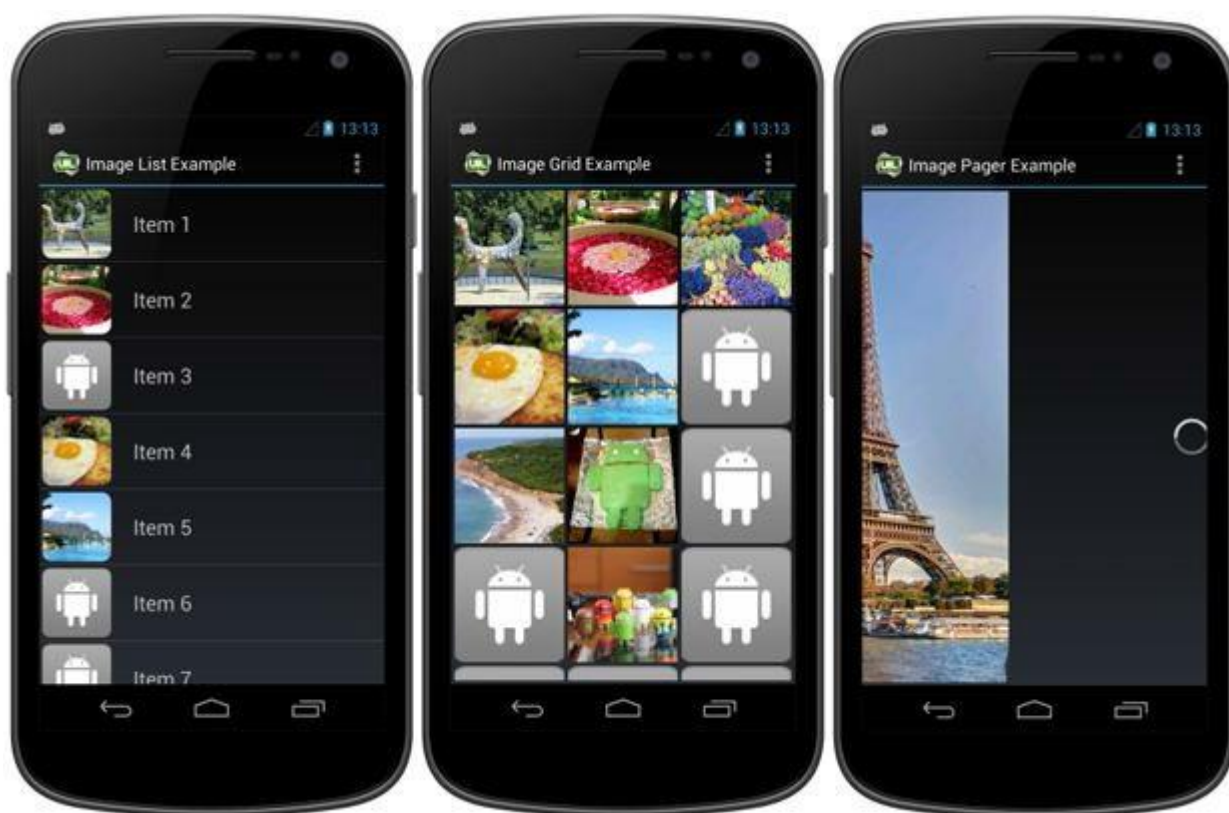


Рисунок 14.6 – Пример работы с библиотекой Universal Image Loader for Android

jsoup: Java HTML Parser предназначена для парсинга HTML-страниц. Предоставляет очень удобный API для извлечения данных и манипуляции с ними, используя DOM, CSS и методы в стиле jQuery. Поддерживает спецификации HTML5 и позволяет парсить страницы так же, как это делают современные браузеры.

Возможности:

- 1) Может принимать в качестве параметра URL, файл или строку.
- 2) Находит и извлекает данные, используя DOM и селекторы CSS.
- 3) Позволяет манипулировать HTML-элементами, атрибутами и текстом.
- 4) Выводит чистый HTML.

Android Holo ColorPicker удобная библиотека, позволяющая выбирать цвет с использованием цветового колеса, выполненная в официально рекомендованном стиле Holo. Сайт библиотеки содержит описание работы с ней и необходимые ссылки (рис. 14.7).



Рисунок 14.7 – Пример использования Android Holo ColorPicker

Библиотека **MapNavigator** предназначена для работы с картами Google Maps. Позволяет определять направления и отображать маршрут на карте (рис. 14.8). Работает только с Google Maps v2.

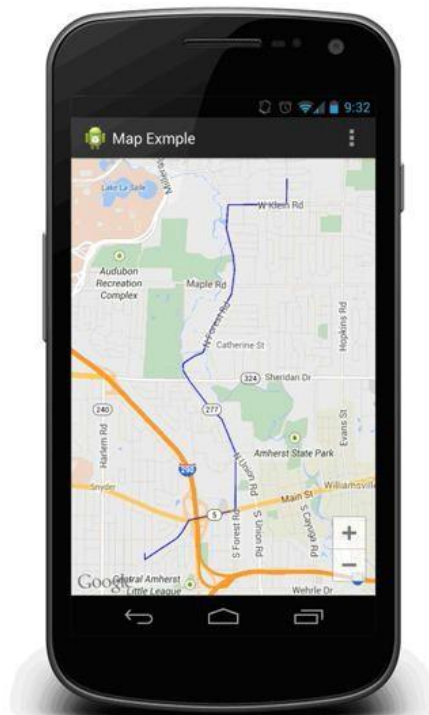


Рисунок 14.8 – Пример использования библиотеки MapNavigator

AChartEngine библиотека, предназначенная для построения графиков. Позволяет строить графики различных типов:

- 1) Линии графиков функций.
- 2) Поточечные графики.
- 3) Гистограммы.
- 4) Круговые диаграммы.
- 5) Пузырьковые диаграммы.
- 6) Комбинированные диаграммы.
- 7) Другие виды диаграмм и графиков.

Все типы диаграмм поддерживают несколько рядов данных (рис. 14.9).



Рисунок 14.9 – Пример использования библиотеки AChartEngine

Разумеется, мы рассмотрели лишь малую долю существующих библиотек. Обзор не претендует на полноту, но это не так важно, так как его целью было представить многообразие возможностей, открывающихся перед разработчиком. Кроме того, большое количество разнообразных библиотек описаны на сайте <http://www.androidviews.net/>.

8) Безопасность использования подключаемых библиотек

Подключаемые библиотеки являются очень удобным инструментом, облегчающим труд программиста. Однако разработчики приложений, использующие сторонние библиотеки подобного рода, часто не подозревают об их проблемах с безопасностью. Библиотека может содержать возможности, которые могут использоваться злоумышленниками в преступных целях.

Например, в октябре 2013 года была опубликована статья с результатами исследования, согласно которому популярная у разработчиков библиотека, предоставляющая возможность отображения рекламы в приложениях, может

использоваться для сбора информации и запуска вредоносного кода. Исследователи не раскрыли истинного названия библиотеки, зато описали возможный вред. Например, она может запускать на устройстве произвольный код, извлекать текстовые сообщения, список контактов и вызовов, передавать секретную информацию пользователя в виде простого текста по протоколу HTTP, использовать камеру без ведома пользователя, запускать вредоносные java-скрипты. Злоумышленник может превратить эту библиотеку в ботнет, перехватывая ее трафик и отправляя вредоносные команды и код.

При выборе библиотеки следует соблюдать осторожность. Если библиотека поставляется в виде исходников и о ней мало информации, не лишним будет просмотреть ее код в поисках странных функций, которые не нужны для ее функционирования. Но для уже собранной библиотеки анализ исходников может стать непростой задачей. Поэтому при выборе библиотеки следует соблюдать ряд правил:

9) Не использовать скомпрометированные библиотеки. Если о какой-то библиотеке появляются сведения, что она может содержать вредоносный код, следует отказаться от ее использования в новых проектах и по возможности пересмотреть ее применение в уже существующих.

10) С осторожностью использовать библиотеки из сомнительных источников.

11) Обязательно ознакомиться с форумами и сайтами, где могут обсуждаться библиотеки.

Кроме того, это может помочь вам подобрать наиболее подходящее решение для вашей конкретной задачи.

12) По возможности просмотреть исходники.

Применять другие правила информационной безопасности, которые могут иметь значение в каждом конкретном случае.

Задания к лабораторной работе

Задание 1. Использование библиотеки для построения графиков AChartEngine Рассмотрим пример подключения библиотеки AChartEngine, предназначенной для построения графиков. На сайте разработчика

(<http://code.google.com/p/achartengine>) помимо самой библиотеки можно найти подробную документацию, оформленную в стиле Javadoc

pages, примеры использования библиотеки, а также ее исходный код.

В разделе **Downloads** также можно скачать демонстрационные примеры использования библиотеки.

1. Создайте проект **Graphics**.

2. Чтобы подключить библиотеку, нужно скачать ее с сайта разработчика, потом просто перетащить из проводника в папку **libs** вашего проекта в Eclipse.

3. теперь нужно добавить строку `<activity android:name=«org.achartengine.GraphicalActivity»/>` в раздел `<application>` в файле манифеста вашего проекта. Библиотека подключена.

4. Теперь перейдем к файлу **MainActivity.java**. Создайте и инициализируйте три массива (в методе `onCreate()`). В первом будут содержаться цвета, во втором подписи, в третьем значения.

```
int[] values = new int[] { 25,25,25,25 };  
String[] bars = new String[] {«Bananas», «Kiwi»,  
«Oranges», «Cream»}; int[] colors = new int[] {  
Color.YELLOW,Color.GREEN, Color.RED, Color.WHITE };
```

5. Создайте объект `CategorySeries` и загрузите в него массивы строк с подписями и значения. При построении диаграмм будут использоваться именно эти данные.

```
CategorySeries series = new CategorySeries(«Pie  
Chart»);
```

6. Создайте объект `DefaultRenderer`. `CategorySeries` и `DefaultRenderer` определены в библиотеке `AChartEngine`.

```
DefaultRenderer dr = new DefaultRenderer;
```

7. Для каждого цвета в диаграмме создайте `SimpleSeriesRenderer` и установите его в этот цвет. Затем каждый `SimpleSeriesRenderer` добавьте в `DefaultRenderer`. Таким образом, имеем один `Render` на каждый `Series`. К слову, `Render` может включать другой `Render`, таким образом можно создавать подкатегории.

```

        for (int v=0; v<4; v++){ series.add(bars[v],
values[v]);
        SimpleSeriesRenderer r = new SimpleSeriesRenderer();
r.setColor(colors[v]);
        dr.addSeriesRenderer(r);
    }

```

8. Создайте объект `PieChartIntent` и передайте ему `CategorySeries` и `DefaultRenderer`.

```

        return ChartFactory.getPieChartIntent(this, series,
dr, «Fruit Salad»);

```

С помощью `PieChartIntent` можно показать диаграмму, как отдельную деятельность.

