Иван Андреевич Трещев

Программирование для мобильных платформ. Android

Для студентов технических специальностей

Иван Андреевич Трещев

Программирование для мобильных платформ. Android. Для студентов технических специальностей

http://www.litres.ru/pages/biblio_book/?art=50282751
ISBN 9785449812971

Аннотация

Данное пособие содержит основные сведения о создании приложений с использованием Eclipse для мобильной платформы Android.

Содержание

Программирование для мобильных платформ. Android Для студентов технических специальностей

Иван Андреевич Трещев

ISBN 978-5-4498-1297-1
Создано в интеллектуальной издательской системе Ridero

ВВЕДЕНИЕ

Разработка мобильных приложений сегодня – неотъемлемая часть работы любого предприятия в сфере IT. Стоит отметить, что для того, чтобы начать разрабатывать приложения для платформы Android необходимо оплатить единоразовый взнос – на 2019 год 50$. Что по сравнению с аналогичной разработкой приложений для платформы Apple в два раза меньше и его нет необходимости оплачивать ежегодно. Хотя опыт показывает, что приложения размещаемые в App Store проходят более тщательную проверку, нежели приложения размещаемые в Google Play.

Лаборатория которой руководил автор на протяжении 5 лет занималась разработкой различных приложений для самых популярных за последнее пятилетие операционных систем носимых устройств – Android, IOS, Windows Phone. Сегодня платформа Android насчитывает многомиллионную аудиторию и располагает одним из самых удобных и эргономичных способов для авторов (будь то песни, книги или приложения) для монетизации своих творений при этом не неся затрат на тиражирование, продажу, экспозицию и другие накладные расходы.

Данная книга посвящена разработке приложений именно под платформу от Android и является третьей в цикле.

У читателя предполагается опыт программирования

на объектно-ориентированном языке, желательно опыт на Java.

Основы Java

1 Типы данных

Элементарные типы данных и переменные

Java предоставляет множество элементарных типов данных. Под каждый тип данных выделяется строго фиксированное количество бит, определенное в официальной документации. Еще одним отличием Java от других языков является тот факт что все типы данных содержат бит выделенный под знак и избавится от него нельзя. В таблице представлены параметры всех типов данных.

Имя	назначение	Количество выделенной памяти, бит
byte	целочисленный	8
short	целочисленный	16
int	целочисленный	32
long	целочисленный	64
char	символьный/ целочисленный	16
double	С плавающей точкой	64
float	С плавающей точкой	32
boolean	булев	-

Переменная – это область памяти, выделенная для хранения какого-либо значения.

Литералы

Литералами называют константные значения, представленные в явном виде. Например:

int i = 5; // 5-литерал

Для каждого литерала выделяется отдельная область памяти. Разберем сколько именно памяти выделяется под тот или иной тип литерала.

В случае если мы представили литерал в целочисленном виде, то для него выделяется 32 бита, а сам литерал имеет

тип int. Но в случае если переменной меньшего размера присваивается целое число, оно автоматически конвертируется в значение той переменной.

Символьный литерал имеет тип char и обозначается в одинарных кавычках:

char ch = «h’; // символьный литерал

все дробные числа изначально имеют тип double. Если попробовать присвоить дробное число переменной типа float, то это вызовет ошибку. Простой способ присвоить число переменной float – это конце числа подставить символ *F:*

float f = 2.0F;// переменная типа float

Так же в Java присутствуют булевы тип. Литералы этого типа имеют всего два значения: *true* и *false*. В случае присвоения другого значения булевой переменной, компилятор выдаст ошибку.

2 Класс, объект, метод

Введение в ООП

Java является объектно-ориентированным языком программирования. Определяющими понятиями данной парадигмы программирования можно назвать *класс и объект.*

Класс – это тип данных определяемый разработчиком.

Объект – это экземпляр класса. Давайте разберем более подробно, что это значит.

В классе мы задаем поведение и все необходимые параметры (переменные, объекты других классов) и поведение (методы и конструкторы) объекта данного класса. При создании и инициализации объекта выделяется необходимый объем памяти, ровно столько сколько было прописано в классе, так же объекты вынуждены подчиняться поведениям, описанных в классе.

Приведу пример: допустим нам необходимо реализовать класс учебной группы. В качестве параметров можно принять количество студентов (может выступать переменная int numberStud), наименование группы (например объект класса String name), так же необходимо указать какие именно студенты будут входит в состав группы (можно сделать массив объектов класса студентов, который в свою очередь тоже имеют свои параметры и свое поведение). в процессе существования группа может: поменять имя, поменять коли-

чество студентов, прекратить свое существование и другое. Это и называется поведение объекта. В процессе написания программы программист всячески совершает над объектами различные действия. Все эти действия необходимо прописать в нашем классе, иначе они попросту не смогут выполняться.

Еще одними из определяющими понятиями в ООП (не только в Java) являются: *полиморфизм, наследование и инкапсуляция.* Это своего рода принципы (свойства) на которых построена парадигма ООП, именно на них она и базируется.

Наследование

В рамках объектно-ориентированного программирования любой класс может унаследовать некоторые свойства от другого класса. Такой принцип называется наследованием. В качестве свойств могут выступать методы, переменные и тд.

Разберем пример: пусть нам необходимо описать породу собаки в нашем классе. Существуют довольно много собачьих пород, которые в свою очередь обладают уникальными свойствами. Для каждой породы нам придется написать свой собственный класс. В то же время все породы обладают параметрами, присущие абсолютно для всех собак. У нас есть вариант описать абсолютно все свойства в каждом классе, в том числе уникальные для каждой породы и универ-

сальные, присущие для всех пород, но это займет много кода и вовсе усложнит задачу. Можно просто создать главный класс собак, описать в нем параметры и поведение, которые есть у всех и класс, описывающую конкретную породу унаследовать от главного.

Инкапсуляция

При создании классов важно помнить об уровне защиты компонентов. Важно не допускать, чтобы код, содержащийся вне класса, не влиял без надобностей на данные класса. Это может произойти, например, по вине программиста, в особенности, если он работает в команде, и по другим причинам. Последствия могут быть довольно плачевные. Суть инкапсуляции состоит в том, что для данных и кода, который работает с данными, условно создается рабочее пространство. Тем самым программу можно разделить на несколько систем, внутри которых происходит вся работа с данными, но снаружи никак не влияющие на работу другой системы. Это делается для того, чтобы защитить данные от плохого влияния стороннего кода.

Конечно же программно эти пространства никак не описываются. Защищенность данных достигается путем присваивания *модификаторов доступа*.

