РАЗРАБОТКА МОБИЛЬНЫХ ПРИЛОЖЕНИЙ

Учебно-методическое пособие

В данном учебно-методическом пособии представлены базовые темы для начала разработки приложений под мобильные устройства на платформе Android. Рассматриваются принципы создания пользовательского интерфейса, работы с данными, файловой системой, активностями, интентами, фрагментами, навигационными элементами и анимациями. Описаны базовые принципы построения архитектуры приложений.

ВВЕДЕНИЕ

Android – операционная система для смартфонов, планшетов, электронных книг, цифровых проигрывателей, наручных часов, фитнесбраслетов, игровых приставок, ноутбуков, очков Google Glass, телевизоров и других устройств [1]. Она основана на ядре Linux и собственной реализации виртуальной машины Java от Google. Изначально разрабатывалась компанией Android Inc. (теперь Google).

Каждая версия системы начиная с версии 1.5 получает собственное кодовое имя на тему сладостей. Кодовые имена присваиваются в порядке латинского алфавита (рисунок) [1]. На настоящий момент выпущено 15 версий системы. Исходя из статистики на май 2019 г. [1].

Как показано на рисунке, самой распространенной версией остается Android 6.0, она установлена на 16,9% всех устройств. Следом за ней расположилась система Android 8.1 с 15,4%.

Версия	Название	Год	Доля
2.3	*Gingerbread*	2010	0,3 %
4.0	*Ice Cream Sandwich*	2011	0,3 %
4.1	*Jelly Bean*	2012	1,2 %
4.2		2012	1,5 %
4.3		2013	0,5 %
4.4	*KitKat*	2013	6,9 %
5.0	*Lollipop*	2014	3 %
5.1		2015	11,5 %
6.0	*Marshmallow*	2015	16,9 %
7.0	*Nougat*	2016	11,4 %
7.1		2016	7,8 %
8.0	*Oreo*	2017	12,9 %
8.1		2017	15,4 %
9.0	*Pie*	2018	10,4 %
10.0	*Q*	2019	< 0,1 %

Версии операционной системы Android

Для разработки приложений под операционную систему Android нужно скачать и установить SDK. Сейчас существует несколько сред разработки:

– NetBeans;

– Eclipse; –
Intellij IDEA;
– Android Studio.

Android Studio ориентирована именно под ОС Android, а также не требует установки дополнительных плагинов. Примеры выполнения заданий в данном пособии будут рассматриваться на Android Studio.

Рассмотрим языки разработки нативных приложений.

Java – официальный язык программирования, поддерживаемый средой разработки Android Studio. На Java ссылается большинство официальной документации Google.

Kotlin – язык был официально представлен в мае 2017 г. на Google I/O и позиционируется Google как второй официальный язык программирования под Android после Java. Kotlin совместим с Java и не вызывает снижения производительности и увеличения размера файлов. Отличие от Java в том, что он требует меньше служебного кода, поэтому более легкий для чтения.

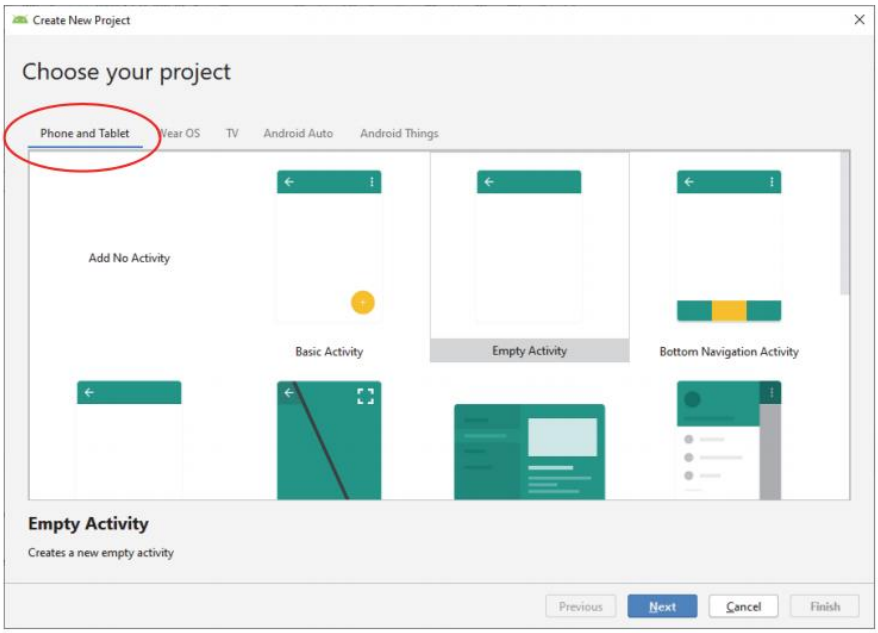
Более низкоуровневые языки поддерживаются Android Studio с использованием Java NDK (Native Development Kit). Это позволяет писать нативные приложения, что может пригодиться для создания игр или других ресурсоемких программ.

Примеры, приведенные в данном учебно-методическом пособии, написаны на языке Java.

1. СОЗДАНИЕ ПРИЛОЖЕНИЙ НА ПЛАТФОРМЕ ANDROID

1.1. Создание первого Android-приложения

Для создания нового проекта в Android Studio (рис. 1.1) необходимо выбрать шабон приложения.



Затем при задании параметра *Package name* (рис. 1.2) необходимо использовать обратное доменное имя, например *by.bstu.fit.фио*.

Следующий шаг – установка версии API (рис. 1.2). При выборе более нового API предоставляется небольшой процент поддерживаемых устройств.

1.1.1. Компоненты приложения

Компоненты приложения являются блоками, из которых состоит приложение. Каждый компонент представляет собой отдельную точку,

через которую система может войти в приложение. Не все компоненты являются точками входа для пользователя. Однако каждый компонент – это самостоятельная структурная единица, которая определяет работу приложения в целом. Компоненты приложения можно отнести к одному из пяти типов.