9. С помощью методов библиотеки можно изменить размер и цвет подписей, включить зум:

```

        dr.setZoomEnabled(true);
dr.setChartTitleTextSize(20);
dr.setLegendTextSize(TEXT_SIZE);
dr.setChartTitleTextSize(20);
dr.setZoomButtonsVisible(false);
dr.setLabelsTextSize(TEXT_SIZE);
dr.setLegendTextSize(TEXT_SIZE);
dr.setLabelsColor(Color.BLACK);

```

10. Для удобства можно создать дополнительную переменную, в которой будет храниться размер текста.

```

        private static final int TEXT_SIZE = 40; Intent intent
= buildIntent();
        startActivity(intent);
    }
    public Intent buildIntent() {
        int[] values = new int[] { 25,25,25,25 };
        . . .
    }

```

```

dr.addSeriesRenderer(r);
}
dr.setChartTitleTextSize(20);
. . .
dr.setLabelsColor(Color.BLACK);
return
ChartFactory.getPieChartIntent( this, series, dr, «Fruit
Salad»);
} (рис. 14.10, 14.11)

```

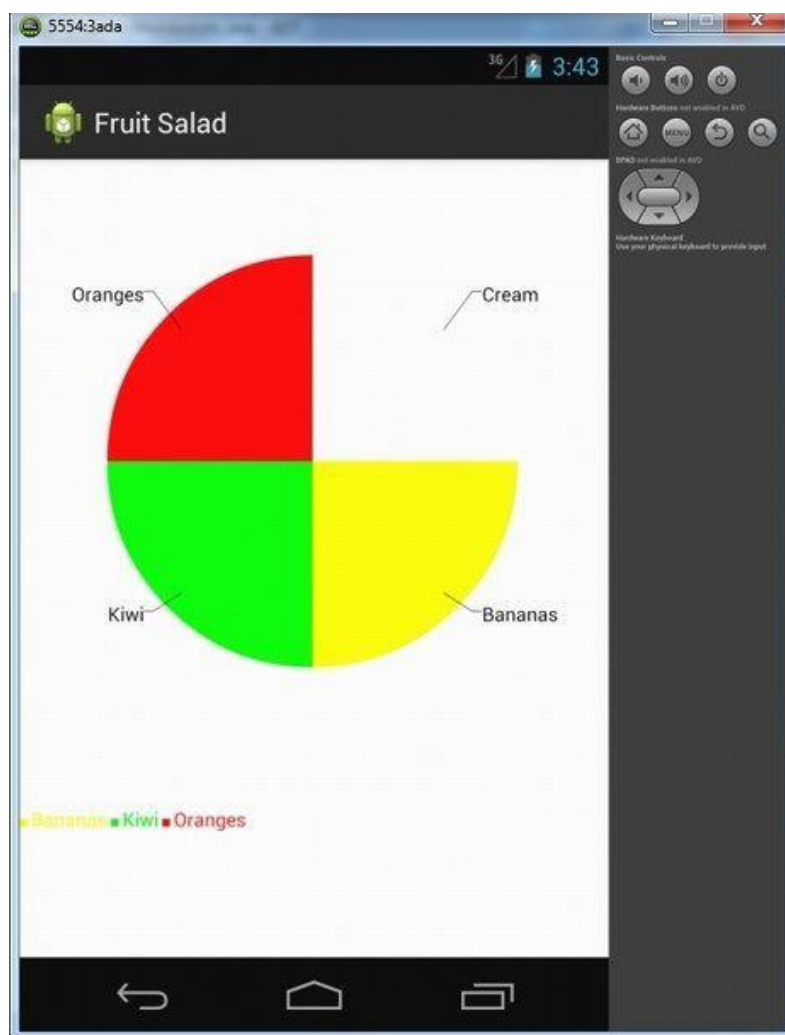


Рисунок 14.10 – Приложение, запущенное на эмуляторе

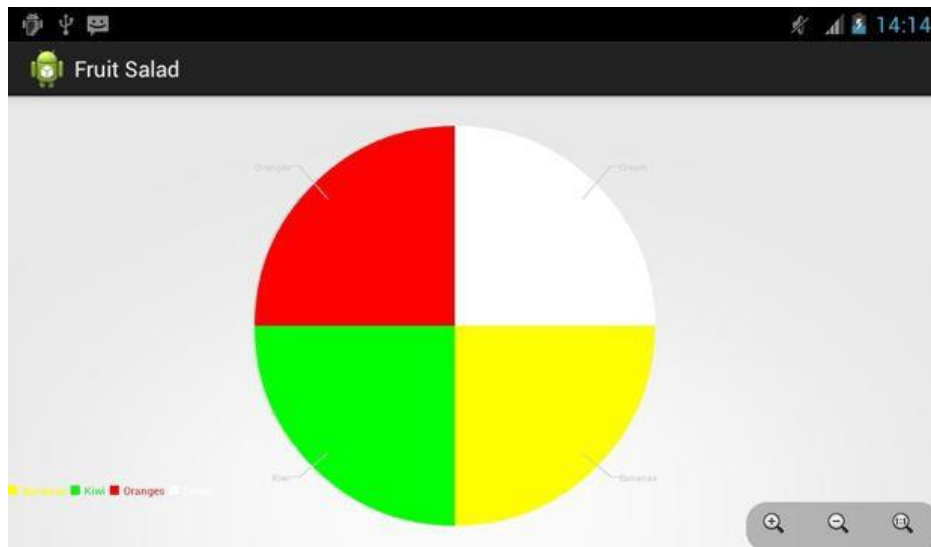


Рисунок 14.11 – Приложение, запущенное на устройстве

Листинг 1. Код MainActivity.java

```
package com.mypackage.graphics;

import org.achartengine.ChartFactory;      import
org.achartengine.chart.PieChart;
import org.achartengine.model.CategorySeries;
import org.achartengine.renderer.DefaultRenderer;
import
org.achartengine.renderer.SimpleSeriesRenderer;
import android.content.Context;
import android.content.Intent;      import
android.graphics.Color;      import android.app.Activity;
import android.os.Bundle;

public class MainActivity extends Activity { private
static final int TEXT_SIZE = 40;

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    Intent intent = buildIntent(); startActivity(intent);
}
```

```

    public Intent buildIntent() {
        int[] values = new int[] { 25,25,25,25 };
        String[] bars = new String[] {«Bananas»,    «Kiwi»,
«Oranges»,
    «Cream»};
        int[] colors = new int[] {    Color.YELLOW,
                                        Color.GREEN,
Color.RED,
        Color.WHITE };

        CategorySeries series = new CategorySeries(«Pie
Chart»); DefaultRenderer dr = new DefaultRenderer();
        for (int v=0; v<4; v++){ series.add(bars[v],
values[v]);
        SimpleSeriesRenderer r = new SimpleSeriesRenderer();
        r.setColor(colors[v]);
        dr.addSeriesRenderer(r);
    }
    dr.setZoomButtonsVisible(true);
    ad»);
    }

    }
    dr.setZoomEnabled(true);
dr.setChartTitleTextSize(20);
dr.setLegendTextSize(TEXT_SIZE);
dr.setChartTitleTextSize(20);
dr.setZoomButtonsVisible(false);
dr.setLabelsTextSize(TEXT_SIZE);
dr.setLegendTextSize(TEXT_SIZE);
dr.setLabelsColor(Color.BLACK);

```



```
return ChartFactory.getPieChartIntent(this, series,
dr, «Fruit Sal
```

Содержание отчета и его форма

Отчет к лабораторной работе оформляется в виде текстового документа по форме простого реферата и должен включать:

1. Название лабораторной работы.
2. Цель лабораторной работы.
3. Краткие выводы по результатам выполнения лабораторной работы согласно пунктам раздела «Методики выполнения работы».
4. Ответы на Вопросы к лабораторной работе.
5. Формулировку задания для самостоятельной работы и результат его выполнения.

Задания для самостоятельной работы

1. Подумайте, какие особенности мобильного устройства могут повлиять на появление дополнительных возможностей в работе вашего приложения.

2. Научиться подключать библиотеки.
3. Научиться использовать библиотеки в своих приложениях.

Вопросы к лабораторной работе

1. Технология подключения библиотек.
2. Характеристика Android Support Library.
3. Характеристика сторонних библиотек.
4. Характеристика библиотек специального назначения.
5. Характеристика прикладных библиотек.
6. Технология использования библиотеки для построения графиков AChartEngine.

Лабораторная работа 15

Работа с базами данных в Android

Цель работы: разработка Android приложения, демонстрирующего возможности работы с базой данных SQLite.

Теоретическое обоснование

Для достижения цели, поставленной в лабораторной работе, сформулируем требования к разрабатываемому приложению. Приложение демонстрирует возможности работы с базой данных, предполагает реализацию следующих действий:

- 1) добавление записей в базу данных;
- 2) считывание строк и вывод на экран;
- 3) удаление базы данных.

Задания к лабораторной работе

Задание 1. Создание приложения

Создадим новое Android приложение:

Project Name: Lab7_1-SQLite;

Package Name: com.example.lab7_1_sqlite;

Activity Files: SQLiteActivity.java, activity_sqlite.xml.

Задание 2. Настройка интерфейса

Разрабатываемое приложение является учебно-тренировочным, предполагает демонстрацию возможностей работы с базой данных: создание, добавление записей, просмотр записей, удаление базы данных. Поэтому интерфейс будет максимально простым: нам понадобятся два поля для ввода данных (в таблице базы данных два столбца), поле для вывода записей базы данных и три кнопки, по одной на каждое действие: добавление записей, вывод записей, удаление базы (рис. 15.1).



Рисунок 15.1 – Внешний вид приложения для демонстрации основных возможностей базы данных SQLite в системе Android

В нашем случае активность приложения имеет вид, показанный на рис. 1. Разумеется, не обязательно в точности воспроизводить предложенный внешний вид приложения, а на самом деле даже имеет смысл настроить интерфейс по своему усмотрению, чтобы потренировать навыки работы с компонентами и настройками пользовательского интерфейса.

Задание 3. Реализация логики

Для создания и обновления базы данных, Android предоставляет класс SQLiteOpenHelper.

Для работы с этим классом в приложении необходимо создать класс-наследник, в котором обязательно реализовать методы: onCreate() и onUpgrade().

Создадим новый класс, назовем его, например, MyOpenHelper:

```
public class MyOpenHelper extends
SQLiteOpenHelper{...}
```

Реализуем метод onCreate():

```

public void onCreate(SQLiteDatabase DB) {
    String query=«create table « + TABLE_NAME + « (_id
integer primary key autoincrement, « + field_name_1 + «
TEXT, « + field_name_2 + « TEXT)»; DB.execSQL(query);
}

```

Параметром метода onCreate() является объект класса SQLiteDatabase, позволяющего работать с базой данных напрямую.

В строке query формируется запрос на создание таблицы, где

TABLE_NAME	константа, содержащая имя таблицы,
FIELD_NAME_1	константа, содержащая имя первого столбца,
FIELD_NAME_2	константа, содержащая имя второго столбца.

Все эти константы объявлены в классе MyOpenHelper:

```

public String TABLE_NAME=«first_table»; public String
FIELD_NAME_1=«first_field»; public String
FIELD_NAME_2=«second_field»;

```

Метод execSQL() класса SQLiteDatabase запускает запрос на выполнение.

Еще необходимо реализовать метод onUpgrade(), в нашем случае он ничего существенного не делает, но необходимо прописать реализацию этого метода даже, если это будет пустая реализация:

```

public void onUpgrade(SQLiteDatabase DB, int
oldVersion, int newVersion) {
    Log.d(«myLogs»,»| Upgrade |»+DB.toString());
}

```

Система потребует создать конструктор класса MyOpenHelper, т. к. для его предка конструктор без параметров не определен, поэтому добавим в код класса следующие строчки: MyOpenHelper(Context ct, String nm, SQLiteDatabase.CursorFactory cf, int vs){
super(ct, nm, cf, vs);

```
}
```

Полученный конструктор не делает ничего особенного, просто вызывает конструктор предка, т. е. класса `SQLiteOpenHelper`.