Как пример могу привести работу автомобиля. Его можно воспринимать как программу. Разберем работу коробки передач. Она хранит в себе множестве сведений: максималь-

ное количество передач, текущая включенная передача, конструкция – все это можно назвать данными. В ходе работы в ней происходят множество процессов, которые изменяют эти данные. При этом коробка не влияет, например, на систему световых приборов. Обе эти системы работают независимо друг от друга. В этом и есть суть инкапсуляции.

Полиморфизм

Полиморфизм для понимания считается самым сложным принципом. С точки зрения полиморфизма для объектов схожих классов можно задать одинаковое поведение.

Например: представим класс реализующий круг (в качестве параметров можно указать радиус, площадь, периметр; его можно увеличивать, уменьшать, менять цвет контура или фона, выводить на экран – это и будет являться поведением круга) и представим еще один класс реализующий квадрат (в качестве параметров так же можно принять площадь, длина ребра и тд.; его так же можно увеличивать, уменьшать, менять цвет фона и контура и тд.).

Круг и квадрат являются схожими элементами (классами), а значит и поведение для них может быть задано одинаковое. Оба этих класса можно унаследовать от класса реализующие геометрические фигуры. Далее в родительском классе создать метод изменения цвета фона. И к объектам дочерних классов, в данном случае это квадрат и круг, можно применять этот метод. В этом случае поведение этих фи-

гур будет одинаково, и в том, и в другом случае фон будет меняться, хоть и более подробная реализация этого метода в каждом дочернем классе может отличаться, так как сами фигуры разные.

Можно привести другой пример: в случае если создать объект родительского класса, а инициализировать его как объект дочернего, то это не вызовет никакой ошибки. Программа корректно обработает данный код. В случае нашего примера можно создать массив объектов геометрических фигур (родительского класса) и потом рандомно присваивать разным ячейкам этого массива разные дочерние классы, то это так же не вызовет никакой ошибки.

В этом и есть идея полиморфизма: объекты схожих классов обрабатываются одинаково. Однако если изначально создать массив объектов одного из дочерних классов, а потом к одной из ячеек присвоить объект другого дочернего класса, то это вызовет ошибку. Это вполне логично, так как при более детальной реализации эти классы разные.

Методы

В рамках ООП поведение объекта задают *методы*, которые так же описаны должны быть описаны при создании класса. Метод содержит в себе набор процедур, которые выполняются над объектом.

Метод аналогично функции в C++ может принимать или не принимать аргументы. В листинге указан способ созда-

ния метода, *тип_метода* – тип возвращаемого методом значения, *тип1* и *тип2* – типы данных первого и второго аргументов.

```
тип_метода имя_метода (тип1 имя_аргумента1, тип2 имя_аргумента2, …)
{
// тело метода
}
```

Если метод возвращает значение, то в его теле должен присутствовать оператор *return имя_значения*. Этот оператор возвращает *имя_значения* и прекращает выполнение метода. Важно чтобы переменная *имя_значения* имела тот же тип данных что и *тип_метода* и была инициализирована.

Если метод не возвращает значение, то *тип_метода* будет равен *void*, в этом случае присутствие оператора *return* вызовет ошибку. В листинге представлены примеры создания метода.

```
void changeName (int index, String Key) //объявление метода
{
// тело метода
}
//////////////////////////////////////////////////////////////
String getName () //создание метода nbgf String
{
String name;
```

```
// тело метода
return name;//возврат значения
}
```

Выполнить метод можно только над объектом внутри класса, которого он объявлен. Или внутри самого класса.

В случае если метод необходимо выполнить внутри другого класса, то указывается объект, над которым следует выполнить этот метод, как показано в листинге.

```
changeName (45, «GH4297K») //вызов внутри класса в котором метод объявлен
//или
String NAME = student.changeName () //student является объектом класса в //котором объявлен метод
```

В случае если метод вызывается внутри самого класса, то указывать объект не необходимо, так как и так понятно какой объект имеется в виду. Ошибкой не будет считаться, если вы в качестве объект укажите оператор *this.* Так же необходимо чтобы тип данных всех аргументов соответствовал типу данным значений, которые передаются методу в качестве аргумента.

```
class Student
{
int age;
string university;
string name;
float schoolperformance;
```

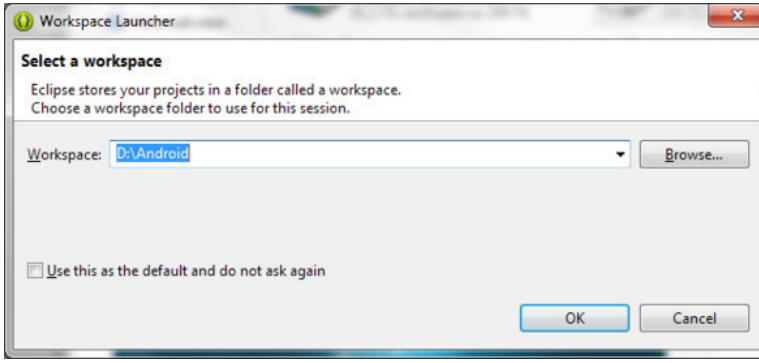
```
void changeName ()
{
// метод изменения имени
}

void incAge ()
{
// метод инкремента возраста
}
void delete ()
{
// удаление студента
}}
```

Запуск

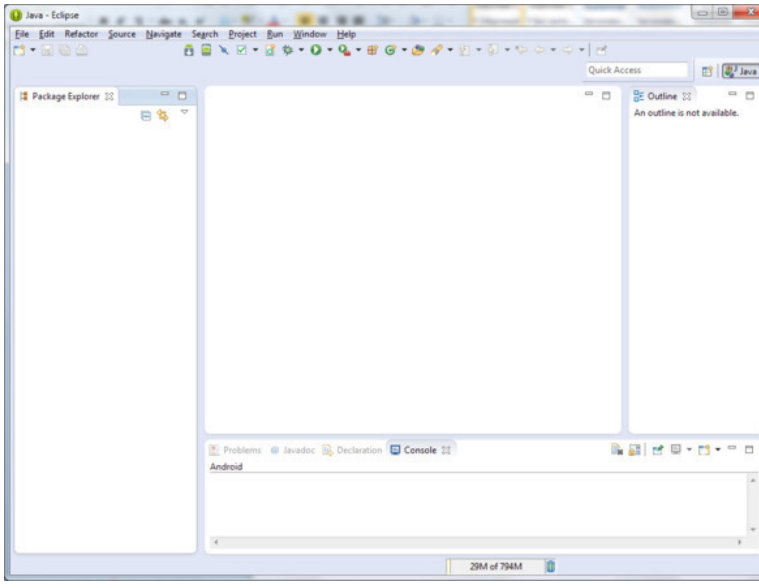
1 Запуск Eclipse

Для создания приложений нам понадобится Eclipse с предустановленными плагинами ADT (Android Developer Tools). Скачать его можно на официальном сайте разработчиков под Android. Так как разработка проходят с помощью языка Java, то нам еще понадобится JDK, его можно найти на официальном сайте Oracle. У вас может появиться проблема с JDK или JRE. В этом случае создайте переменную среду JAVA_HOME и укажите путь JDK.

