Министерство образования Республики Беларусь

ПОЛОЦКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра вычислительных систем и сетей

**ПРОГРАММИРОВАНИЕ МОБИЛЬНЫХ СИСТЕМ**

Методические указания по выполнению   
лабораторных работ для студентов специальности 1-40 02 01  
«Вычислительные машины, системы и сети»  
всех форм обучения

А.О. Лукьянов

Полоцк 2017

**Лабораторная работа №1**

«Создание первого приложения под Android. Ресурсы в Android приложении»

**Цель работы**

Создать приложение под Android. Изучить жизненный цикл активности. Ознакомиться с основными видами ресурсов. Научиться использовать ресурсы в разрабатываемых приложениях.

**Теоретическая часть**

**Активности (Activity) в Android**

При создании экранов графического интерфейса пользователя наследуется класс Activity и используются представления (View) для взаимодействия с пользователем.

Каждая Активность – это экран (по аналогии с Формой), который приложение может показывать пользователям. Чем сложнее создаваемое приложение, тем больше экранов (Активностей) потребуется. При создании приложения потребуется, как минимум, начальный (главный) экран, который обеспечивает основу пользовательского интерфейса приложения. При необходимости этот интерфейс дополняется второстепенными Активностями, предназначенными для ввода информации, ее вывода и предоставления дополнительных возможностей. Запуск (или возврат из) новой Активности приводит к «перемещению» между экранами UI.

Большинство Активностей проектируются таким образом, чтобы использовать все экранное пространство, но можно также создавать полупрозрачные или плавающие диалоговые окна.

**Создание Активности**

Для создания новой Активности наследуется класс Activity. Внутри реализации класса необходимо определить пользовательский интерфейс и реализовать требуемый функционал. Базовый каркас для новой Активности показан ниже:

*package com.example.myapplication;*

*import android.app.Activity;*

*import android.os.Bundle;*

*public class MyActivity extends Activity {*

*/\*\* Вызывается при создании Активности \*/*

*@Override*

*public void onCreate(Bundle savedInstanceState) {*

*super.onCreate(savedInstanceState);*

*}*

*}*

Базовый класс Activity представляет собой пустой экран, который не особенно полезен, поэтому первое, что вам нужно сделать, это создать пользовательский интерфейс с помощью Представлений (View) и разметки(Layout).

Представления (View) – это элементы UI, которые отображают информацию и обеспечивают взаимодействие с пользователем. Android предоставляет несколько классов разметки (Layout), называемых также View Groups, которые могут содержать внутри себя несколько Представлений, для создания пользовательского интерфейса приложения.

Чтобы назначить пользовательский интерфейс для Активности, внутри обработчика onCreate используется метод setContentView:

*@Override*

*public void onCreate(Bundle savedInstanceState) {*

*super.onCreate(savedInstanceState);*

***TextView textView = new TextView(this);***

***setContentView(textView);***

*}*

В этом примере в качестве UI для Активности выступает объект класса TextView.

При создании реальных приложений чаще применяется метод проектирования, использующий ресурсы приложения, отделенные от кода. Такой подход позволяет создавать приложения, обеспечивающие высокое качество реализации UI, не зависящее от условий работы программы: приложения предлагают удобный для пользователя языковый интерфейс (зависит от локализации), слабо зависят от разрешения и размеров экрана и т. д.). Самое главное, что такая адаптация приложения к новым условиям не требует каких-либо изменений в коде приложения, нужно только обеспечить необходимые ресурсы (картинки, локализованные строки и т. п.). Это стандартный для Android подход показан ниже:

*@Override*

*public void onCreate(Bundle savedInstanceState) {*

*super.onCreate(savedInstanceState);*

***setContentView(R.layout.main);***

*}*

Для использования Активности в приложении ее необходимо зарегистрировать в Манифесте путем добавления элемента <activity> внутри узла <application>, в противном случае ее невозможно будет использовать. Ниже показано, как создать элемент <activity> для Активности MyActivity:

*<activity android:label="@string/app\_name"*

***android:name=".MyActivity">***

*</activity>*

В теге <activity> можно добавлять элементы <intent-filter> для указания Намерений (Intent), которые Активность будет отслеживать. Каждый «Фильтр Намерений» определяет одно или несколько действий (action) и категорий (category), которые поддерживаются Активностью. Важно знать, что Активность будет доступна из главного меню запуска приложений только в случае, если в Манифесте для нее указан <intent-filter> для действия MAIN и категории LAUNCHER, как показано на примере:

*<activity android:label="@string/app\_name"*

*android:name=".MyActivity">*

***<intent-filter>***

***<action android:name="android.intent.action.MAIN" />***

***<category android:name="android.intent.category.LAUNCHER" />***

***</intent-filter>***

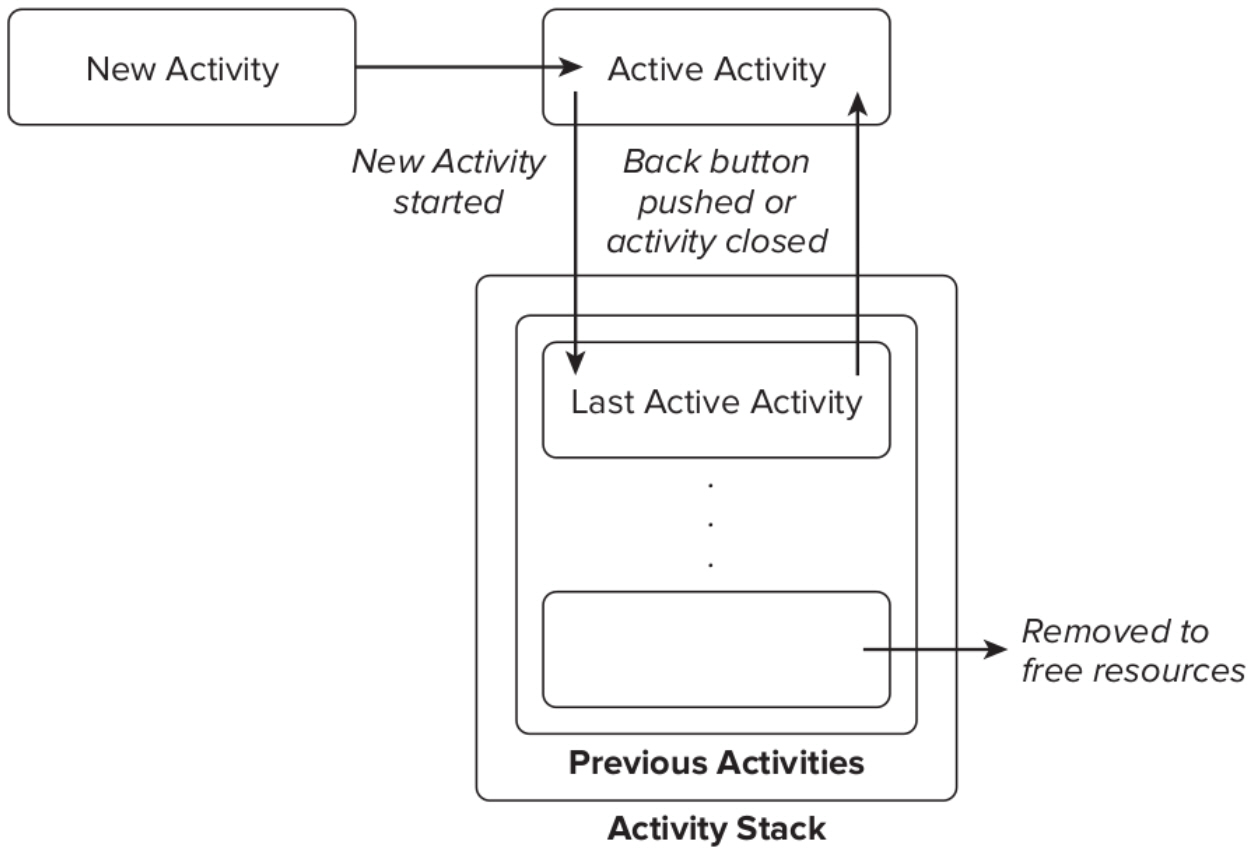
*</activity>*

**Жизненный цикл Активности**

Для создания приложений, правильно управляющих ресурсами предоставляющих пользователю удобный интерфейс, важно хорошее понимание жизненного цикла Активности. Это связано с тем, что приложения Android не могут контролировать свой жизненный цикл, ОС сама управляет всеми процессами и, как следствие, Активностями внутри них. При этом, состояние Активности помогает ОС определить приоритет родительского для этой Активности Приложения (Application). А приоритет Приложения влияет на то, с какой вероятности его работа (и работа дочерних Активностей) будет прервана системой.

*Стеки Активностей*

Состояние каждой Активности определяется ее позицией в стеке (LIFO) Активностей, запущенных в данный момент. При запуске новой Активности представляемый ею экран помещается на вершину стека. Если пользователь нажимает кнопку «назад» или эта Активности закрывается каким-то другим образом, на вершину стека перемещается (и становится активной) нижележащая Активность. Данный процесс показан на диаграмме:



На приоритет приложения влияет его самая приоритетная Активность. Когда диспетчер памяти ОС решает, какую программу закрыть для освобождения ресурсов, он учитывает информацию о положении Активности в стеке для определения приоритета приложения.