Мы описали создание вспомогательного класса, необходимого нам для создания и открытия базы данных, полный код этого класса представлен в листинге 25.1.

Пришло время описать реализацию заявленных функций приложения. Далее будем работать с классом активности, описание которого находится в файле **`SQLiteActivity.java`**.

Выполним необходимую подготовку, зададим переменные, как поля класса активности:

```
EditText field1, field2, result; MyOpenHelper myHelper  
= null; SQLiteDatabase DB;
```

Вторая строка определяет объект класса `MyOpenHelper`, а третья объект класса `SQLiteDatabase`, далее в программе все взаимодействия с базой данных после ее создания будут выполняться через этот объект.

Далее в методе `onCreate()` активности создадим экземпляр класса `MyOpenHelper`, для создания, открытия и возможно управления базой данных, имя которой `myDB`:

```
myHelper=new MyOpenHelper(this, «myDB», null, 1);
```

Для работы приложения необходимо задать действия, которые будут выполняться при нажатии на кнопки, для этого настроим свойство `On Click` кнопок:

1. для кнопки **Ввод данных** свойству `On Click` присвоим значение `insertIntoDatabase`, при нажатии на кнопку будет вызываться метод с этим именем, реализованный в классе активности;

2. для кнопки **Вывод данных** свойству `On Click` присвоим значение `readDatabase`, при нажатии на кнопку будет вызываться метод с этим именем, реализованный в классе активности;

3. для кнопки **Delete** свойству `On Click` присвоим значение `deleteDatabase`, при нажатии на кнопку будет вызываться метод с этим именем,

реализованный в классе активности.

Рассмотрим реализацию метода `insertIntoDatabase()`.

В этом методе после проверки того, что в поля введены значения, получаем экземпляр базы данных с разрешением на запись:

```
DB = myHelper.getWritableDatabase();
```

Далее формируем запись, добавляемую в таблицу:

```
ContentValues CV = new ContentValues();
CV.put(myHelper.FIELD_NAME_1, field1.getText().toString());
;
CV.put(myHelper.FIELD_NAME_2, field2.getText().toString());
;
```

После этого добавляем запись в таблицу и закрываем экземпляр базы данных:

```
DB.insert(myHelper.TABLE_NAME, null, CV); DB.close();
```

Полный код рассмотренного метода можно найти в листинге 25.2. Рассмотрим реализацию метода `readDatabase()`.

В этом методе сначала получаем экземпляр базы данных с разрешением на чтение:

```
DB = myHelper.getReadableDatabase();
```

Далее формируем и выполняем запрос к базе данных на получение всех значений из базы, результат запроса помещается в объект класса `Cursor`:

```
String columns[]={«_id»,myHelper.FIELD_NAME_1,
myHelper.FIELD_NAME_2};
Cursor cursor=DB.query(myHelper.TABLE_NAME, columns,
null,
null, null, null, «_id»);
```

После этого выведем содержимое класса `Cursor` в поле на форме, предназначенное для вывода информации:

```
if(cursor!=null){ cursor.moveToFirst();
if (cursor.moveToFirst()) {
do {
```

```

        result.setText(result.getText().toString()+»\n»+cursor
r.getString(0)+»)
        «+cursor.getString(1)+»,
        «+cursor.getString(2));
    } while (cursor.moveToNext());
    }
}

```

В этом кусочке кода основное внимание можно обратить на работу с классом `курсор`, очень похоже на работу со списками. Начиная с начала списка получаем значения элементов и передвигаемся на следующий элемент, пока не достигнем конца.

После всех манипуляций с базой данных необходимо ее закрыть:

```
DB.close();
```

Полный код рассмотренного метода можно найти в листинге 25.2. Рассмотрим реализацию метода `deleteDatabase`.

В этом методе сначала получаем экземпляр базы данных с разрешением на запись:

```
DB = myHelper.getWritableDatabase();
```

Удалим таблицу, с помощью метода `delete()` класса `SQLiteDatabase`:

```
DB.delete(myHelper.TABLE_NAME, null, null);
```

После всего закроем базу:

```
DB.close();
```

Полный код рассмотренного метода можно найти в листинге 25.2.

В работе на примере простого приложения рассмотрели выполнение основных операций с базами данных: создание, добавление записей, просмотр всех записей таблицы, удаление. Для выполнения выборок из таблицы, необходимо познакомиться с основами построения запросов в `SQLite` и сформировать запрос для метода `query()` класса `SQLiteDatabase`.

Листинг 1. Класс `MyOpenHelper` package

```

com.example.lab7_1_sqlite; import
android.content.Context;

```

```

import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.util.Log;

public class MyOpenHelper extends SQLiteOpenHelper {
public String TABLE_NAME=«first_table»;

    public String FIELD_NAME_1=«first_field»;
    public String FIELD_NAME_2=«second_field»;

    MyOpenHelper(Context ct, String nm,
SQLiteDatabase.CursorFactory cf, int vs){
        super(ct, nm, cf, vs);
    }

    @Override
    public void onUpgrade(SQLiteDatabase DB, int
oldVersion, int newVersion) { Log.d(«myLogs»,»| Upgrade
|»+DB.toString());
    }

    @Override
    public void onCreate(SQLiteDatabase DB) {
Log.d(«myLogs»,»| Create |»+DB.toString());

        String query=«create table « + TABLE_NAME + « ( _id
integer primary key autoincrement, « + FIELD_NAME_1 + «
TEXT, « + FIELD_NAME_2 + « TEXT)»;
        DB.execSQL(query);
    }
}

```

Листинг 2. Класс SQLiteActivity

```

package com.example.lab7_1_sqlite; import
android.os.Bundle;

import android.app.Activity;

import android.content.ContentValues; import
android.database.Cursor;

```



```

import android.database.sqlite.SQLiteDatabase; import
android.util.Log;

import android.view.View;
import android.widget.EditText;

public class SQLiteActivity extends Activity {
MyOpenHelper myHelper = null;

    EditText field1, field2, result; SQLiteDatabase DB;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_sqlite);
myHelper=new MyOpenHelper(this, «myDB», null, 1);
        field1=(EditText)findViewById(R.id.field1);
        field2=(EditText)findViewById(R.id.field2);
        result=(EditText)findViewById(R.id.dbResult);
    }

    public void insertIntoDatabase(View v){
        if(!field1.getText().toString().equals(«») &&
            !field2.getText().toString().equals(«»)){
            Log.d(«myLogs»,»Insert INTO DB
(«+field1.getText().toString()+ «,» +
            field2.getText().toString()+»)»);
            DB = myHelper.getWritableDatabase();
            String query=«create table if not exist « +
myHelper.TABLE_NAME +
            « (_id integer primary key autoincrement, « +
myHelper.FIELD_NAME_1 + «
            TEXT, « + myHelper.FIELD_NAME_2 + « TEXT)»»;
            ContentValues CV = new ContentValues();
            CV.put(myHelper.FIELD_NAME_1,field1.getText().toString())
;

```

```

CV.put(myHelper.FIELD_NAME_2,field2.getText().toString())
; DB.insert(myHelper.TABLE_NAME,null,CV);
    DB.close();
    field1.setText(««);
    field2.setText(««);
}
}
public void readDatabase(View v){ result.setText(««);
Log.d(«myLogs»,»READ FROM DB»);
DB = myHelper.getReadableDatabase();
String columns[]={«_id»,myHelper.FIELD_NAME_1,
myHelper.FIELD_NAME_2}; Cursor
cursor=DB.query(myHelper.TABLE_NAME, columns, null, null,
null, null, «_id»);
    if(cursor!=null){
        cursor.moveToFirst();
        if (cursor.moveToFirst()) {
            do {
                result.setText(result.getText().toString()+ «\n» +
                cursor.getString(0) + «) « + cursor.getString(1) +
«,)» + cursor.getString(2));
            } while (cursor.moveToNext());
        }
    }
    DB.close();
}
public void deleteDatabase(View v){
Log.d(«myLogs»,»Delete Database»);

    DB = myHelper.getWritableDatabase();
    DB.delete(myHelper.TABLE_NAME, null, null);

```

```
DB.close();  
  
}
```

Содержание отчета и его форма

Отчет к лабораторной работе оформляется в виде текстового документа по форме простого реферата и должен включать:

1. Название лабораторной работы.
2. Цель лабораторной работы.
3. Краткие выводы по результатам выполнения лабораторной работы согласно пунктам раздела «Методики выполнения работы».
4. Ответы на вопросы к лабораторной работе.
5. Формулировку задания для самостоятельной работы и результат его выполнения.

Задания для самостоятельной работы

1. Создать приложение.
2. Настроить интерфейс приложения.
3. Реализовать логику приложения.

Вопросы к лабораторной работе

1. Характеристика класса SQLiteOpenHelper.
2. Характеристика реализации методов: onCreate() и onUpgrade().
3. Характеристика реализации метода insertIntoDatabase().

Лабораторная работа 16

Установка и настройка среды программирования Intel XDK.

Создание приложений

Цель работы: установка и настройка среды Intel XDK и создание простейших приложений на основе пустого шаблона и встроенных примеров.

Теоретическое обоснование

Лабораторная работа посвящена различным аспектам разработки мобильных приложений для ОС Android с использованием среды Intel XDK. В ней содержатся указания по установке и настройке среды, созданию приложений как «с нуля», так и из готовых шаблонов, запуску приложений в эмуляторе и на мобильных устройствах. Имеются задания для самостоятельной работы.

Задания к лабораторной работе

Задание 1. Установка и настройка среды

Скачать среду можно на сайте разработчика <http://xdk-software.intel.com/> (рис. 16.1).



Рисунок 16.1 – Сайт разработчика

Чтобы приступить к загрузке, достаточно выбрать «**Download Intel XDK NEW**». Вы будете переведены на страницу скачивания дистрибутива для Вашей операционной системы, например, Intel® XDK download for Microsoft Windows, где через 5 секунд начнется загрузка. Установка среды разработки не

представляет собой ничего сложного типичный набор действий для установки приложения (рис. 16.2). После завершения установки инсталлятор предложит вам две гиперссылки на сайты, где можно найти документацию по среде и задать свои вопросы по работе с Intel XDK (рис. 16.2, 16.3).



Рисунок 16.2 – Установка среды



Рисунок 16.3 – Поддержка пользователей

Итак, среда Intel XDK установлена, ярлык на нее добавлен в главное меню операционной системы (рис. 16.4).

При запуске приложение предложит авторизоваться (рис. 16.5). Если учетной записи

Intel XDK еще нет, среда предложит пройти стандартную процедуру регистрации (рис. 16.6). При следующем запуске окна авторизации достаточно будет выбрать пункт **«Keep me logged in on this computer»** для автоматической авторизации на данном компьютере (рис. 16.4).