При первом запуске, Eclipse предложит выбрать папку на компьютере, где будут храниться ваши проекты.



И так нам удалось запустить Eclipse. Выглядит он примерно так:

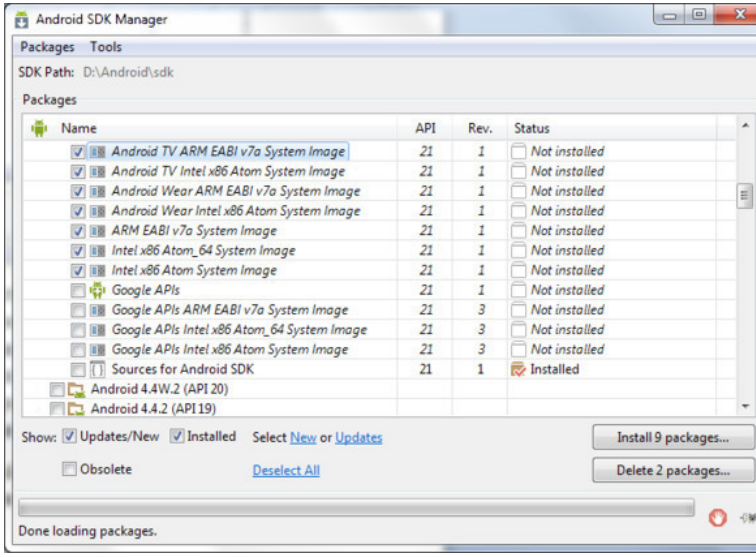


Это и есть наше рабочее пространство. Но с этого момента могло показаться, что можно создавать проект и создавать полезные приложения, но это не совсем так. Дело в том, что нам необходим инструментарий для разработки (Android SDK). Он представляет множество библиотек, файлов и ресурсов для программирования приложений и возможности взаимодействия с Android API и все что связано с ним.

И так заходим во вкладку Windows, далее Android SDK Manager, Мы видим совершенно простой интерфейс ок-

на. Нам показаны списки инструментов, укомплектованные в иерархии. При этом установленные файлы отмечаются. Среди иерархий можно проследить названия, например: «Android 5.0 (API 21)», это значит, что файлы этой иерархии представляют собой инструменты и библиотеки для создания приложений под Android 5. При этом это приложение будет работать и на более поздних версиях, обратная совместимость не всегда действует. Номер API присваивается к каждой версии Android индивидуально. При этом API может обновляться, в этом случае SDK Manager скажет вам об этом. Следите за обновлениями, периодически запуская SDK Manager.

Отмечаем необходимое SDK, мы не рекомендем скачивать сразу абсолютно все API, с большинством из них вы не будете работать, так как версии Android устарели. В целях обучения можно скачать, например API 14 и еще несколько. Нажимаем на установку, далее выделяем все галочками и соглашаемся, ждем.

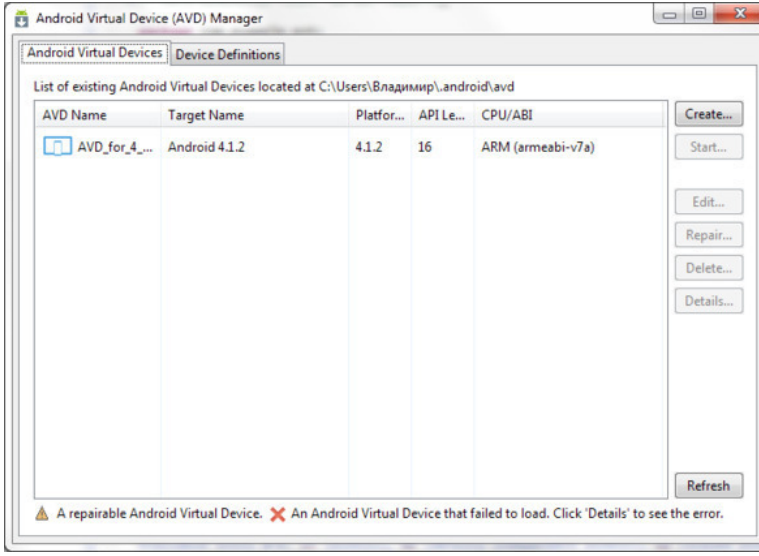


После установки необходимых вам API, ваша IDE уже полностью готова с созданию приложений.

2 Тестирование приложения

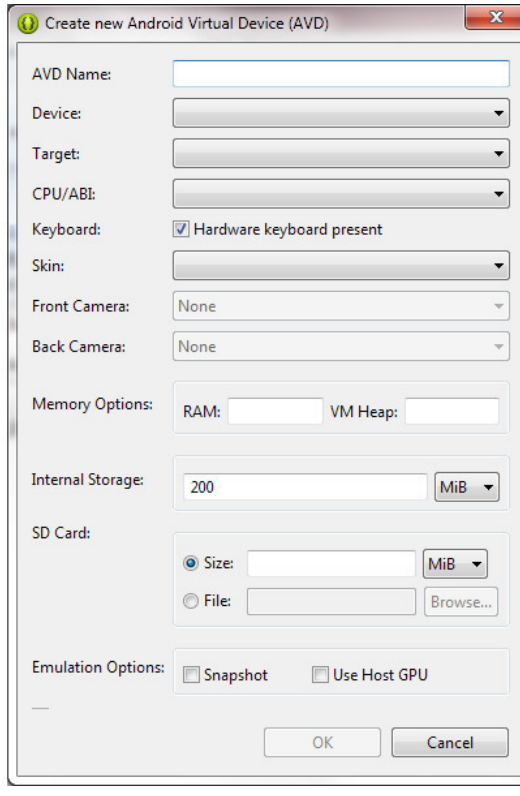
Однако предлагаю подумать, где вы будете тестировать его. Для этого Google предложил Android Virtual Device (AVD). Он представляет собой виртуальную машину с предустановленным Android OS.

Для создания AVD, заходим во вкладку Windows, выбираем Android Virtual Device Manager.