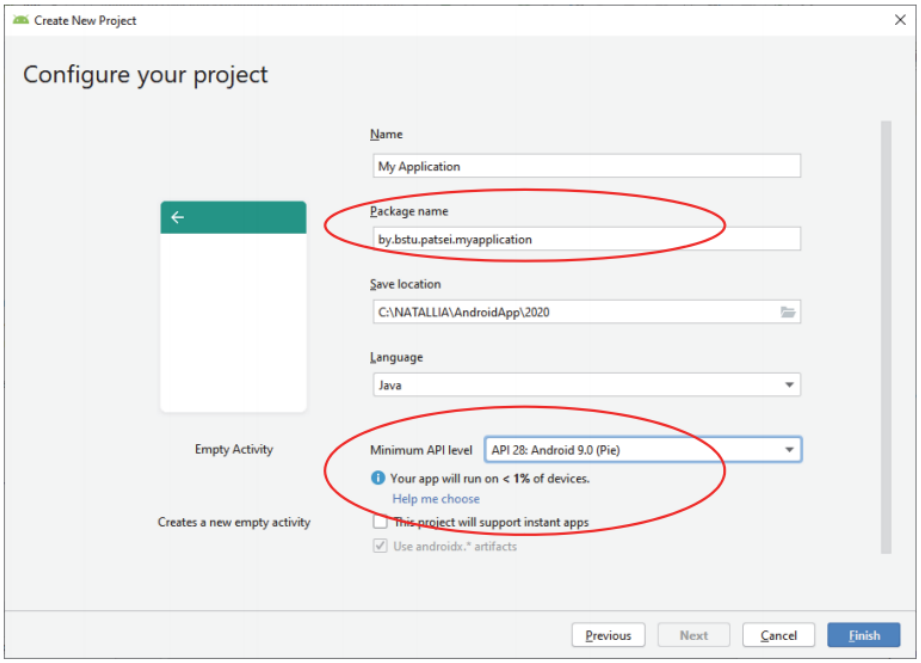


Рис. 1.2. Окно создания проекта Android-приложения

Операция (*Activity*, активность) представляет собой один экран с пользовательским интерфейсом. Например, в приложении для работы с электронной почтой одна активность может служить для отображения списка новых сообщений, другая – для составления сообщения, а третья – для чтения сообщений. Несмотря на то что активности совместно формируют взаимодействие пользователя с приложением, каждая из них не зависит от других. Любые из активностей могут быть запущены другим приложением. Например, приложение для камеры может запустить операцию в приложении по работе с электронной почтой, которая составляет новое сообщение, чтобы пользователь мог отослать фотографию. Операция относится к подклассу *Activity* [2].

Служба (*Service*, сервис) представляет собой компонент, который работает в фоновом режиме и выполняет длительные операции, связанные с работой удаленных процессов [2]. Служба не имеет пользовательского интерфейса. Например, она может воспроизводить музыку в фоновом режиме, пока пользователь работает в другом приложении, или может получать данные по сети, не блокируя взаимодействие пользователя с активностью. Служба может

быть запущена другим компонентом, который затем будет взаимодействовать с ней. Служба относится к подклассу *Service*.

Поставщик контента (*Content provider*) управляет общим набором данных приложения. Данные можно хранить в файловой системе, базе данных SQLite, в интернете или любом другом месте хранения, к которому у приложения есть доступ. Посредством поставщика контента другие приложения могут запрашивать или изменять данные (если поставщик контента позволяет делать это). Например, в системе Android есть поставщик контента, который управляет информацией контактов пользователя. Любое приложение, получившее соответствующие разрешения, может запросить часть этого поставщика контента для чтения и записи. Он относится к подклассу *ContentProvider* и должен реализовывать стандартный набор API-интерфейсов [2].

Приемник широковещательных сообщений (*Broadcast receiver*) представляет собой компонент, который реагирует на объявления, распространяемые по всей системе. Многие из этих объявлений рассылает система, например уведомления о том, что экран выключился, аккумулятор разряжен или был сделан фотоснимок. Объявления также могут рассылаться приложениями. Например, можно сообщить другим приложениям о том, что какие-то данные были загружены на устройство и теперь готовы для использования. Несмотря на то что приемники широковещательных сообщений не имеют пользовательского интерфейса, они могут создавать уведомления в строке состояния, чтобы предупредить пользователя о событии «рассылка объявления» [2]. Чаще всего они являются просто шлюзом для других компонентов и предназначены для выполнения минимального объема работы. Приемник широковещательных сообщений относится к подклассу *BroadcastReceiver*.

Уникальной особенностью системы Android является то, что любое приложение может запустить компонент другого приложения. Для пользователя это будет выглядеть как одно приложение. Когда система запускает компонент, она запускает процесс для этого приложения (если он еще не был запущен) и создает экземпляры классов, которые требуются этому компоненту. Поэтому в отличие от приложений для большинства других систем в приложениях для Android отсутствует единая точка входа (в них нет функции *main*()).

Каждое приложение выполняется системой в отдельном процессе с такими правами доступа к файлам, которые ограничивают доступ другим приложениям. Поэтому одно приложение не может напрямую вызвать компонент из другого приложения. Но это может сделать система Android. Для

того чтобы вызвать компонент в другом приложении, необходимо сообщить системе о своем **намерении** (*Intent*). После чего система активирует этот компонент.

1.1.2. Структура проекта

Вернемся к созданному проекту (рис. 1.3). В его структуре имеется единственный модуль – модуль *app*. Код располагается внутри этого модуля.