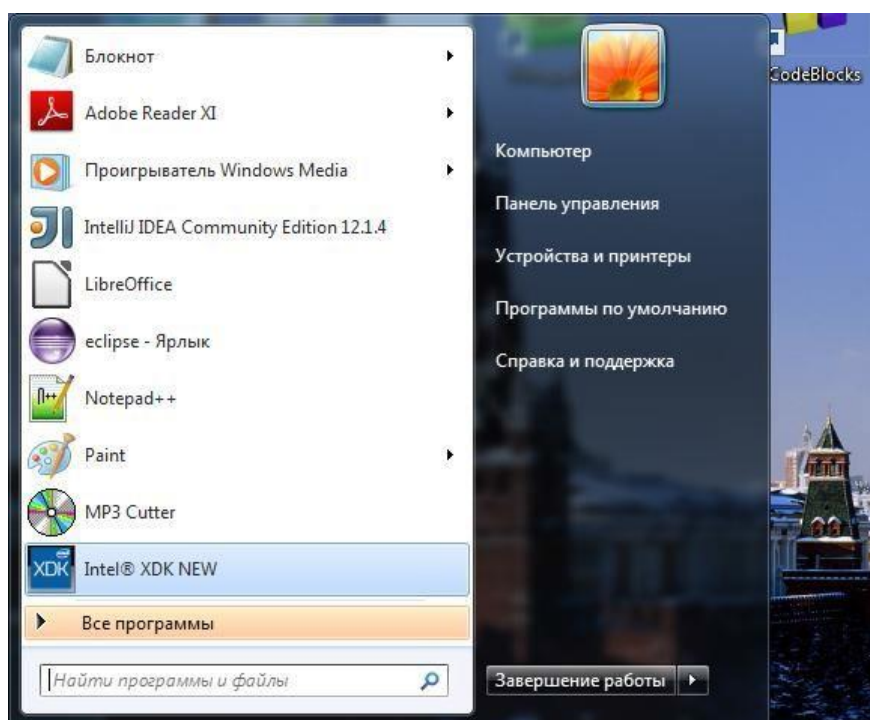


Рисунок 16.4 – Intel XDK в меню пуск WIN7

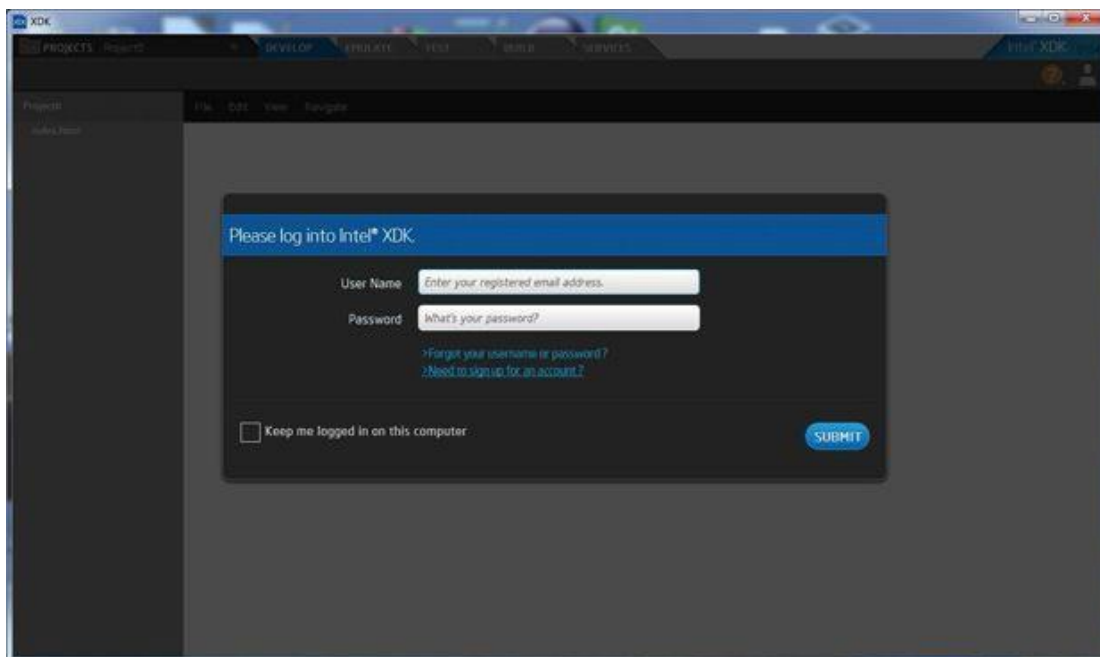


Рисунок 16.5 – Авторизация

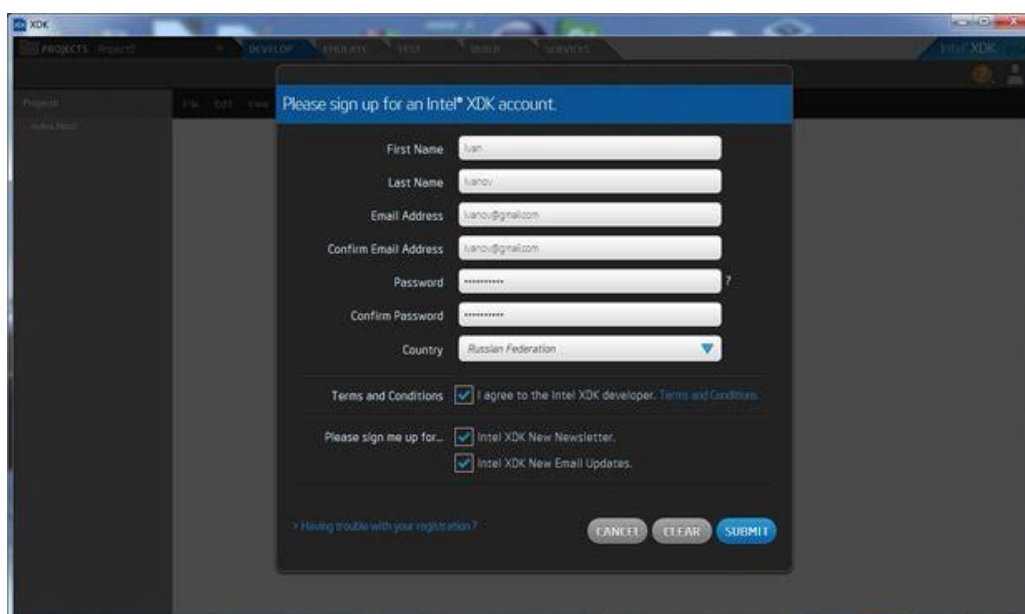


Рисунок 16.6 – Регистрация

После авторизации мы увидим основное окно среды, проекты, над которыми уже велась работа, с указанием даты создания и модификации, а также места их расположения, и пункты меню для работы в панели управления (рис. 16.7).

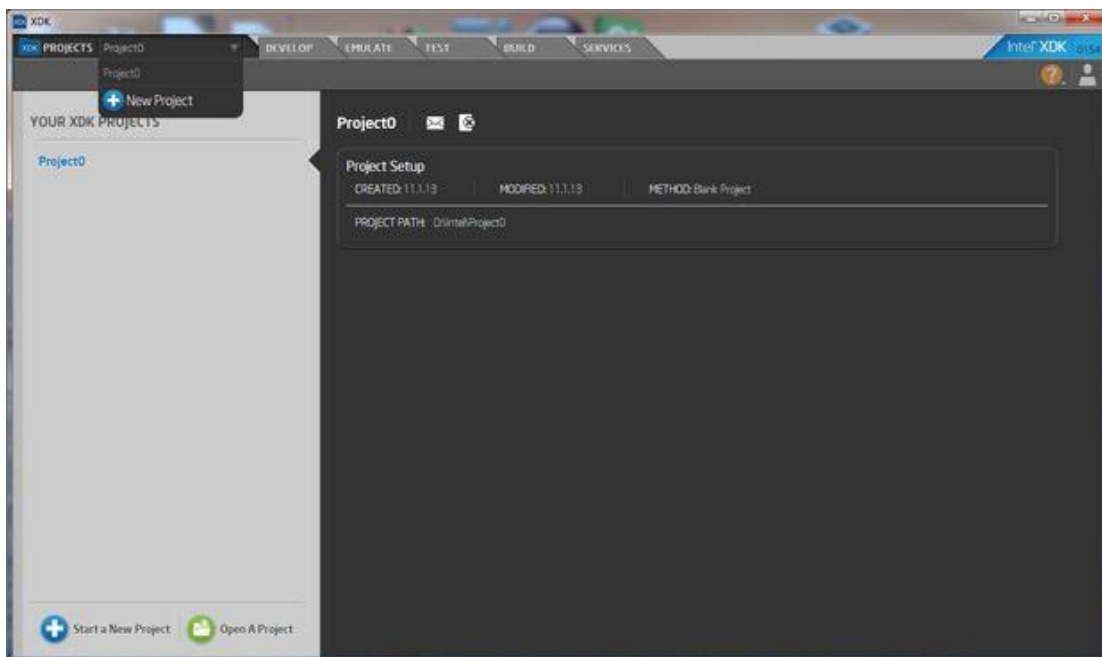


Рисунок 16.7 – Главное окно среды

Задание 2. Создание проекта на основе пустого шаблона

Выбрав пункт «**New Project**», мы попадаем в окно для выбора вида создаваемого приложения. Здесь мы можем создать приложения «с чистого листа»: стартового шаблона

HTML-файла и сценария Javascript. В этом случае достаточно лишь указать имя создаваемого проекта и директорию для его хранения (рис. 16.8). Система дает советы по работе с создаваемым приложением (рис. 16.9).

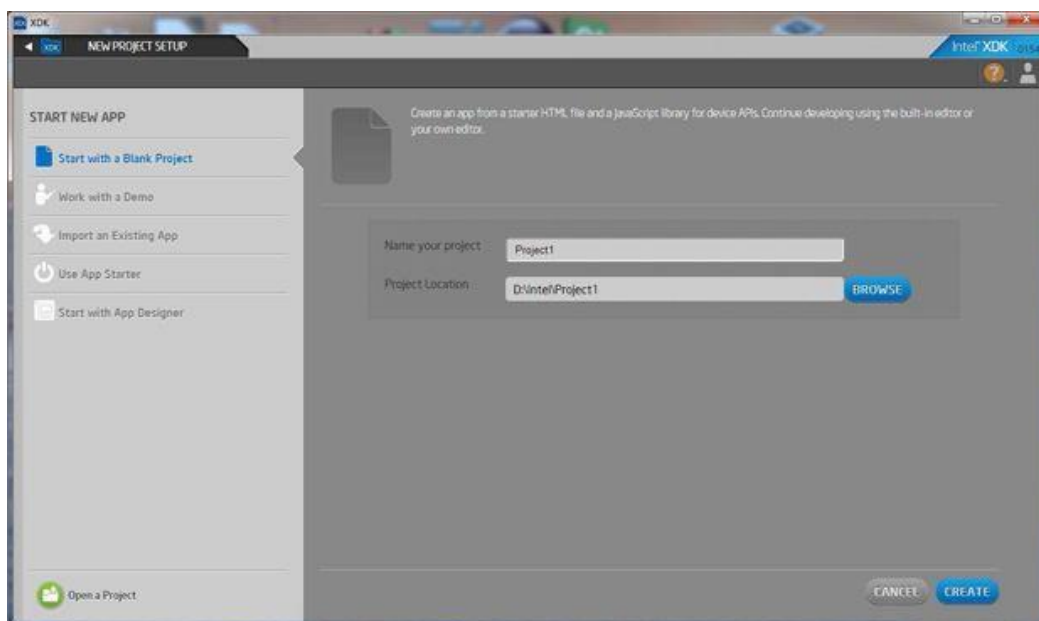


Рисунок 16.8 – Создание нового проекта «с чистого листа»