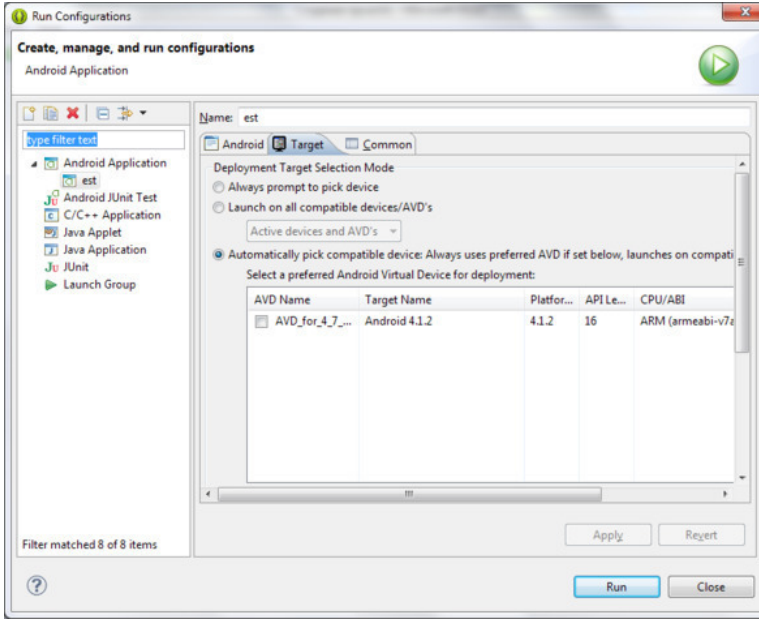
В нем нажимаем Create. Появиться конструктор для со-

здания AVD. Заполняем поля. Target указывает на версию предустановленной ОС; Device на размеры и разрешение экрана; CPU/ABI выбираем ARM, но если для данной версии ОС есть x86 и у вас установлена виртуальная виртуализация, то можно выбрать x86, AVD будет работать быстрее; Skin ставим на no skin. Все остальное можно оставить без изменений. Если все необходимые поля заполнены, то нажимаем на Ок. На этом создания AVD законченно. Если хотите, то можно запустить его, нажав на start в Android Virtual Device, предварительно выделив.



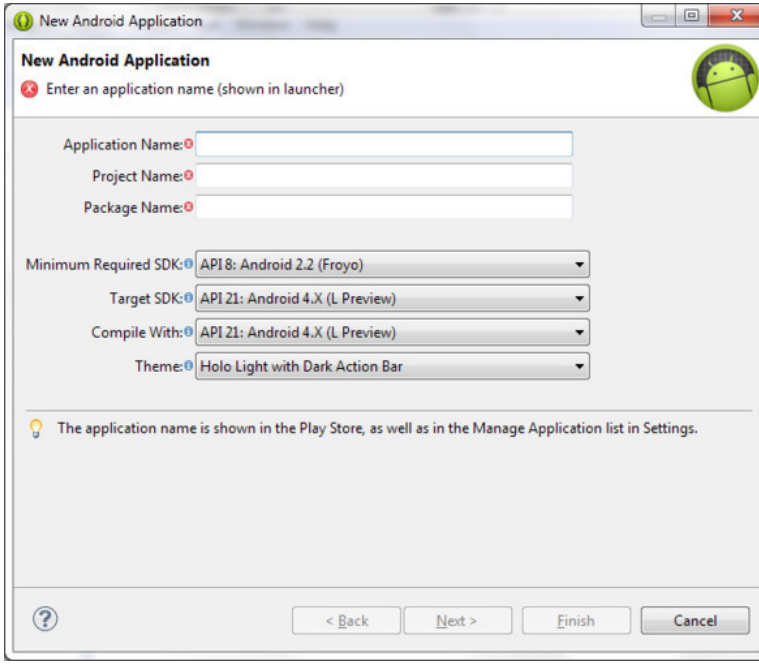
Существует еще один способ тестирование приложений: прямо на устройстве. Для этого заходим в Run -> Run Configuration. Нажимаем два раза на Android Application, заходим во вкладку Target, далее на предложены три варианта, выбираем последний и нажимаем *Run*. Среда попробует за-

пустить проект, отказываемся.



3 Создание проекта

Заходим во вкладку File – new, выбираем Android Application.

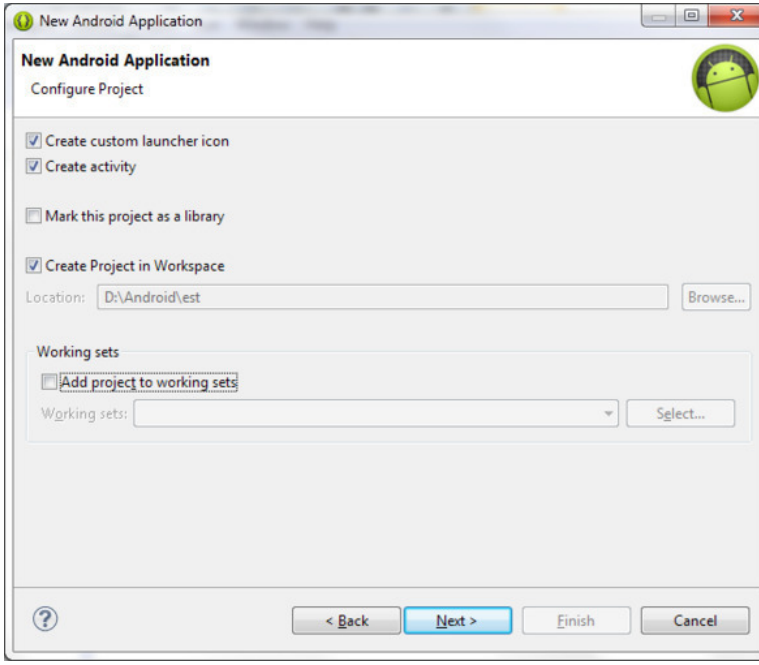


Появляется конструктор для создания проекта. Application Name обозначает имя приложения, оно будет