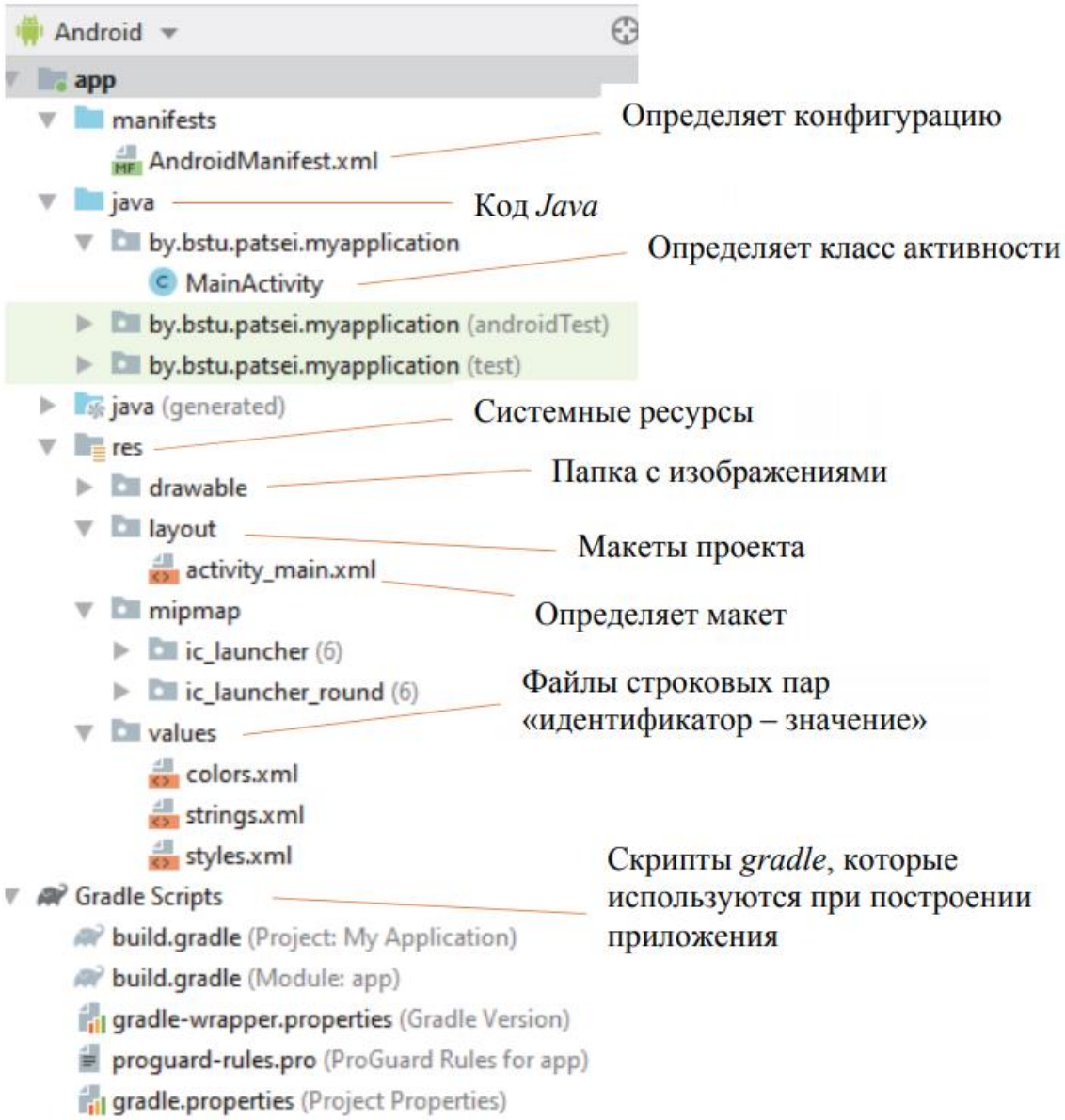


Рис. 1.3. Структура проекта Android-приложения

Все модули в проекте описываются файлом *setting.gradle*. По умолчанию он имеет следующее содержимое:

```
include ':app'
```

Каждый модуль имеет свой файл *build.gradle*, который определяет конфигурацию построения проекта. На начальном этапе данные файлы не столь важны, достаточно лишь понимать, для чего они нужны.

В *app* есть несколько папок и файлов.

Каталоги *androidTest* и *test* предназначены для хранения файлов тестов приложения, а каталог *java* – для хранения исходного кода. Класс *MainActivity* имеет следующую структуру:

```
package by.bstu.patsei.myapplication;
 import androidx.appcompat.app.AppCompatActivity; import
android.os.Bundle;
 public class MainActivity extends AppCompatActivity {


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

Файл *AndroidManifest.xml* представляет собой файл манифеста, который описывает фундаментальные характеристики приложения, его конфигурацию и определяет каждый из компонентов данного приложения.

Для запуска компонента системе Android необходимо знать, что компонент существует. Для этого она читает файл *AndroidManifest.xml* приложения, который должен находиться в его корневой папке. Ниже приведен пример файла манифеста:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="by.bstu.patsei.myapplication">


  <application
      android:allowBackup="true"       android:icon="@mipmap/ic_launcher"
android:label="@string/app_name"
android:roundIcon="@mipmap/ic_launcher_round"
android:supportsRtl="true"       android:theme="@style/AppTheme">
     <activity android:name=".MainActivity">
        <intent-filter>
          <action android:name="android.intent.action.MAIN" />
          <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
      </activity>
  </application>
```

```
</manifest>
```

Здесь *package* объявляет имя Java-пакета данного приложения, которое служит уникальным идентификатором приложения. Атрибут *android:name* в элементе <*activity*> указывает полное имя класса подкласса *Activity*, а атрибут *android:label* указывает строку, которую необходимо использовать в качестве метки активности, отображаемой для пользователя. Компоненты приложения описываются через атрибуты *activity*, *service*, *receiver* и *provider*. Эти объявления позволяют операционной системе узнать, чем компоненты являются и при каких условиях они могут быть запущены. Все компоненты приложения необходимо объявлять следующим образом:

- элементы <*activity*> – для операций;
- элементы <*service*> – для служб;
- элементы <*receiver*> – для приемников широковещательных сообщений;
- элементы <*provider*> – для поставщиков контента.

Системе не видны активности, службы и поставщики контента, которые имеются в исходном коде, но не объявлены в манифесте, поэтому они не могут быть запущены.

Манифест помимо объявления компонентов приложения служит и для других целей. Например, указание всех полномочий пользователя, которые требуются приложению (разрешения на доступ в интернет или на чтение контактов пользователя), для объявления минимального уровня API, требуемого приложению с учетом того, какие API-интерфейсы оно использует, для объявления аппаратных и программных функций, которые нужны приложению или используются им, например функции камеры, службы Bluetooth или сенсорного экрана, для указания библиотек API, с которыми необходимо связать приложение, например библиотеки Google Maps и др.