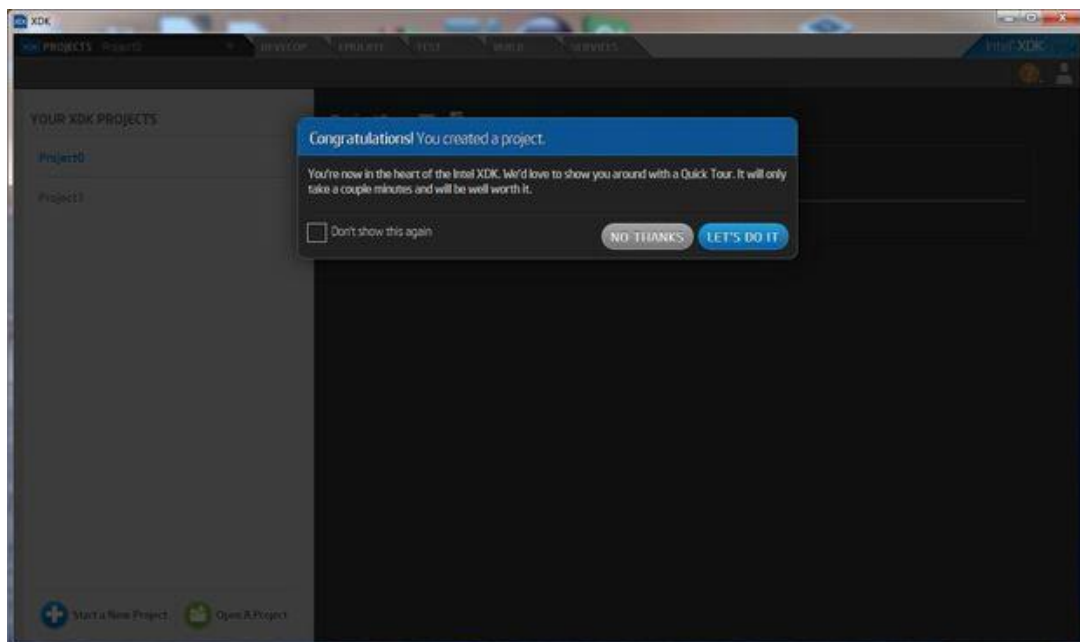


Рисунок 16.9 – Подсказки среды

После этого мы попадаем во вкладку «**Develop**», где можем видеть заготовку нашего будущего приложения (рис. 16.10).

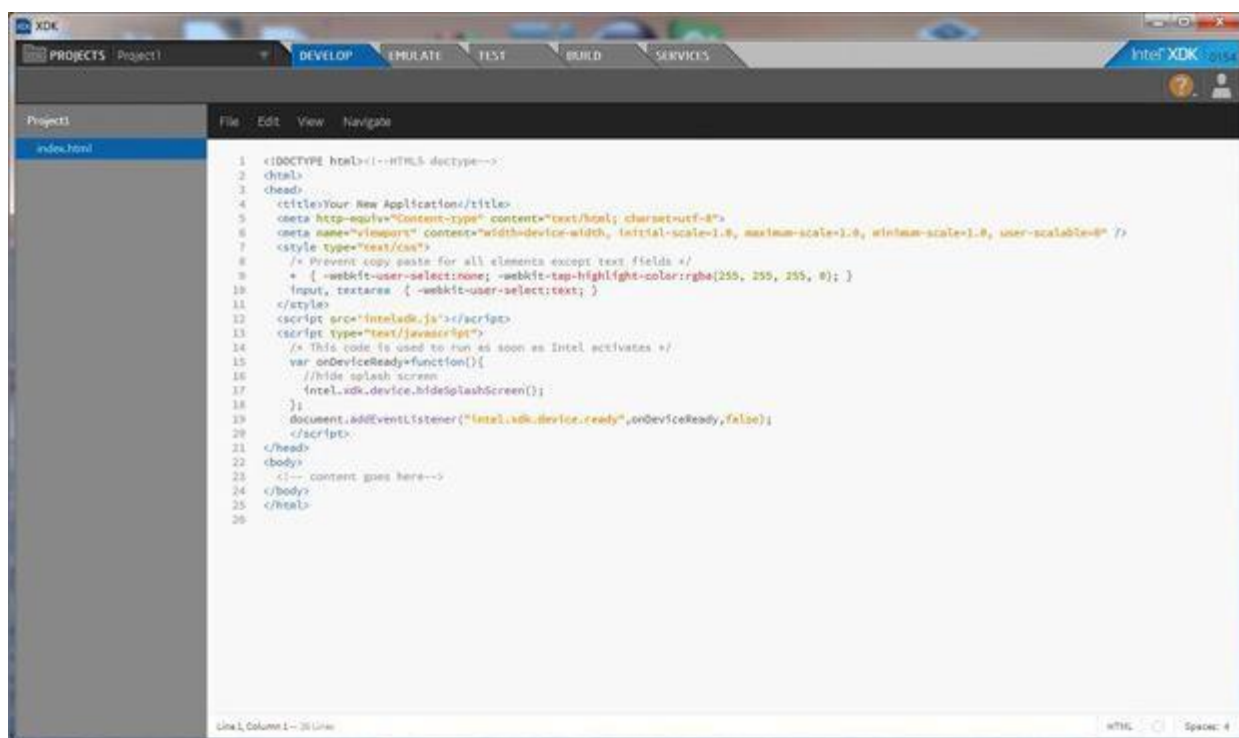


Рисунок 16.10 – Код заготовки будущего проекта

Рассмотрим этот код подробнее. Это практически стандартная web-страница с подключенным скриптом intelxdk.js, отвечающим за основную функциональность приложения. Для того чтобы web-страница функционировала

как мобильное приложение, по умолчанию задан ряд настроек.

Так, сделаны мета-определения для корректной работы в браузере телефона:

```
<meta name=«viewport» content=«width=device-width,
initial-scale=1.0»>
```

Данный код определяет ширину страницы (в данном случае приложения), равной ширине устройства, и задает начальный масштаб.

Кроме того, поставлен запрет на любое масштабирование как начальное при загрузке, так и пользовательское уже после загрузки (maximum-scale=1.0, minimum-scale=1.0, userscalable=0).

С помощью свойств каскадных таблиц стилей реализован запрет на выделение любых

компонентов, за исключением элементов ввода и редактирования текстовых полей:

```
-webkit-user-select:none; и -webkit-user-select:text;
```

За запуск всех функциональных возможностей, реализованных в скрипте intelxdk.js, отвечают строки, реализованные во втором скрипте, явно прописанном в коде приложения:

```
/* This code is used to run as soon as Intel
activates */ var onDeviceReady=function(){
    //hide splash screen
    intel.xdk.device.hideSplashScreen();
};
document.addEventListener(«intel.xdk.device.ready»,on
DeviceReady,false);
```

Программирование на JavaScript программирование каких-либо действий, основанных на событиях. Модель событий DOM2, реализованная в виде иерархической структуры объектов документа HTML, позволяет реализовать стандартные способы создания, захвата, обработки и отмены событий в древовидной структуре типа иерархии.

События начинают жизненный цикл на вершине иерархии (на уровне

Document) и движутся вниз через вложенные объекты к цели. Этот процесс называется фазой захвата.

За это в DOM2 отвечает метод `addEventListener` с тремя аргументами: строковое название события, у нас это `intel.xdk.device.ready`, обработчик события `onDeviceReady`, и логический параметр фаза захвата. Если значение фазы равно `true` то обработчик сработает по мере перемещения вглубь по иерархии, если `false` на обратном «пути».

Как только Javascript будет загружен, событие `intel.xdk.device.ready` будет обработано. Итак, заготовка приложения есть, можно приступать к разработке пользовательских приложений.

Задание 3. Разработка первого приложения

В основе мобильного приложения, разрабатываемого с использованием Intel XDK, как мы уже выяснили ранее, обычная web-страница с таблицей стилей и скриптами javascript для реализации реакций на события. А значит, все наши знания по HTML5, CSS и javascript вполне применимы.

Начнем с классического «**Hello world!**!». Для этого вполне достаточно прописать в теле страницы, в данном случае приложения, например, абзац текста (рис. 16.11): `<p> Hello world!</p>`

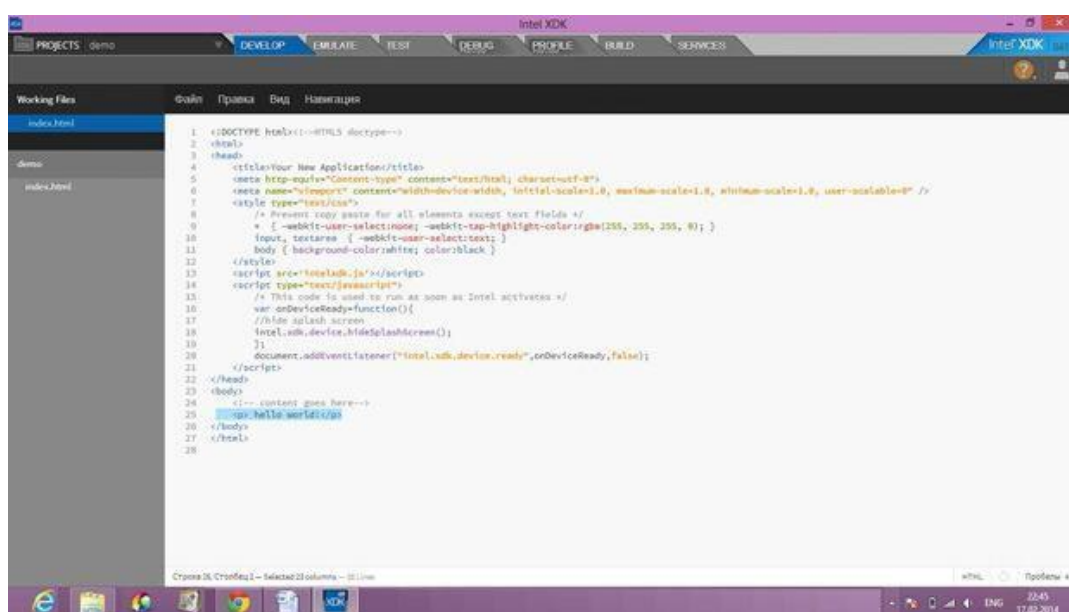


Рисунок 16.11 – Код приложения «Hello world!»

Проверить полученный результат можно с помощью вкладки «**Emulate**»,

где среда позволяет симитировать работу на самых популярных устройствах (рис. 16.12).