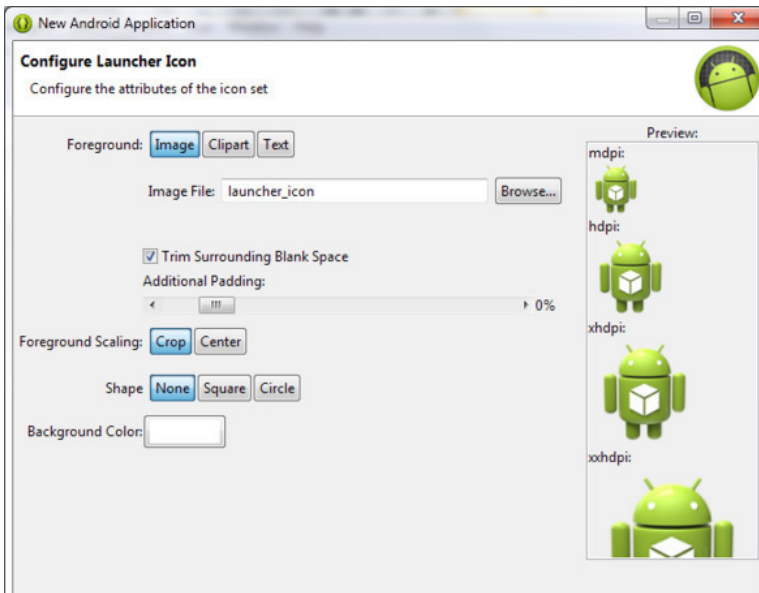
отображаться для пользования, его можно поменять потом. Project Name обозначает проекта, это свойство необходимо для разработчика. Package Name обозначает имя для пакета, где будет хранится фалы с исполняемым кодом.

Minimum SDK устанавливает минимальную версию API для вашего приложения. Theme устанавливает тему для внешнего вида приложения. Все эти данные можно потом поменять.

Нажимаем Next, в следующей вкладке для создания проекта нас спрашивают: необходимо ли создавать иконку, необходимо ли создавать *активность*, где будет располагаться проект и другие параметры, ничего не меняя нажимаем Next.

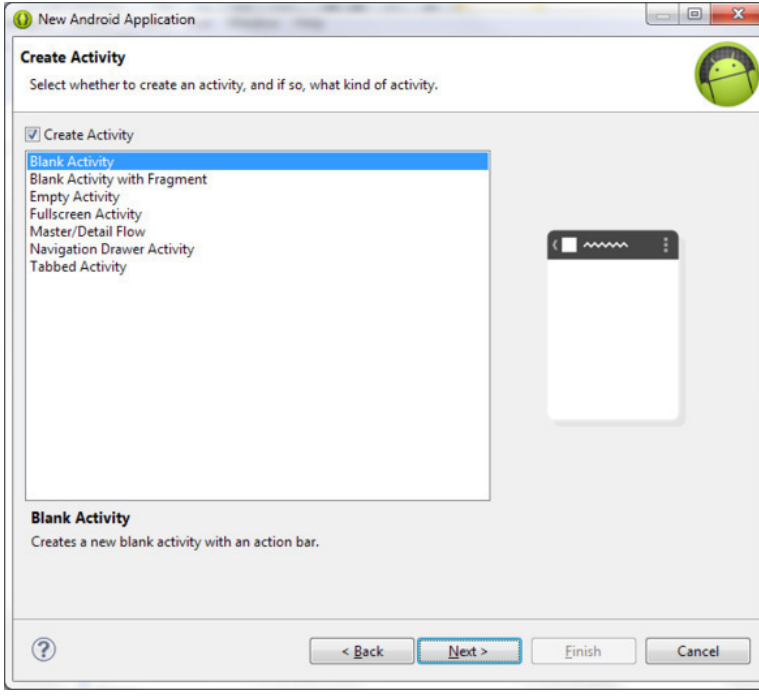


Далее нам предлагают установить иконку, так как это пробное приложение, то предлагаю оставить стандартную иконку, нажав на Next.



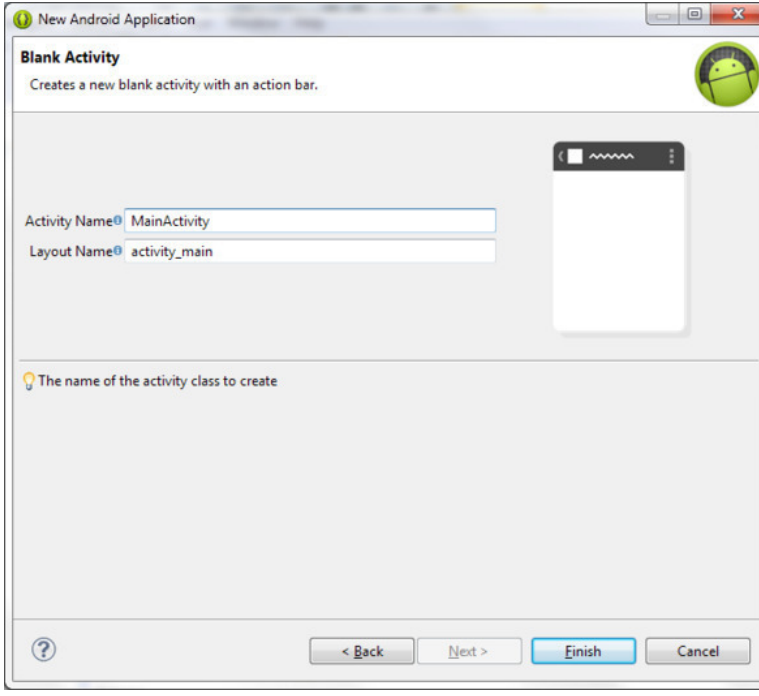
Далее выбираем нам предлагают создать уже некоторые наброски для нашего приложения для облегчения разработки. Если не хотим, то убираем Create Activity. В случае если вы передумаете, то все это можно потом создать, но уже придется писать код собственноручно. По сути говоря выбирая какой-либо из пунктов, среда просто добавляет готовые фрагменты кода в наш проект, тем самым делая за нас небольшую работу, эту же работу мы и сами можем сделать.

Но для учебных целей выбираем Blank Activity и нажимаем Next.



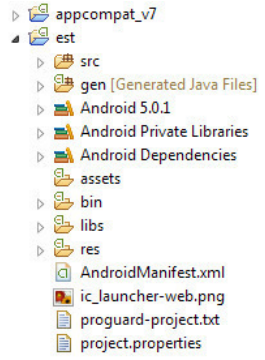
Далее нам предлагают установить имена для файлов разметки, активности и ресурсов, на создание которых мы согласились в предыдущем пункте. Рекомендую оставить как есть, но если вас это не устраивает, можно переименовать. Но учитываете, что названия должны быть логично связанны

друг с другом, чтобы самим не потеряться в проекте, ошибкой не будет если названия будут одинаковы, но опять же мы вам это не рекомендую.



По нажатию Finish, проект успешно создастся.

Первое на что стоит обратить внимание это на иерархию папок, расположенной слева.