Папка *res* содержит каталоги с ресурсами (см. рис. 1.3):

- папка *drawable* предназначена для хранения изображений, используемых в приложении;
- папка *layout* предназначена для хранения файлов, определяющих графический интерфейс. По умолчанию здесь есть файл *activity_main.xml*, который определяет интерфейс для единственной в проекте активности – *MainActivity*;
- папки *mipmap*-xxxx содержат файлы изображений, которые предназначены для создания иконки приложения при различных разрешениях экрана. Соответственно для каждого вида разрешения здесь имеется свой каталог;
- папка *values* хранит различные XML-файлы, содержащие коллекции ресурсов.

Используя ресурсы приложения, можно без труда изменять его характеристики, не меняя код, и, кроме того, путем предоставления наборов альтернативных ресурсов можно оптимизировать приложение для работы с различными конфигурациями устройств (например, для различных языков или размеров экрана).

Для каждого ресурса, включаемого в проект Android, инструменты SDK задают уникальный целочисленный идентификатор, который может использоваться, чтобы сослаться на ресурс из кода приложения или из других ресурсов, определенных в XML. Например, если в приложении имеется файл изображения с именем *logo.png* (сохраненный в папке *res/drawable/*), инструменты SDK сформируют идентификатор ресурса под именем *R.drawable.logo*, с помощью которого на изображение можно будет ссылаться и вставлять его в пользовательский интерфейс.

Один из наиболее важных аспектов предоставления ресурсов отдельно от исходного кода заключается в возможности использовать альтернативные ресурсы для различных конфигураций устройств. Например, после определения строк пользовательского интерфейса в XML, можно перевести их на другие языки и сохранить эти переводы в отдельных файлах. Затем по квалификатору языка, добавленному к имени каталога ресурса, и выбранному пользователем языку система Android применит к вашему пользовательскому интерфейсу строки на соответствующем языке.

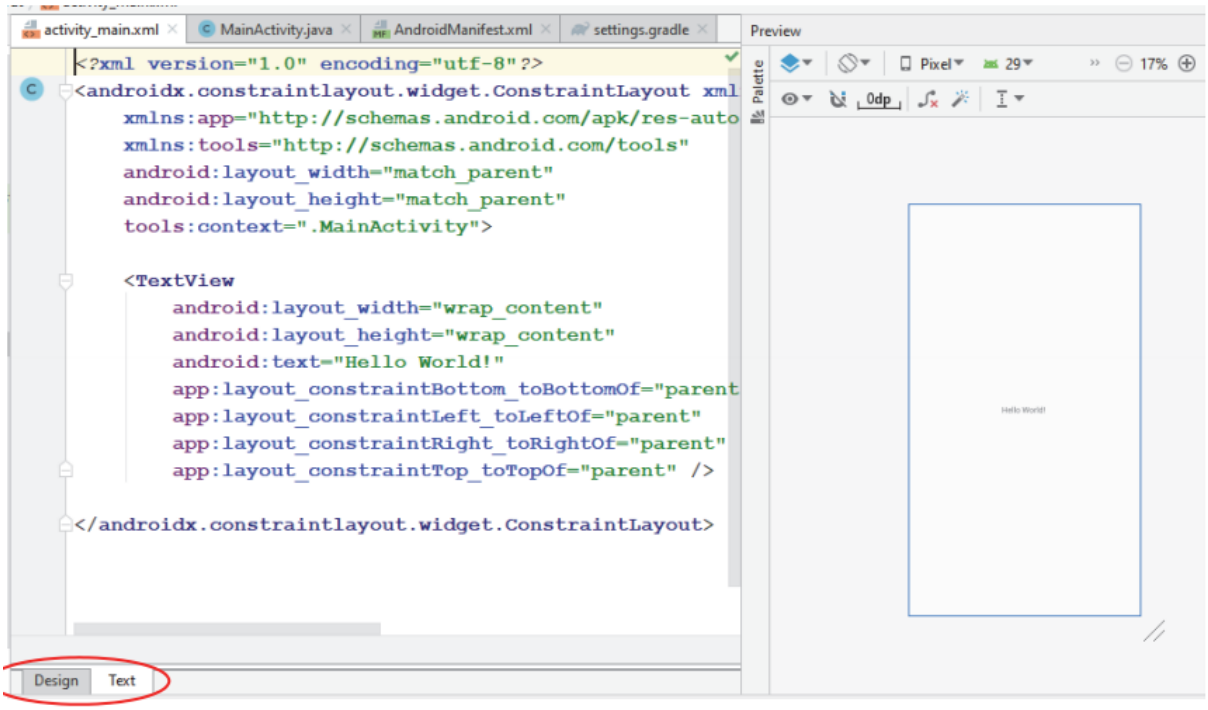
Android поддерживает разные квалификаторы для ресурсов. Квалификатор представляет собой короткую строку, которая включается в имена каталогов ресурсов с целью определения конфигурации устройства, для которой эти ресурсы следует использовать. Например, когда экран устройства имеет книжную ориентацию (расположен вертикально), кнопки в макете можно также размещать по вертикали, а когда экран развернут горизонтально (альбомная ориентация), кнопки следует размещать по горизонтали. Чтобы при изменении ориентации экрана изменялся макет, можно определить два разных макета и применить соответствующий квалификатор к имени каталога каждого макета. После этого система будет автоматически применять соответствующий макет в зависимости от ориентации устройства.

1.1.3. Разработка интерфейса приложения

В студии должен быть открыт файл *activity_main.xml*, который содержит определение графического интерфейса приложения.

Если файл открыт в режиме дизайнера, а в центре отображает- ся дизайн приложения, то надо переключить вид файла в текстовый. Для переключения

режима из текстового в графический и обратно внизу есть две кнопки *Design* и *Text* (рис. 1.4).



Изменим код приложения так, чтобы оно выводило на экран строку «Hello World!». В файле *activity_main.xml* определение элемента *TextView* отвечает за вывод текстовой информации на экран. Выводимый текст задается с помощью атрибута *android:text*.

После сохранения файла можно переключиться в графический режим (см. рис. 1.4), где можно устанавливать режимы *Design* – в нем *view*-компоненты выглядят так, как обычно; *Blueprint* – отображаются только контуры *view*-компонентов; *Design* + *Blueprint* – два экрана одновременно. Кнопки на панели (рис. 1.5) позволяют переключать режимы *Design*, *Blueprint*, *Design* + *Blueprint*.

Окно *Палитра* (*Palette*) представляет собой список всех *view*компонентов, которые можно добавлять на экран: кнопки, поля ввода, чекбоксы, прогрессбары и т. д.