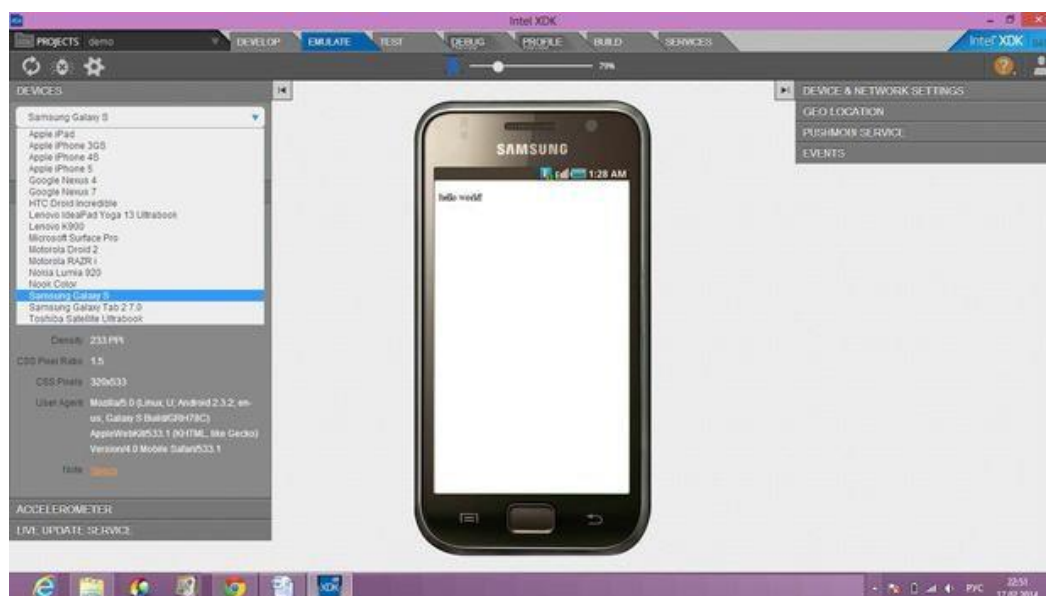


Рисунок 16.12 – Простейшее приложение на эмуляторе

Всем, кто знаком с тройкой **HTML+CSS+Javascript**, открыты огромные просторы для творчества!

Например, почему бы не создать свои «авторские» часы? Отображаться время будет с помощью заголовка 1 уровня, которому мы зададим размер, цвет и выравнивание по центру:

```
<h1 id=«times»> </h1>
```

Стиль прописан в таблице стилей

```
#times {color:red;font-size:50pt;text-align:center;}
```

За «автоматику» часов будет отвечать две функции Javascript.

Функция `startTime` получает текущее системное время и задает его элементу с идентификатором `times`.

```
function startTime() {  
    var d=new Date();  
    var h=d.getHours();  
    var m=d.getMinutes();  
    var s=d.getSeconds(); m=FullTime(m); s=FullTime(s);
```

```

document.getElementById('times').innerHTML=h+»:»+m+»:
»+s; t=setTimeout('startTime()',500);
}

```

Функция FullTime позволяет корректно отобразить время, значение минут и секунд которого занимает 1 разряд (добавляет ведущий нуль)

```

function FullTime(i) {
if (i<10)
{
i=«0» + i;
}
return i;
}

```

Остается добавить эти функции в тег <script type=«text/javascript»> и прописать, чтобы функция была выполнена при загрузке приложения (рис. 16.13).

```
<body onload=«startTime()»>
```

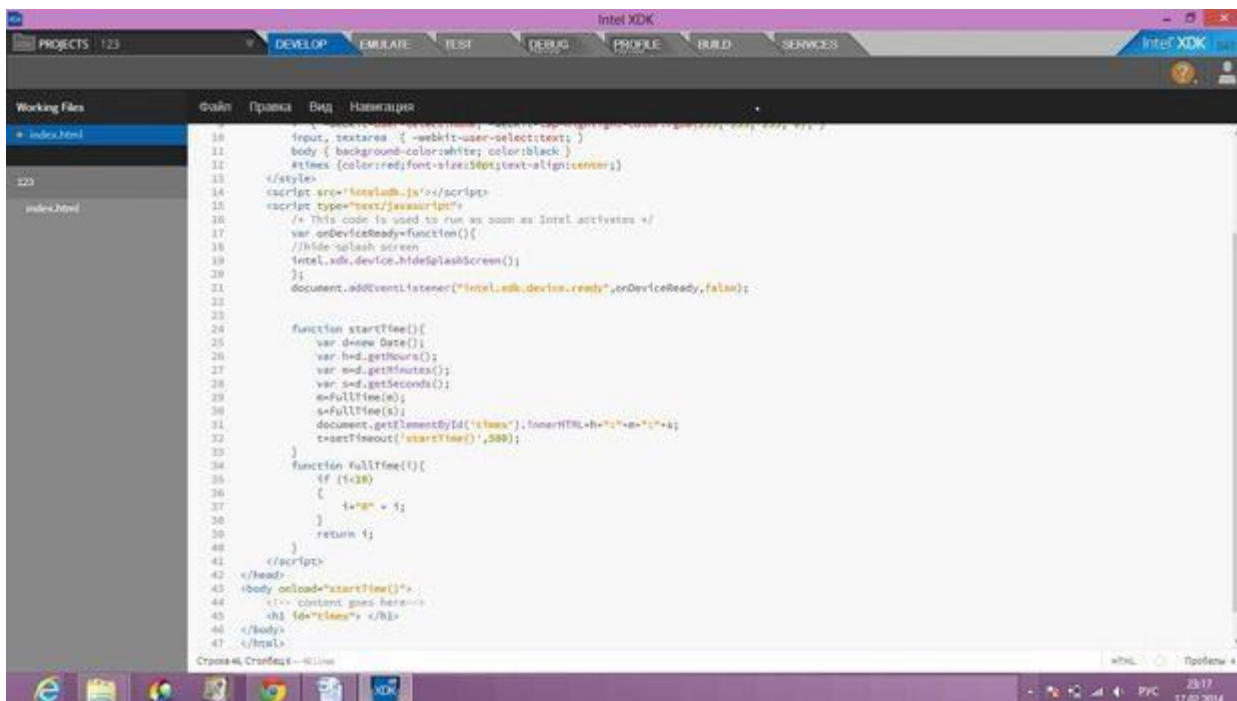


Рисунок 16.13 – Код приложения «Часы»

Проверяем результат работы (рис. 16.14).

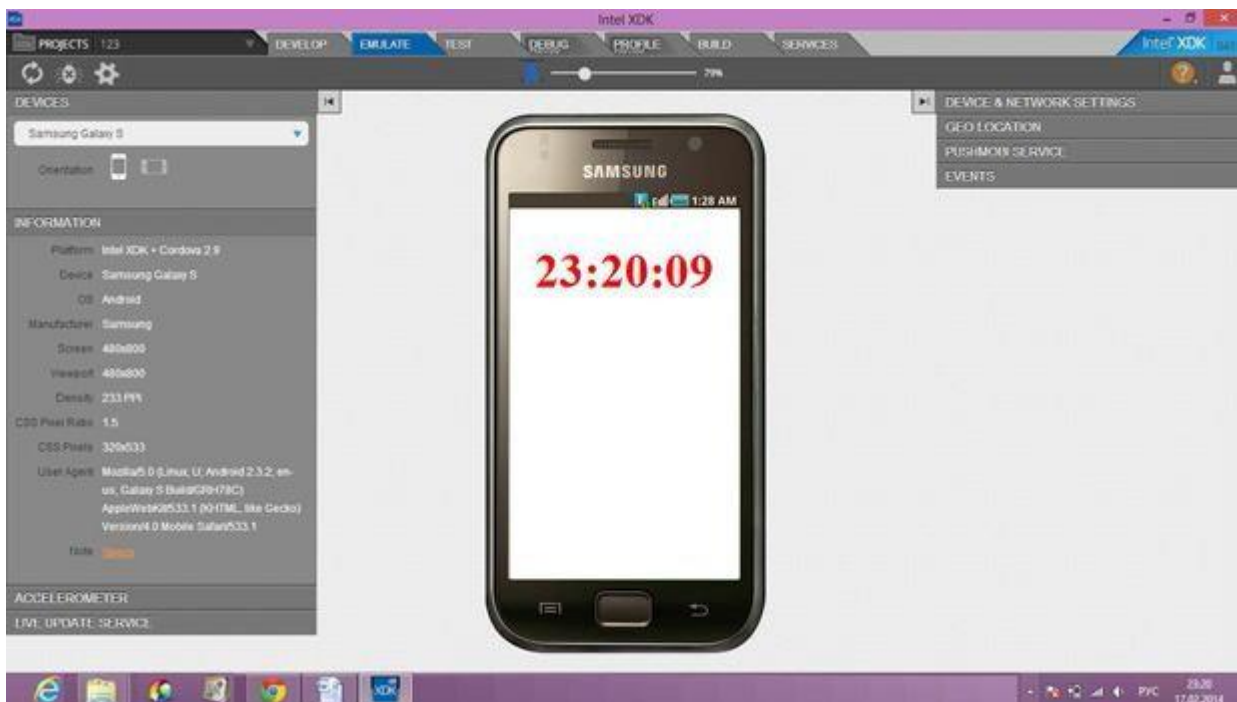


Рисунок 16.14 – Приложение «Часы» на эмуляторе

Задание 4. Разработка приложения на основе шаблона

Разрабатывать приложения можно не только на основе пустого шаблона, но и воспользоваться встроенными примерами и создать на их базе собственное. Для этого достаточно в окне создания приложения выбрать пункт «**Work with a Demo**» выбрать понравившийся пример (рис. 16.15).



Рисунок 16.15 – Выбор встроенного примера

Рассмотрим, например, приложение «**Orientation**». Для этого выберем его в списке и нажмем на кнопку «**Next**». Зададим имя нашему проекту и нажмем кнопку «**Create**». Получим следующий код (рис. 16.16). Рассмотрим его поподробнее.

Для начала заметим, что автоматически были созданы 3 директории: для таблиц стилей, для изображений и для скриптов, а также ряд вспомогательных файлов.

Остановимся подробнее на скрипте **main.js**. Остальные два особого интереса не представляют: intelxdk.js нам уже знаком, встроенный скрипт отвечает за начальные соотношения ширины и длины, настройки портретной и альбомной ориентации.

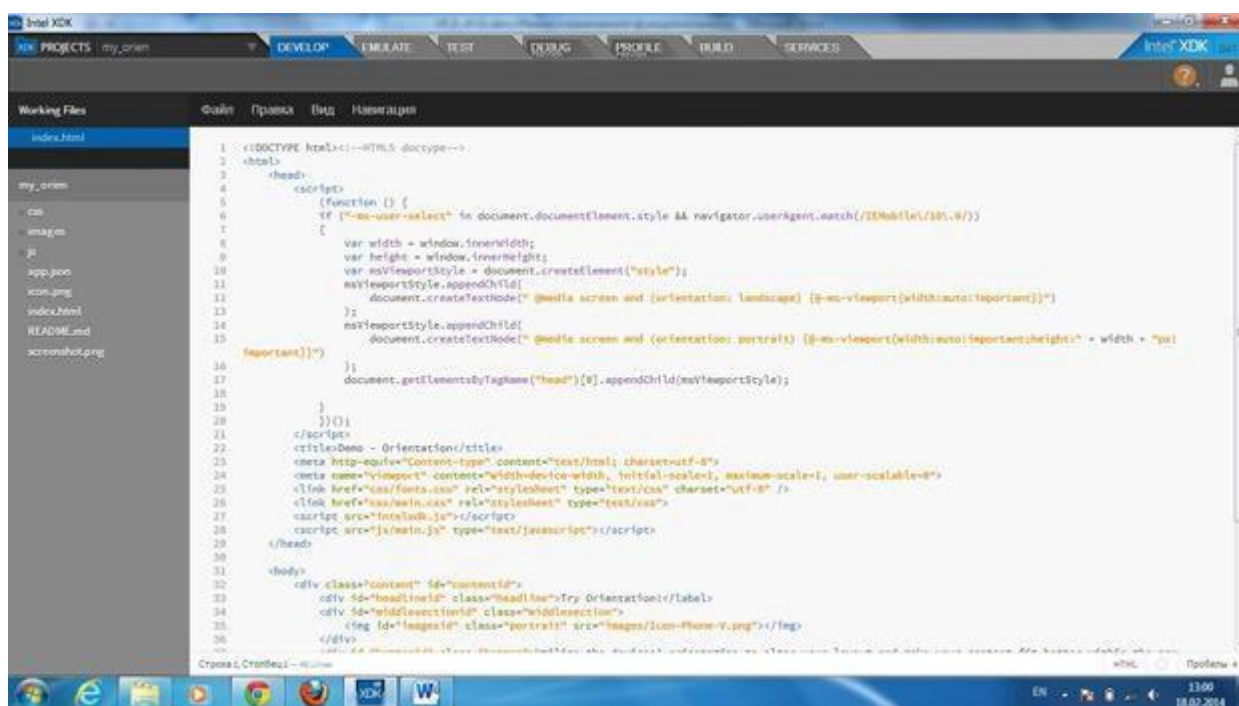


Рисунок 16.16 – Код приложения «Orientation»

Итак, функция **init** начинает свою работу в момент, когда приложение уже загружено, а библиотека JavaScript еще нет и передает значения ширины и высоты для встроенного скрипта.