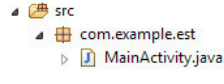


Данная иерархия и есть структура нашего с вами проекта проекта. Конечно программист, увидевший ее впервые растеряется, однако не все так страшно. Дело в том, что папок и файлов, с которыми мы будем работать, из них немного. Конечно же опытным программистам, которые занимаются разработкой под данную систему уже ни один год, могут понадобятся они все. Очень часто такие программисты добавляют свои ресурсы, файлы, папки, библиотеки, разработанные собственноручно. В ходе таких изменений проект сильно меняется. Однако наш уровень пока что не позволит этого сделать. Конечно, мы будем создавать собственные классы, добавлять ресурсы, но в ходе таких изменений структура проекта не поменяется, а инструментов, которые мы будем использовать, будет немного

Давайте разберемся, что мы будем использовать при разработке.

Папка *src* хранит в себе пакеты классов исполняемого кода. Это могут быть и классы активностей и простые классы Java, которые понадобятся нам для выполнения приложения, и многие другие классы.

Изначально создан всего один пакет, имя которого мы присвоили при создании проекта. В нем содержится наша активность, если конечно мы ее создали при создании проекта. Класс новой помещается в этот же пакет.



Далее папка res. В ней содержаться все ресурсы нашего приложения. Это могут быть и картинки, и *layout*-файлы, и файлы анимации, и константные ресурсы и многое другое. Для каждого вида ресурса (например анимации или изображения) создается своя папка. Некоторые ресурсы уже были созданы средой разработки при создании проекта, а некоторые придется создавать нам самим.

Стоит обратить внимание на то, в проекте несколько па-

пок *drawable*, каждая из которых имеет свой уникальный постфикс. В этих папках хранятся изображения, используемые в проекте. Создано это для того, чтобы адаптировать изображения под разные разрешения или размеры экрана. Однако если вы не собираетесь этого делать, достаточно скопировать изображение в одну из папок *drawable*, качество картинки может ухудшиться. Если же вы хотите, чтобы качество гарантировано не страдало при переносе с одного экрана на другой, то вам придется создать каждую картинку с разными разрешениями, присвоить для каждого файла одно имя и скопировать их в папки соответствующих указанному разрешению.

Далее папка *gen*. В ней хранятся id для наших ресурсов. В этой папке могут хранится несколько пакетов, нам нужен тот, чью имя совпадает с именем пакета проекта. В нем хранится файл (класс) R. java. В нем созданы вложенные классы, которые содержат id ресурсов. Имя вложенного класса совпадает с именем папки ресурса. Именно через id и происходит взаимодействие между ресурсами и исполняемым кодом. То есть в коде через id можно получить наш ресурс и далее уже делать с ним все, что нам необходимо: изменять размеры, менять текст, удалять и т.д.. Хочу отметить, что при создании ресурса и присвоении ему id, если это требуется, в класс R среда автоматически поместит id этого ресурса, однако на практике иногда бывает так, что id не создается, в этом случае нам придется самим его создавать, это совсем

не трудно.

Еще один файл, располагающий в корне проекта, это *AndroidManifest. xml*, он представляет собой файл XML. В него записываются все свойства и параметры нашего приложения и отдельной активности в частности. В нем указано какая активность должна запуститься первой, разрешения на использования некоторых функций устройства (интернет, Bluetooth и т.д.), стиль нашего приложения, его имя и другие параметры.

4 Ресурсы

Так как ресурсов в Android много, хотелось бы рассказать о них подробней.

И итак Android выделяет несколько видов ресурсов. Из них: изображения, layout-файлы, анимации, меню, строковые ресурсы и другие. Рассмотрим лишь те, которые мы будем использовать на начальном этапе.

Первый и самый главный ресурс это **layout-файл**, в дальнейшем буду называть слой.

Слой представляет собой файл xml. Он является элементом пользовательского интерфейса. Создается с помощью языка разметки xml. По сути дела в нем программист определяет и размечает пользовательский интерфейса (размеры составных элементов, их расположение и т.д.).

Изображение. Изображения располагаются в папка *drawable*. В drawable хранятся все изображения, используемые в приложении, в том числе и иконки. Поддерживаемые форматы файлов изображений: PNG (предпочтительный), JPG и GIF.

Стоит обратить внимание на то, в проекте несколько папок *drawable*, каждая из которых имеет свой уникальный постфикс. Создано это для того, чтобы адаптировать изображения под разные разрешения или размеры экрана. Однако если вы не собираетесь этого делать, достаточно ско-

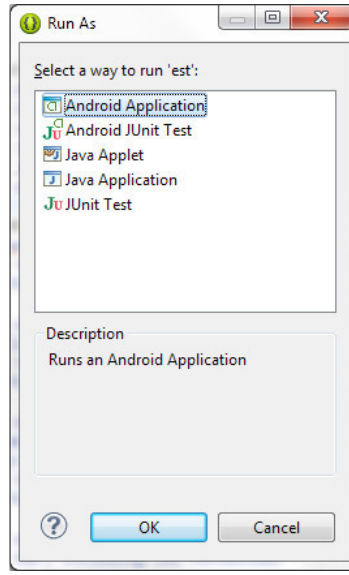
пировать изображение в одну из папок *drawable*, качество картинки может ухудшиться. Если же вы хотите, чтобы качество гарантировано не страдало при переносе с одного экрана на другой, то вам придется создать каждую картинку с разными разрешениями, присвоить для каждого файла одно имя и скопировать их в папки соответствующих указанному разрешению.

Строковые ресурсы – это еще одни вид ресурсов. Создаются они так же с помощью языка XML. Он представляют собой обычные строки, отформатированные специальным образом. Хранятся они в папке *values*.

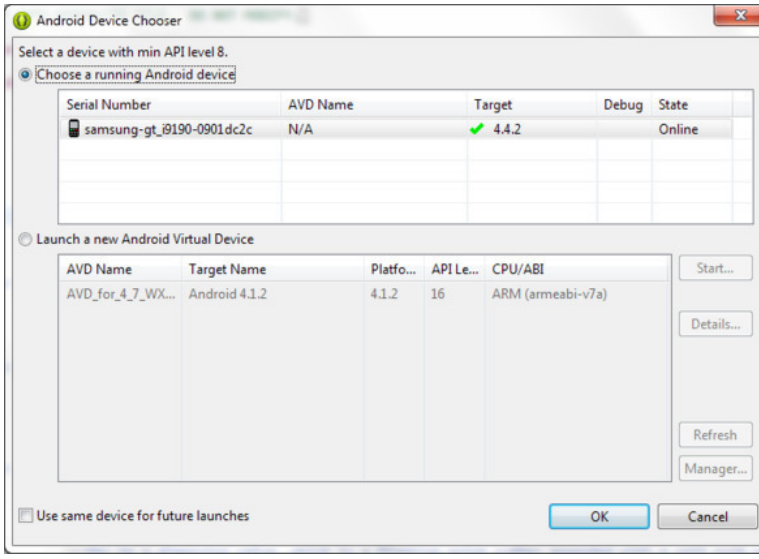
5 Запуск приложения

И так, допустим, что вы довольно долго разрабатывали свое приложение и хотите увидеть результаты. Нам необходимо запустить свое приложение. Для этого выделим его корневую папку и кликнем на кнопку *Run.*

Далее у нас появится следующее окно:



Нажимаем *Ok*. После этого появится окно, где вы можете выбрать на каком устройстве выполнить запуск. В верхнем листе записаны все реальные устройства подключенные к ПК и настроенные должным образом (включена отладка на устройстве), в нижнем листе – все виртуальные устройства, созданные вами. Вы можете выбрать любое и нажать на запуск.