Окно *Дерево компонентов* (*Component Tree*) отображает иерархию *view*-компонентов. Например, на рис. 1.5 корневой элемент – это *ConstraintLayout*, а в него вложен *TextView*.

В окне *Свойства* (*Properties*) при работе с *view*-компонентом будут отображаться его свойства (рис. 1.6).

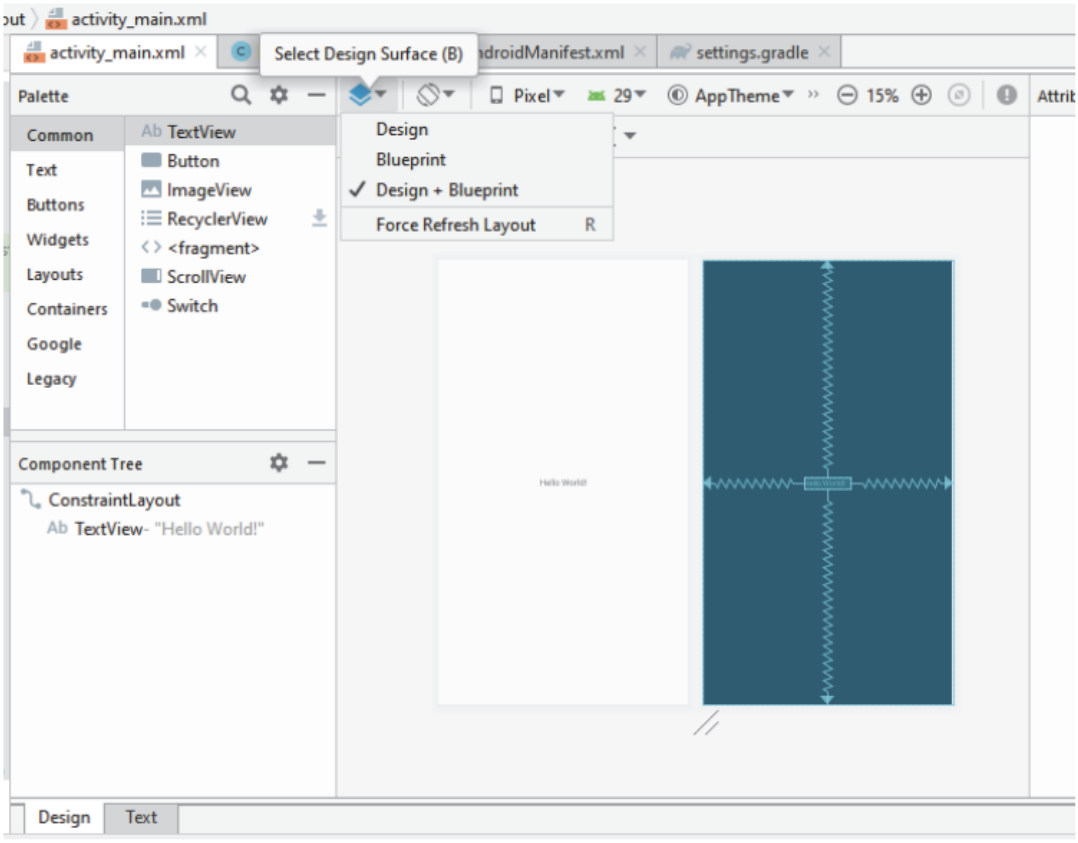


Рис. 1.5. Структура окон разработки и редактирования экрана приложения

Activity является классом, который по сути представляет отдельный экран (страницу) приложения или его визуальный интерфейс. Приложение может иметь одну операцию, а может и несколько. Каждая отдельная активность задает окно для отображения.

Рассмотрим код активности, которая генерируется автоматически в Android Studio (файл кода можно найти в проекте в папке *src/main/java*):

```
package by.bstu.patsei.myapplication; import
androidx.appcompat.app.AppCompatActivity; import
android.os.Bundle; public class MainActivity extends
AppCompatActivity {

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

Класс *MainActivity* представляет обычный Java-класс, в начале которого идут определения пакета и импорта внешних пакетов. По умолчанию он содержит только один метод *onCreate*(), в котором фактически и создается весь интерфейс приложения.

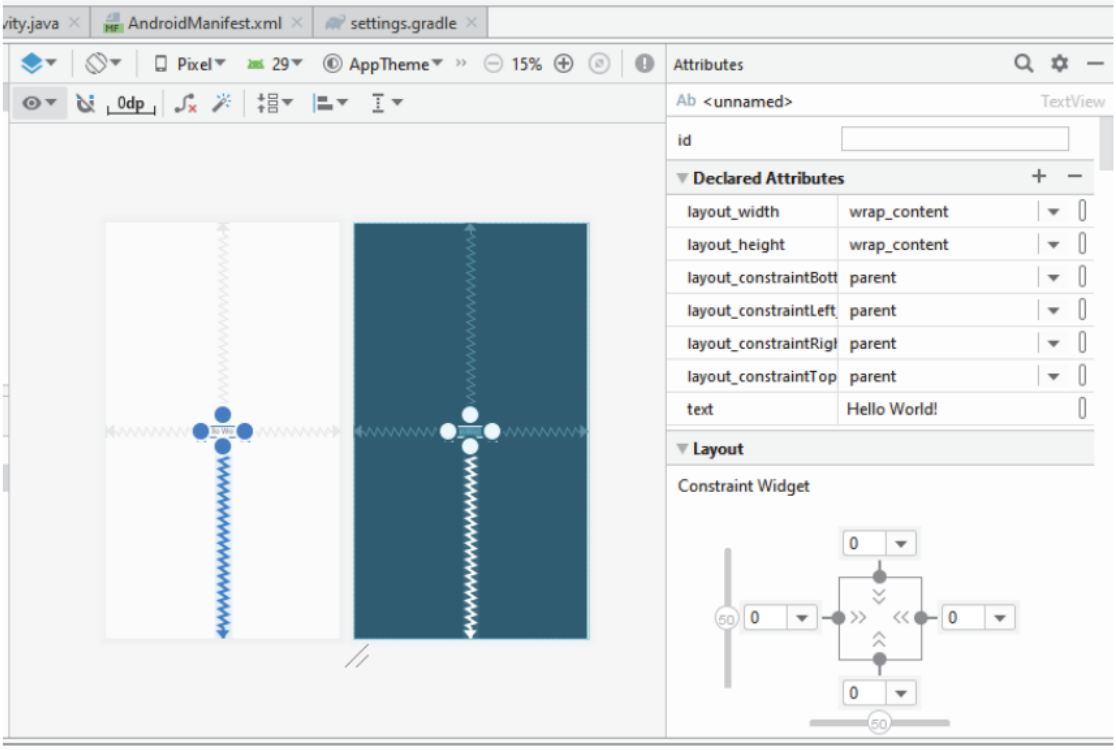


Рис. 1.6. Окно свойств

В методе *OnCreate*() идет обращение к методу родительского класса и установка ресурса разметки дизайна:

```
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
```

Чтобы установить ресурс разметки дизайна, вызывается метод *setContentView*, в который передается идентификатор ресурса. Идентификатор ресурса выглядит следуюшим образом: *R.layout.activity_main*. Это и есть ссылка на файл *activity_main.xml*, который находится в каталоге *res/layout*. Таким образом, при запуске приложения сначала запускается класс *MainActivity*, который в качестве графического интерфейса устанавливает разметку из файла *activity_main.xml*. Однако в классе *MainActivity* используются не файлы, а идентификаторы ресурсов: *R.layout.activity_main*.

Все идентификаторы ресурсов определены в классе *R*, который автоматически создается утилитой *appt* и находится в файле *R.java* в каталоге

build/r_class_sources/debug/r/... (рис. 1.7). Класс *R* содержит идентификаторы для всех ресурсов. Для каждого типа ресурсов в классе *R* создается внутренний класс (например, для всех графических ресурсов из каталога *res/drawable* создается класс *R.drawable*) и каждому ресурсу присваивается идентификатор (рис. 1.7).

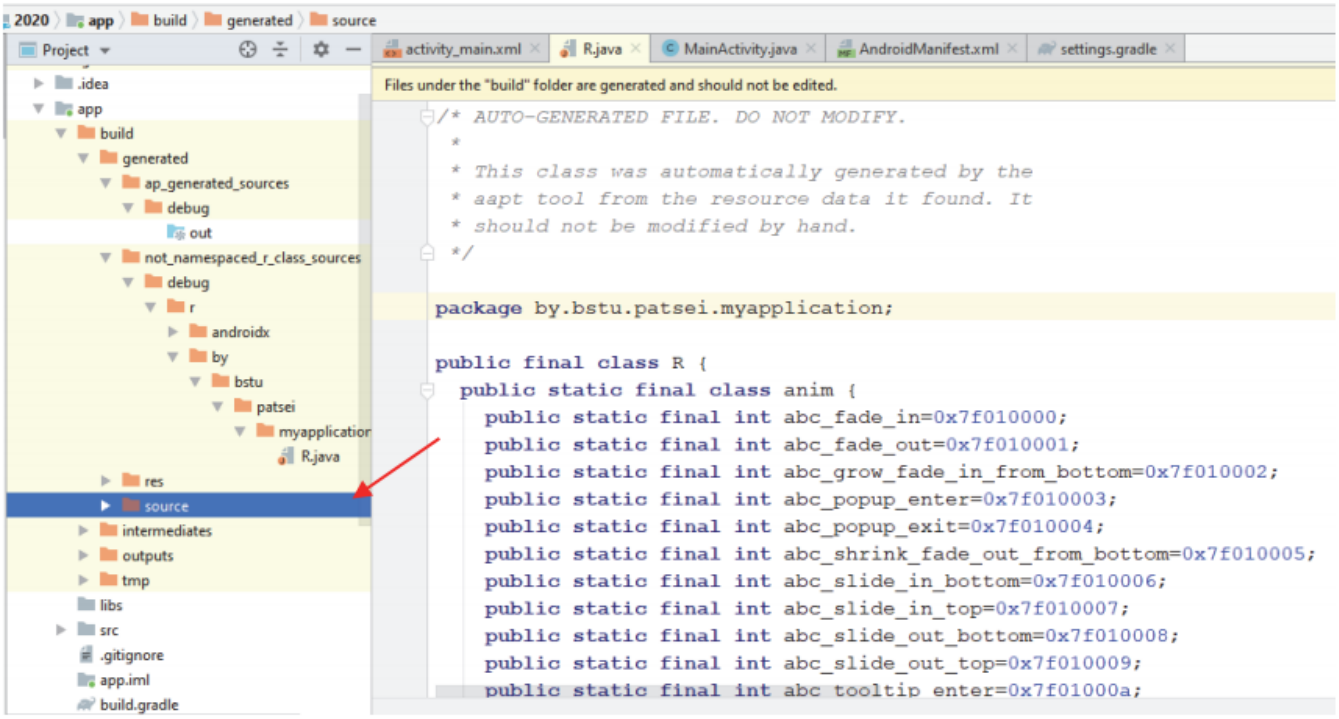


Рис. 1.7. Содержимое и физическое размещение файла ресурсов

По этому идентификатору впоследствии можно извлечь ресурс в файле кода. При обновлении ресурсов во время компиляции этот файл также обновляется.

1.1.4. Запуск приложения

Запускать приложение можно на реальном Android-устройстве или эмуляторе. При этом на эмуляторе можно выбрать одно из установленных виртуальных устройств, создать новый эмулятор (рис. 1.8) в Android Studio или использовать сторонние эмуляторы: Genymotion, Droid4x, BlueStacks и др.

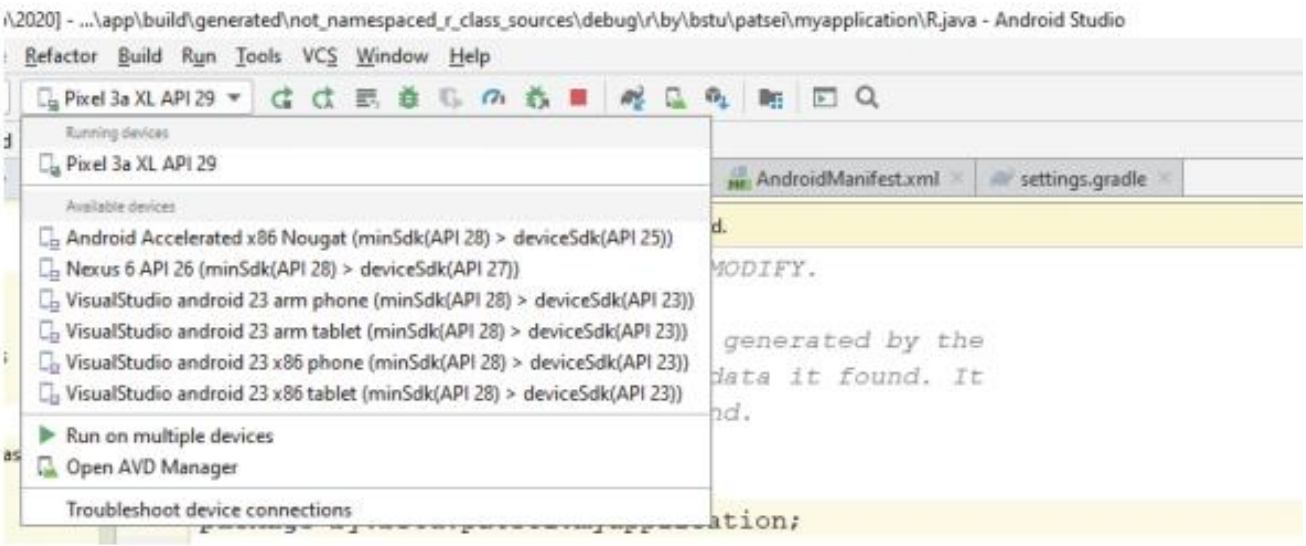


Рис. 1.8. Выбор места установки приложения

При использовании мобильного устройства для размещения приложения необходимо установить опции разработчика на мобильном устройстве. Для этого следует найти *Settings* → *About phone* (*Настройки* → → *О телефоне*) и семь раз нажать *Build Number* (*Номер сборки*). Вернуться к предыдущему экрану, на котором будет доступный пункт *Developer options* (*Для разработчика*). Перейти к пункту *Для разработчиков* и включить возможность отладки по USB. Через SDK Manager установить пакет Google Usb Driver или другой драйвер. После чего окно установки приложения отобразит подключенное устройство (рис. 1.9).

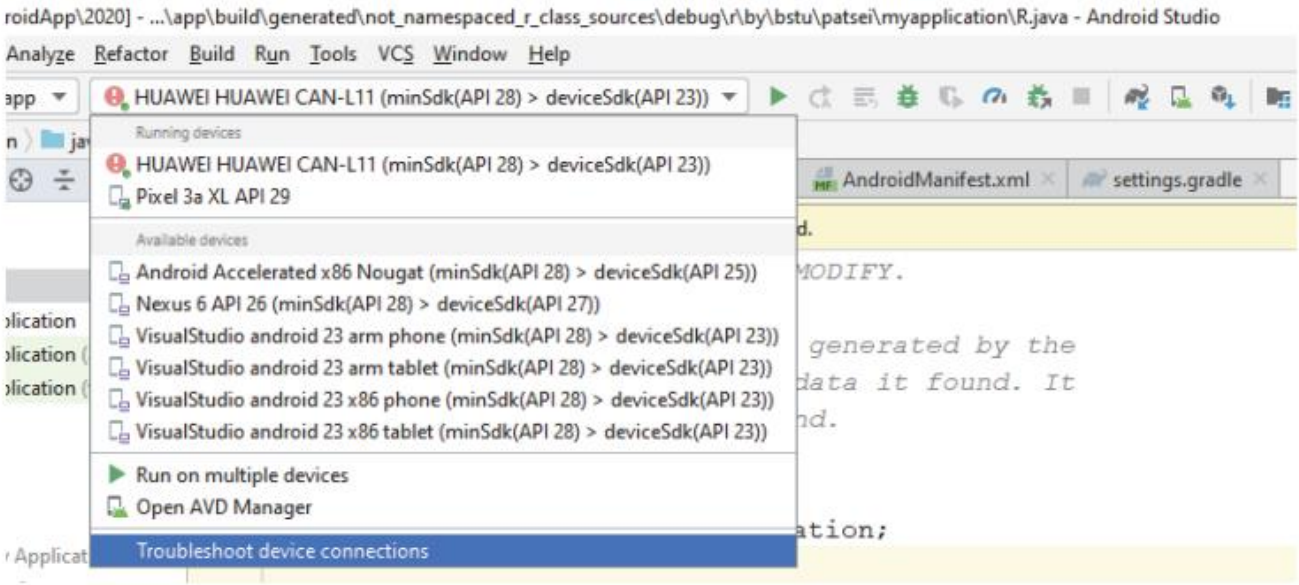