Далее прописан запрет на изменения размера приложения (скроллинг) как реакция на событие **touchmove**.

```

preventDefaultScroll = function(event)
{
    //      Prevent      scrolling      on      this      element
event.preventDefault(); window.scroll(0,0);
    return false;
};
window.document.addEventListener («touchmove»,
preventDefaultScroll, false);
/*

```

Как только все библиотеки Javascript, в том числе Intel XDK, загружены, возникает событие `intel.xdk.device.ready`. В ее обработчике `onDeviceReady()` помимо описанных в «чистом» шаблоне команд по скрытию окна заставки прописана фиксация текущей ориентации (`intel.xdk.device.setRotateOrientation («any»)`) и возможность отслеживания поворотов (`intel.xdk.device.setAutoRotate(true)`). Команда `intel.xdk.device.managePower(true,false)` не дает приложению перейти в спящий режим.

```

function onDeviceReady()
{
    //lock orientation

    intel.xdk.device.setRotateOrientation («any»);
intel.xdk.device.setAutoRotate(true);

    //manage                                     power
intel.xdk.device.managePower(true,false);

    //hide                                     splash                                     screen
intel.xdk.device.hideSplashScreen();
}

```

Теперь можно заняться отслеживанием события «Поворот» и изменением

свойств изображения `imagesid` и трех текстовых элементов-слоев `headlineid`, `bottomed`, `endingid`. За это у нас будет отвечать обработчик `intel.xdk.device.orientation.change`.

За портретную ориентацию отвечают значения 0 и 180, за альбомную 90 и 270 в соответствии со значением градусов поворота. Остается в зависимости от текущего значения ориентации задать соответствующие адреса изображений для `imagesid` и имена классов для объектов `imagesid`, `headlineid`, `bottomed`, `endingid` одни для альбомной, другие для портретной. Все стилевые свойства прописаны классами в подключенной таблице стилей **main.css**.

```
function orientationchange(value)
{
  //Portrait orientation
  if(value.orientation == 0 || value.orientation == 180)
  {
    document.getElementById(«imagesid»).className=«portrait»;
    document.getElementById(«imagesid»).src=«./images/Icon-
    Phone-V.png»;
    document.getElementById(«headlineid»).className=«headline
    »;
    document.getElementById(«bottomid»).className=«bottom»;
    document.getElementById(«endingid»).className=«ending»;
  }
  //Landscape orientation
  else
  {
    document.getElementById(«imagesid»).src=«./images/Icon-
    Phone-H.png»;
    document.getElementById(«imagesid»).className=«imglandsca
    pe»;
    document.getElementById(«headlineid»).className=«headline
    headlinelandscape»;
```

```

document.getElementById(«bottomid»).className=«bottom
bottomlandscape»;
document.getElementById(«endingid»).className=«ending
endinglandscape»;
}
}

```

Проект готов, проверен в режиме эмуляции, можно приступить к его тестированию и отладке, сборке и компиляции.

Задание 5. Тестирование и отладка на мобильном устройстве

Для реализации тестирования и отладки на мобильном устройстве необходимо установить

на последнем приложение App Preview. Скачать его можно с помощью Apple AppStore, Google Play Store, Windows Phone Store. Также необходимо наличие на мобильном устройстве приложения по чтению QR-кодов. Остается перейти во вкладку «**Test**», нажать «**Push files**» для отправки файлов на сервер intel XDK и прочитать QR-код мобильным устройством (рис. 16.17).

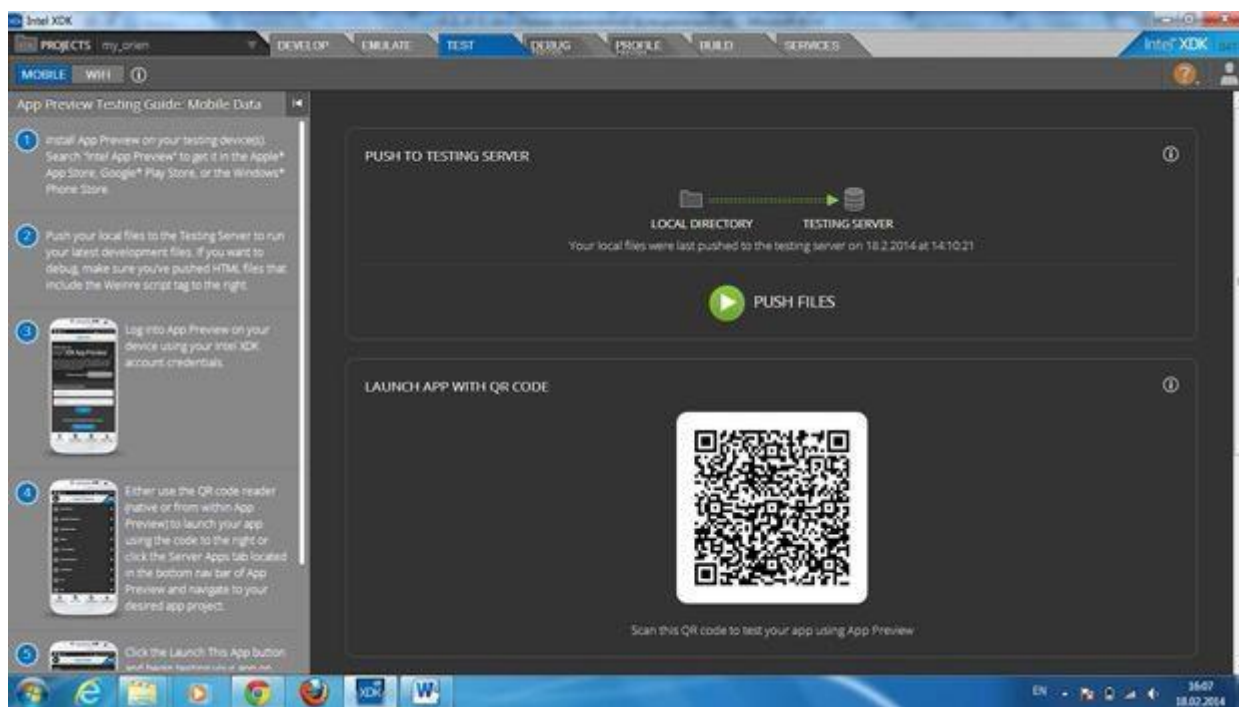


Рисунок 16.17 – Код приложения «Orientation»

Код распознает гиперссылку, ресурс скачивается и автоматически

запускается программой App Preview. В зависимости от скорости Интернет-соединения это займет некоторое время. После это приложение будет запущено. Можно экспериментировать (рис. 16.18).



Рисунок 16.18 – Запуск приложения на мобильном устройстве

Задание 6. Сборка и компиляция приложения

Сборка и компиляция приложения предполагает сборку арк-файла. Для этого переходим во вкладку «**Build**» и выбираем тип операционной системы на нашем мобильном устройстве (рис. 16.19).

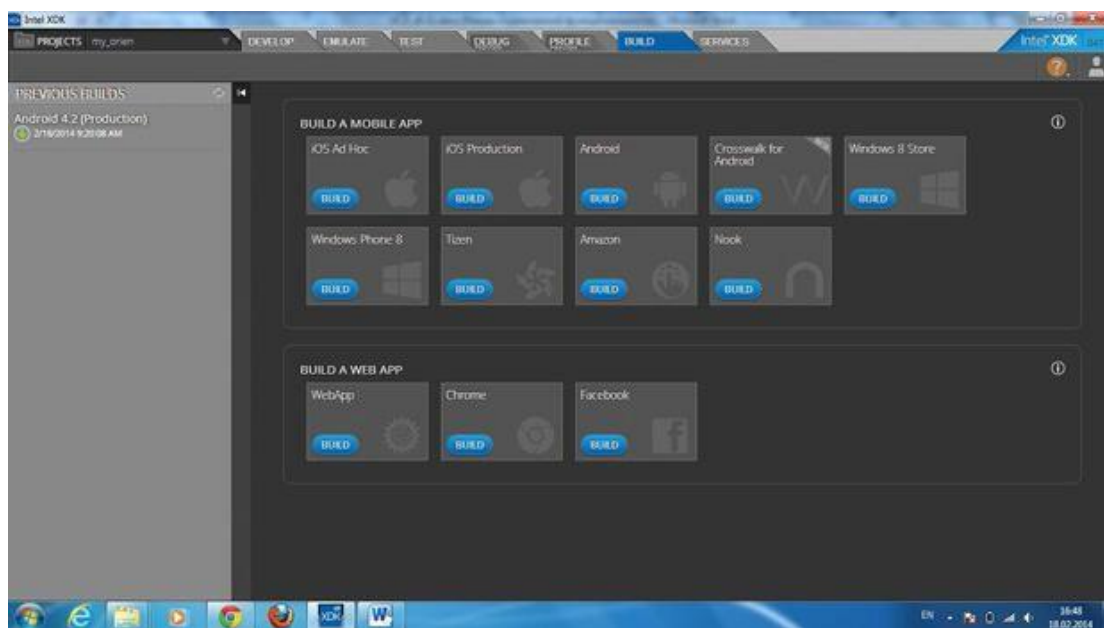


Рисунок 16.19 – Выбор ОС мобильного устройства

Нажимаем кнопку «**Build App Now**» и начинаем процесс сборки приложения. Здесь существует возможность тонкой настройки создаваемого приложения, ограничимся настройками по умолчанию (рис. 16.20). После завершения процесса сборки и компиляции скачиваем полученный арк-файл на мобильное устройство любым из способов (рис. 16.21).