Далее произойдет запуск на устройстве. Если вы выбрали виртуальную машину, то скорее всего вам придется подождать пока она включится, повторные запуске на виртуальных машинах будут производится намного быстрее, поэтому если вам требуется неоднократно запускать проект, не закрывайте окно виртуальной машины. Можно так же запустить один проект на нескольких устройствах, в том числе и на реальных, и виртуальных, для этого достаточно при каждом запуске выбирать разные варианты. Если вы выбрали устройство, на котором уже запущено приложение, то произойдет перезапуск.

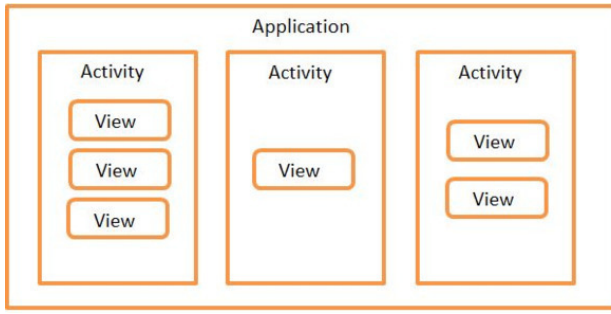
Декларативный способ создания экранных элементов

Структура приложения

Если проводить аналогию с Windows, то приложение состоит из окон, называемых ***Activity***. В конкретный момент времени обычно отображается одно Activity и занимает весь экран, а приложение переключается между ними. В качестве примера можно рассмотреть почтовое приложение. В нем одно Activity – список писем, другое – просмотр письма, третье – настройки ящика. При работе вы перемещаетесь по ним.

Содержимое Activity формируется из различных компонентов, называемых ***View***. Самые распространенные View – это кнопка, поле ввода, чекбокс и т. д.

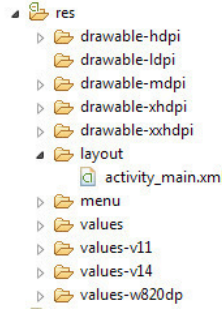
Примерно это можно изобразить так:



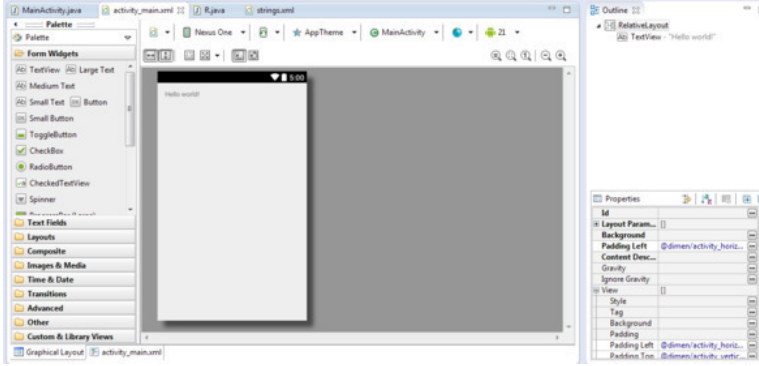
Необходимо заметить, что View обычно размещаются в ***ViewGroup***. Самый распространенный пример ViewGroup – это ***Layout***. Layout бывает различных типов и отвечает за то, как будут расположены его дочерние View на экране (таблицей, строкой, столбцом и т.д.).

ViewGroup можно рассматривать как контейнер View элементов (дочерних), в этом контейнере элементы хранятся согласно поведению и структуре, который присущ данной ViewGroup. ViewGroup можно назвать родительским элементом, для View, которые хранятся в нем, а сами View – дочерними для этого ViewGroup.

И так нам интересен layout-файл, он располагается в папке res -> layout.



Открываем его и видим следующее:



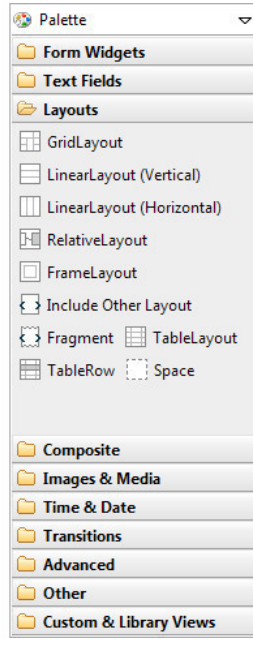
Это макет нашего экрана, именно это увидит пользователь когда приложение запустит данный layout-файл. А так как этот слой отобразится первым, то соответственно это первое что увидит пользователь в вашем приложении. Стоит отме-

тить что порядок отображения слоев должно регулироваться исполняемым кодом. То есть именно программист при создании приложения должен определять когда и при каких обстоятельствах отобразится данный слой.

Слева мы видим набор View-элементов укомплектованные в иерархии. Если мы перетащим какой-нибудь элемент на наш слой, то он автоматически добавится в наш layout-файл.

Справа в верхнем углу видим логическую структуру нашего слоя, давайте разберемся с этим подробнее. И так наш layout-файл имеет корневой слой RelativeLaoyt (один из видов слоев), в данном случаем слой является элементом ViewGroup. Стоит понимать разницу между layout-файлом (файлом разметки) и layout (слоем). Первое это наш экран, вторым можно назвать View-элемент, хотя это не совсем так, как было сказано layout (слой) – это элемент ViewGroup. Любой layout-файл имеет свой корневой слой, в этом слое хранятся дочерние элементы, согласно правилам этого слоя. При создании проекта у нас уже добавился автоматически дочерний View – TextView с надписью: «Hello world».

Как вы уже, наверное, в наборе View, расположенной слева, есть иерархия Layouts.