Рис. 1.9. Выбор места установки приложения

Результат запуска рассмотренного приложения представлен на рис. 1.10.

Рис. 1.10. Результат запуска приложения

1.2. Архитектура Android и процесс компиляции мобильного приложения

1.2.1. Архитектура операционной системы Android

Основой платформы Android является ***ядро Linux***. Android Runtime (ART) полагается на ядро Linux для базовых функций, таких как потоковая обработка и управление памятью на низком уровне. Ядро Linux позволяет Android использовать ключевые функции безопасности и производителям устройств разрабатывать аппаратные драйверы (рис. 1.11).

Рис. 1.11. Архитектура Android

Уровень абстракции аппаратного обеспечения (HAL) предоставляет стандартные интерфейсы. HAL состоит из нескольких библиотечных модулей, каждый из которых реализует интерфейс для определенного типа аппаратного компонента, такого как камера или модуль *bluetooth*.

Для устройств под управлением Android версии 5.0 (API уров- ня 21) или выше каждое приложение запускается в собственном процессе и с собственным экземпляром Android Runtime (ART). ART записывается для запуска нескольких виртуальных машин на устройствах путем выполнения DEX-файлов (формат байт-кода, разработанный специально для Android, который оптимизирован для минимального объема памяти).

До версии Android 5.0 (уровень API 21) Dalvik (регистровая виртуальная машина для выполнения программ, написанных на языке программирования Java, входит в операционную систему Android) была машиной времени исполнения Android. Android также включает в себя набор основных библиотек времени выполнения, которые обеспечивают большую часть функциональных возможностей языка программирования Java.

Многие базовые компоненты и сервисы Android, такие как ART и HAL, построены из собственного кода, для которого требуются библиотеки, написанные на C и C++. Платформа Android предоставляет API-интерфейсы Java Framework, чтобы выявить функциональность некоторых из этих родных библиотек для приложений. Например, доступ к OpenGL ES можно получить через Java OpenGL API платформы Android, чтобы добавить поддержку для рисования и управления 2D- и 3D-графикой в приложении.

Весь набор функций Android доступен через API, написанный на языке Java. Эти ***API-интерфейсы*** образуют строительные блоки, необходимые для создания Android-приложений, упрощая повторное использование компонентов и сервисов. Разработчики имеют полный доступ к тем же API-интерфейсам платформы, которые используют системные приложения Android.