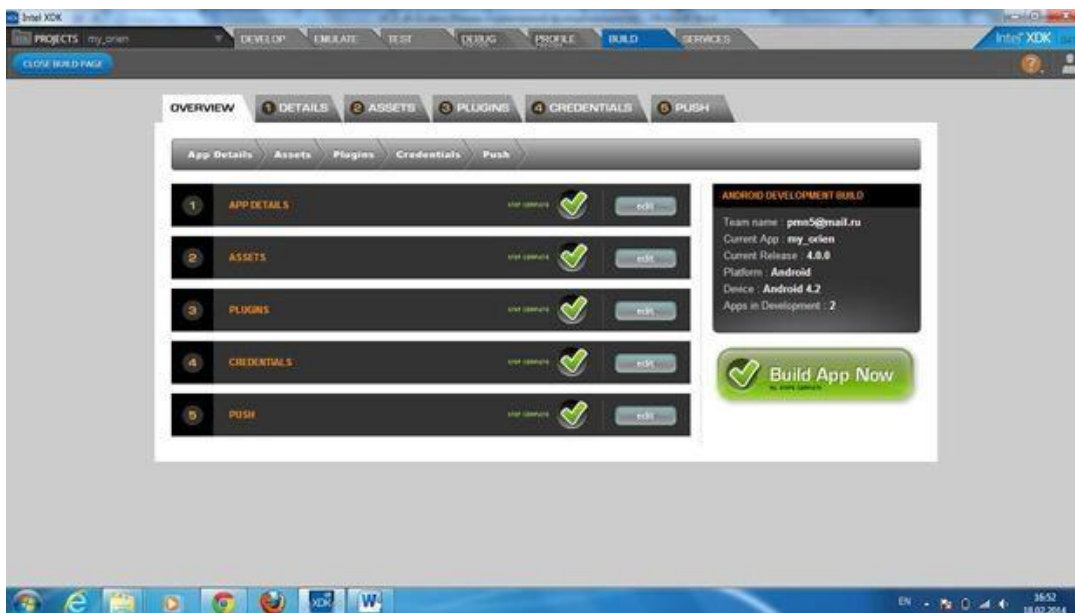


Рисунок 16.20 – Сборка и компиляция приложения

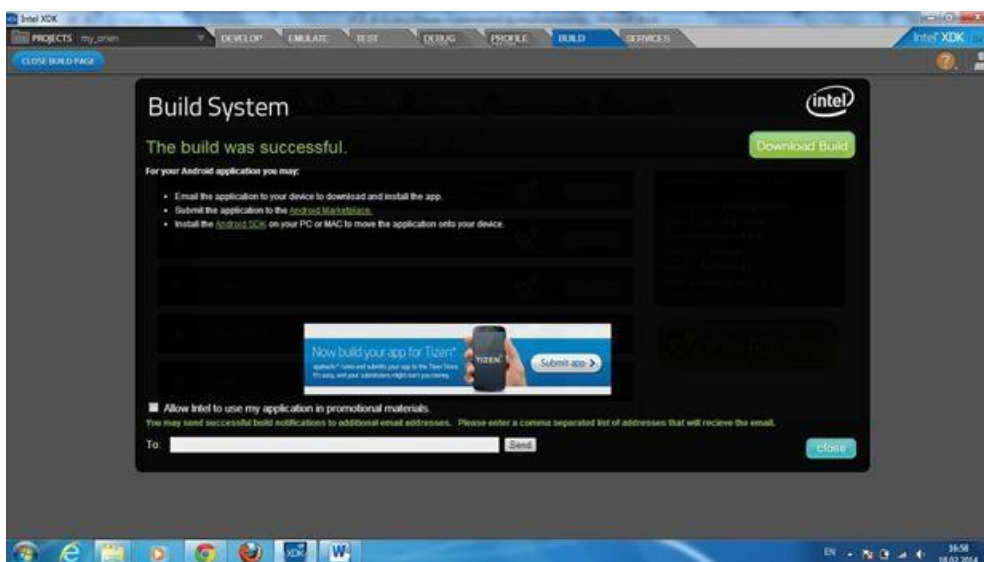


Рисунок 16.21 – Проект скомпилирован

Установка и запуск приложения на мобильном устройстве не представляет большого труда.

Для установки приложения выполняем стандартный набор действий (рис. 16.22).

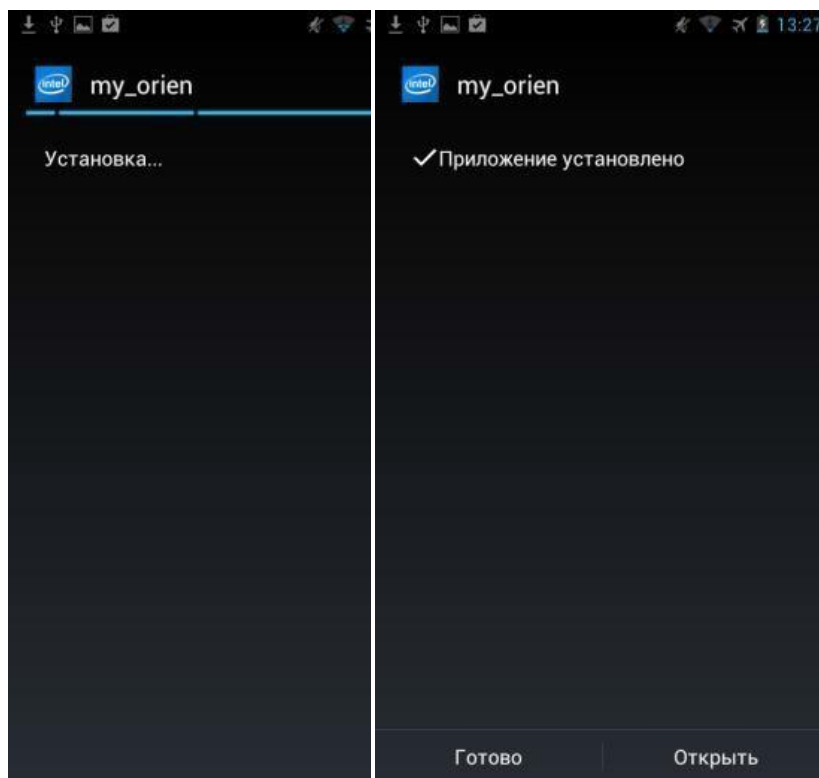


Рисунок 16.22 – Установка приложения

После установки приложения находим его иконку и запускаем (рис. 16.23).

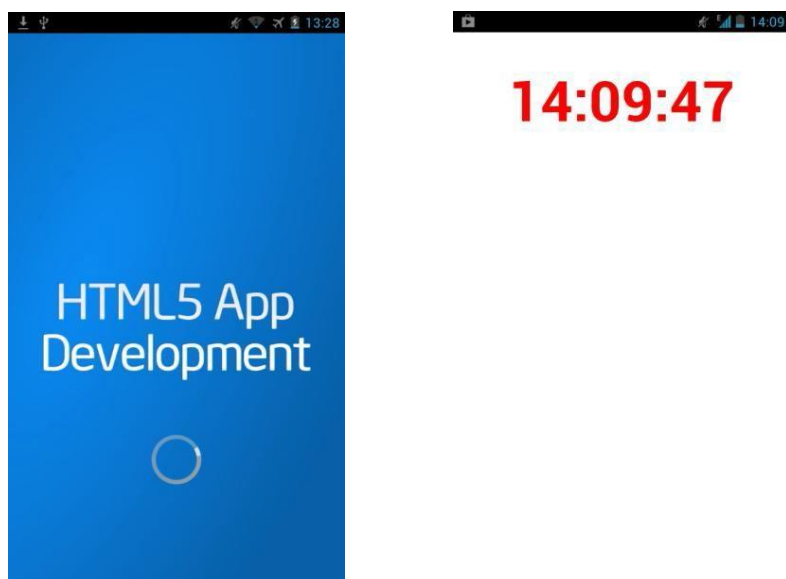


Рисунок 16.23 – Запуск приложения «Часы» на мобильном устройстве

Содержание отчета и его форма

Отчет к лабораторной работе оформляется в виде текстового документа по форме простого реферата и должен включать:

1. Название лабораторной работы.
2. Цель лабораторной работы.
3. Краткие выводы по результатам выполнения лабораторной работы согласно пунктам раздела «Методики выполнения работы».
4. Ответы на Вопросы к лабораторной работе.
5. Формулировку задания для самостоятельной работы и результат его выполнения.

Задания для самостоятельной работы

1. На основе примера «**Orientation**» разработать приложение с анимированным человечком: в режиме портрета он занимается спортом, в режиме альбома сладко спит.

2. Предлагаем вам проявить творчество и разработать небольшое приложение на основе пустого шаблона и на основе любого из примеров.

Листинг 1. Приложение «Часы»

```
<!DOCTYPE html><!--HTML5 doctype-->
<html>
<head>
<title>Your New Application</title>
<meta http-equiv=«Content-type» content=«text/html;
charset=utf-8»>
<meta name=«viewport» content=«width=device-width,
initial-scale=1.0,
maximum-scale=1.0, minimum-scale=1.0, user-
scalable=0» />
<style type=«text/css»>
/* Prevent copy paste for all elements except text
fields */
* { -webkit-user-select:none; -webkit-tap-
```

```

highlightcolor:rgba(255, 255, 255, 0); }
    input, textarea    { -webkit-user-select:text; }
    body { background-color:white; color:black }
    #times {color:red;font-size:50pt;text-align:center;}
</style>
<script src='intelxdk.js'></script>
<script type=«text/javascript»>
/* This code is used to run as soon as Intel activates
*/
var onDeviceReady=function(){
//hide splash screen
intel.xdk.device.hideSplashScreen();
};
document.addEventListener(«intel.xdk.device.ready»,on
DeviceReady,false); function startTime(){
var d=new Date(); var h=d.getHours();
var m=d.getMinutes();
var s=d.getSeconds();
m=FullTime(m);
s=FullTime(s);
document.getElementById('times').innerHTML=h+»:»+m+»:
»+s; t=setTimeout('startTime()',500);
}
function FullTime(i){
if (i<10)
{
i=«0» + i;
}
return i;
}
</head>

```

```
</script>
  <body onload=«startTime()»>
<!--content goes here-->
<h1 id=«times»> </h1>
</body>
</html>
```

Вопросы к лабораторной работе

1. Технология установка и настройка среды программирования Intel XDK.
2. Технология создания проекта на основе пустого шаблона.
3. Технология разработки приложения на основе шаблона.
4. Технология тестирования и отладки на мобильном устройстве.
5. Технология сборки и компиляции приложения.

Список рекомендуемой литературы

Основная литература

1. Брыксина, О. Ф. Информационно-коммуникационные технологии в образовании: учебник / О.Ф. Брыксина, Е.А. Пономарева, М.Н. Сони́на. – Москва: ИНФРА-М, 2019. – 549 с. – (Высшее образование: Бакалавриат). – www.dx.doi.org/10.12737/textbook_59e45e228d2a80.96329695. – ISBN 978-5-16-012818-4. – Текст: электронный. – URL: <https://znanium.com/catalog/product/1025485>.

2. Шишов, О. В. Современные технологии и технические средства информатизации: учебник / О.В. Шишов. – М.: ИНФРА-М, 2019. – 462 с. + Доп. материалы [Электронный ресурс; Режим доступа <http://www.znanium.com>]. – (Высшее образование: Бакалавриат). – ISBN 978-5-16-011776-8. – Текст: электронный. – URL: <https://znanium.com/catalog/product/1002745>.

Дополнительная литература

3. Сысоева, Л. А. Управление проектами информационных систем: учеб. пособие / Л.А. Сысоева, А.Е. Сатунина. — Москва: ИНФРА-М, 2019. — 345 с. — (Высшее образование: Бакалавриат). — www.dx.doi.org/10.12737/textbook_5cc01bbf923e13.56817630. - ISBN 978-5-16-013775-9. - Текст: электронный. - URL: <https://znanium.com/catalog/product/953767>.

4. Пушкарёва, Т. П. Основы компьютерной обработки информации: Учебное пособие / Пушкарёва Т.П. - Краснояр.: СФУ, 2016. - 180 с.: ISBN 978-5-7638-3492-5. - Текст: электронный. - URL: <https://znanium.com/catalog/product/967586>.