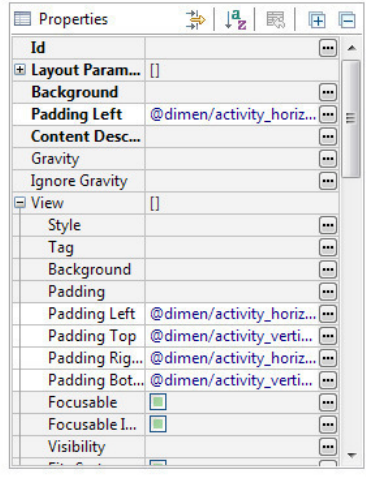


В этой иерархии хранятся все виды слоев. Вы можете перетащить любой из них на ваш корневой layout. В этом случае корневой слой будет считаться родительским, а слой, который мы добавили – дочерний. Дочерний слой будет принимать те правила существования, которые присущи его родительскому слою. В то же время, если мы добавим в созданный слой какой-нибудь элемент (другой слой или просто элемент View), то этот элемент будет уже дочерним и будет принимать правила уже созданного поверх корневого слоя,

а не самого корневого слоя. То есть суть в том, что дочерний элемент существует согласно поведению, которое диктует родительский.

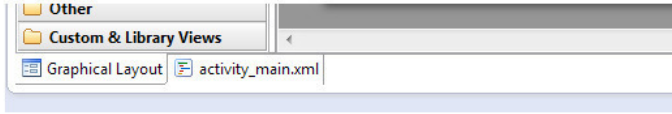
Еще хотел бы отметить несколько моментов интерфейса создания layout-файлов.

Во-первых: это окно свойств View-элемента. Для начала выделим элемент, свойства которого мы хотим поменять, и в правом нижним углу появится панель.

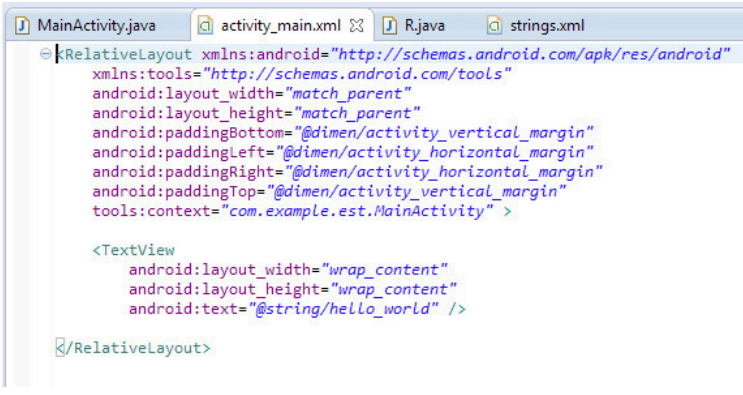


Собственно в нем все довольно понятно. Вы можете поменять расположение элемента, его размеры, фон, текст, отступ и другие параметры.

Во-вторых: это текстовый способ заполнения layout-файла. В низу расположена панель:



Легко догадаться, что сейчас мы создаем в Graphical Layout (то есть путем перетаскивания элементов на наш экран). Существует еще один способ, для этого переключим на activity_main. xml (если ваш layout-файл называется по-другому, то будем название вашего файла). Мы видим совершенно, на первый взгляд непонятый код.



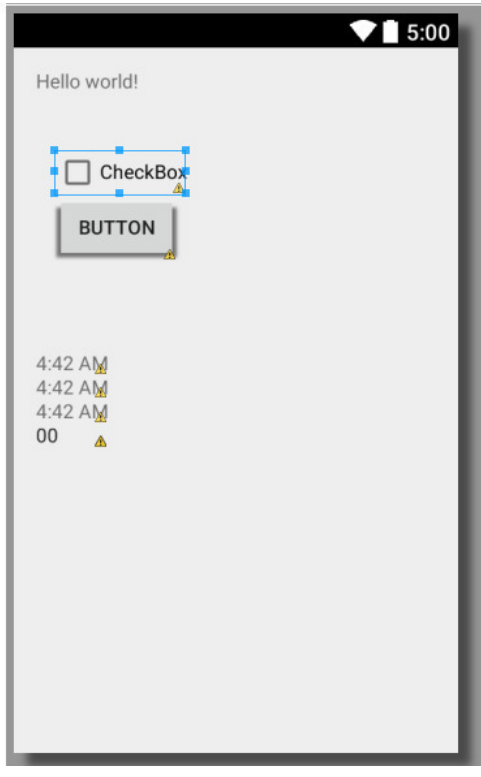
Собственно это и есть наш файл, просто в другом, непривычном для нас, представлении. Заполнение выполняется на языке XML. В корне мы видим наш корневой слой и его свойства, далее созданный элемент в этом слое и его свойства, параметры в том и другом способе совпадают.

В этом и заключается декларативный способ создания экрана. Декларативный способ описывает конечный результат, который должна сгенерировать программа. Затем наш слой закрывается и любые элементы, за этими границами будут содержаться в другом слое или элементе ViewGroup. Если создать элемент за рамками корневого слоя, то среда сгенерирует ошибку.

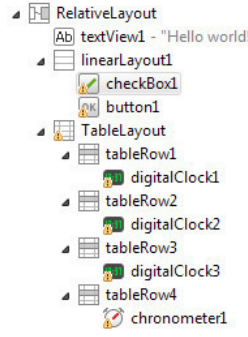
На самом деле если вы найдем папку, где хранится наш layout-файл, и откроем его простым текстовым редактором, то увидим в нем точно такой же текст. То есть изначально

layout имеет такое представление, а затем наша среда на основе этого текста создает для нас макет программы. Хочу отметить, что если вы создадите элементы или поменяйте свойства какого либо элемента одним способом, то изменения и коснутся в рамках другого способа.

И так приступим к практике, чтобы все эти технологии стали более понятными. Перекинем в наш слой какие-нибудь элементы.



После небольших экспериментов мы получили такой результат, этого мало что понятно что мы создал. Чтобы понять, необходимо посмотреть на иерархию элементов, расположенную слева.



И так самый главный у нас корневой слой RelativeLayout. В нем текстовый элемент, который среда создала автоматически. Далее мы добавил еще один слой LinearLayout, он обладает такими свойства, что элементы расположены по вертикале (существует такой же только элементы будут располагаться горизонтально), в него мы добавил пару элементов. Потом был добавлен слой TableLayout поверх корневого слоя, в нем дочерние элементы являются слои TableRow (элементы в нем расположены по горизонтали и растянуты по всей ширине). В каждый TableRow мы добавил еще по одному элементу.

Конец ознакомительного фрагмента.