Системные приложения Android поставляются с набором основных приложений для электронной почты, SMS-сообщений, календарей, просмотра контактов и т. д. Системные приложения функционируют как самостоятельные, а также предоставляют ключевые возможности, которые разработчики могут получать из своего собственного приложения.

Для применения новых возможностей языка Java необходимо также использовать набор инструментов *Jack*. С его помощью Android компилирует языковой источник Java в считываемый Android байт-код Dalvik Executable (DEX). В *Jack* предусмотрен собственный формат библиотеки *jack*.

1.2.2. Процесс сборки Android-приложения

Когда запускается сборка, первым делом читается файл *AndroidManifest.xml*, в нем есть важные параметры, такие как *package* (например, by.belstu.app) и *targetSdkVersion*.

Затем вызывается программа *aapt* (Android Asset Packaging Tool), которой передается *AndroidManifest.xml*, папка с ресурсами *res/*, *assets/*, путь к *android.jar* нужной target-версии. Программа *aapt* проверяет ресурсы и компилирует их, создавая при этом класс *R.java*, содержащий идентификаторы ресурсов и файл *resources.arsc*, в котором имеется информация об XML-ресурсах и их атрибутах (рис. 1.12).

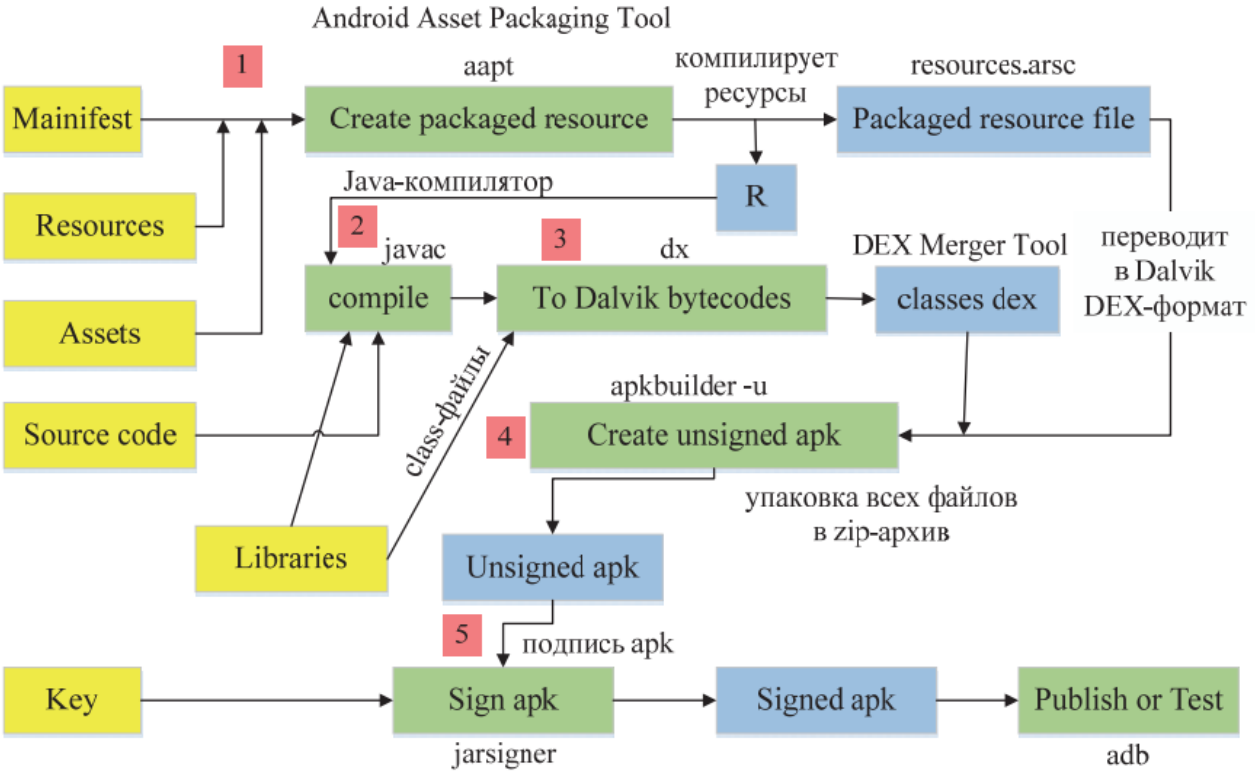


Рис. 1.12. Процесс сборки Android-приложения

Далее подхватываются все библиотеки, которые используются в проекте и запускается Java-компилятор *javac*. Полученные *class*-файлы передаются в программу *dx*, которая переводит их в DEX-формат. Причем для оптимизации готовые библиотеки дексируются отдельно,
а классы проекта отдельно (оптимизация в том, что индексированные библиотеки можно закешировать). Если собралось несколько

DEXфайлов, то все они объединяются при помощи *DEX Merger Tool*. В конечном итоге получается единственный файл *classes.dex* (или несколько, если используется *multidex*) (рис. 1.12). Теперь есть все компоненты и можно собирать apk. По сути, это просто упаковка всех файлов в zip-архив, но используется специальная программа *apkbuilder*. После ее выполнения получаем неподписанный apk-файл, то есть без папки META-INF внутри.

Последний этап – подпись apk. Берется заранее сгенерированный ключ и передается в *jarsigner* вместе с неподписанным apk-файлом. На выходе имеем готовое приложение, которое можно устанавливать (рис. 1.12).