*Состояния Активностей*

Активности могут находиться в одном из четырех возможных состояний:

**• Активное (Active ).** Активность находится на переднем плане (на вершине стека) и имеет возможность взаимодействовать с пользователем. Android будет пытаться сохранить ее работоспособность любой ценой, при необходимости прерывая работу других Активностей, находящихся на более низких позициях в стеке для предоставления необходимых ресурсов. При выходе на передний план другой Активности работа данной Активности будет приостановлена или остановлена.

• **Приостановленное (Paused ).** Активность может быть видна на экране, но не может взаимодействовать с пользователем: в этот момент она приостановлена. Это случается, когда на переднем плане находятся полупрозрачные или плавающие (например, диалоговые) окна. Работа приостановленной Активности может быть прекращена, если ОС необходимо выделить ресурсы Активности переднего плана. Если Активность полностью исчезает с экрана, она останавливается.

**• Остановленное (Stopped ).** Активность невидима, она находится в памяти, сохраняя информацию о своем состоянии. Такая Активность становится кандидатом на преждевременное закрытие, если системе потребуется память для чего-то другого. При остановке Активности разработчику важно сохранить данные и текущее состояние пользовательского интерфейса (состояние полей ввода, позицию курсора и т. д.). Если Активность завершает свою работу или закрывается, он становится неактивным.

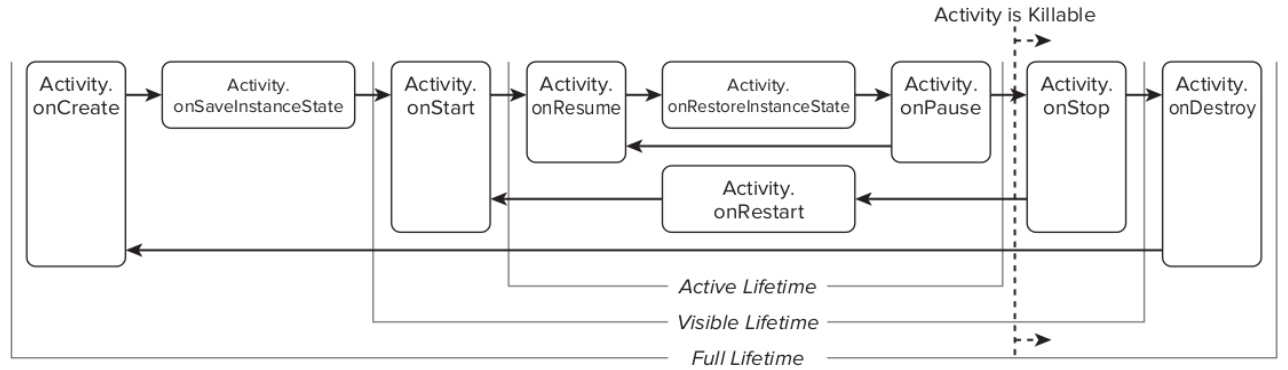
**• Неактивное (Inactive).** Когда работа Активности завершена, и перед тем, как она будет запущена, данная Активности находится в неактивном состоянии. Такие Активности удаляются из стека и должны быть (пере)запущены, чтобы их можно было использовать.

Изменение состояния приложения – недетерминированный процесс и управляется исключительно менеджером памяти Android. При необходимости Android вначале закрывает приложения, содержащие неактивные Активности, затем остановленные и, в крайнем случае, приостановленные.

Для обеспечения полноценного интерфейса приложения, изменения его состояния должны быть незаметными для пользователя. Меняя свое состояние с приостановленного на остановленное или с неактивного на активное, Активность не должна внешне меняться. При остановке или приостановке работы Активности разработчик должен обеспечить сохранение состояния Активности, чтобы его можно было восстановить при выходе Активности на передний план. Для это в классе Activity имеются обработчики событий, переопределение которых позволяет разработчику отслеживать изменение состояний Активности.

*Отслеживание изменений состояния Активности*

Обработчики событий класса Activity позволяют отслеживать изменения состояний соответствующего объекта Activity во время всего жизненного цикла:



Ниже показан пример с заглушками для таких методов – обработчиков событий:

*package com.example.myapplication;*

*import android.app.Activity;*

*import android.os.Bundle;*

*public class MyActivity extends Activity {*

*// Вызывается* ***при создании*** *Активности*

*@Override*

*public void onCreate(Bundle savedInstanceState) {*

*super.onCreate(savedInstanceState);*

*// Инициализирует Активность.*

*}*

*// Вызывается после завершения метода onCreate*

*// Используется* ***для восстановления состояния*** *UI*

*@Override*

*public void onRestoreInstanceState(Bundle savedInstanceState) {*

*super.onRestoreInstanceState(savedInstanceState);*

*// Восстановить состояние UI из объекта savedInstanceState.*

*// Данный объект также был передан методу onCreate.*

*}*

*// Вызывается перед тем, как Активность* ***снова*** *становится видимой*

*@Override*

*public void onRestart(){*

*super.onRestart();*

*// Восстановить состояние UI с учетом того,*

*// что данная Активность* ***уже была видимой****.*

*}*

*// Вызывается когда Активность* ***стала видимой***

*@Override*

*public void onStart(){*

*super.onStart();*

*// Проделать необходимые действия для*

*// Активности,* ***видимой*** *на экране*

*}*

*// Должен вызываться в начале* ***видимого*** *состояния.*

*// На самом деле Android вызывает данный обработчик только*

*// для Активностей, восстановленных из* ***неактивного*** *состояния*

*@Override*

*public void onResume(){*

*super.onResume();*

*// Восстановить приостановленные обновления UI,*

*// потоки и процессы, «замороженные, когда*

*// Активность была в неактивном состоянии*

*}*

*// Вызывается перед выходом из* ***активного*** *состояния,*

*// позволяя сохранить состояние в объекте savedInstanceState*

*@Override*

*public void onSaveInstanceState(Bundle savedInstanceState) {*

*// Объект savedInstanceState будет в последующем*

*// передан методам onCreate и onRestoreInstanceState*

*super.onSaveInstanceState(savedInstanceState);*

*}*

*// Вызывается перед выходом из* ***активного*** *состояния*

*@Override*

*public void onPause(){*

*// «Заморозить» обновления UI, потоки или*

*// «трудоемкие» процессы, ненужные, когда Активность*

*// не на переднем плане*

*super.onPause();*

*}*

*// Вызывается перед выходом из* ***видимого*** *состояния*

*@Override*

*public void onStop(){*

*// «Заморозить» обновления UI, потоки или*

*// «трудоемкие» процессы, ненужные, когда Активность*

*// не на переднем плане.*

*// Сохранить все данные и изменения в UI, так как*

*// процесс может быть в любой момент убит системой*

*super.onStop();*

*}*

*// Вызывается перед уничтожением активности*

*@Override*

*public void onDestroy(){*

*// Освободить все ресурсы, включая работающие потоки,*

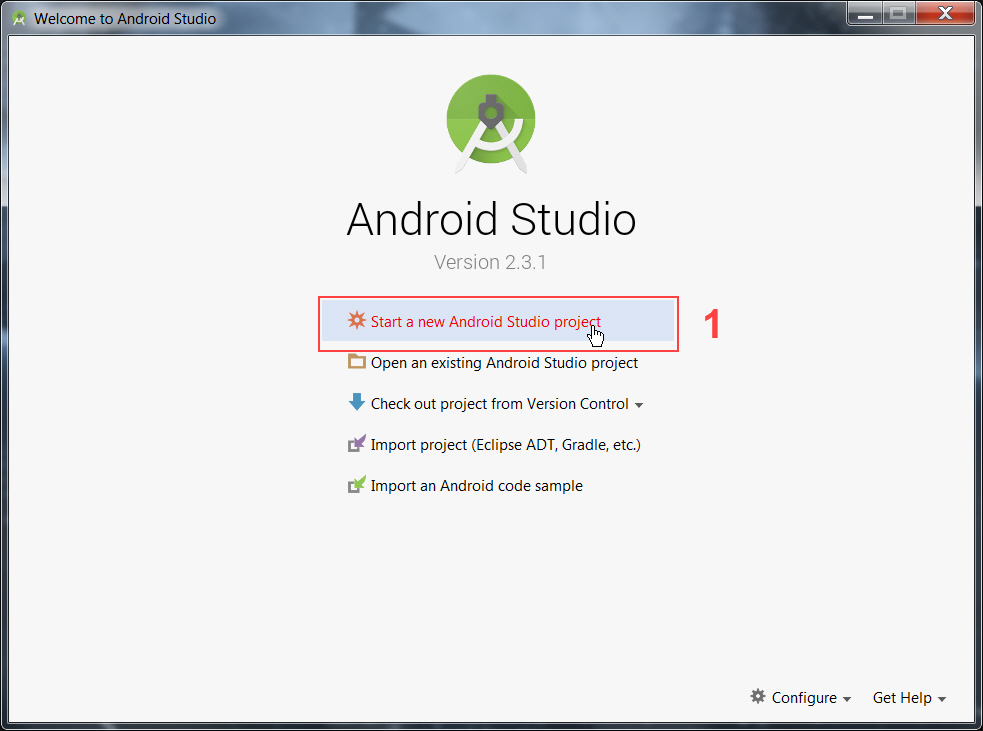
*// соединения с БД и т. д.*

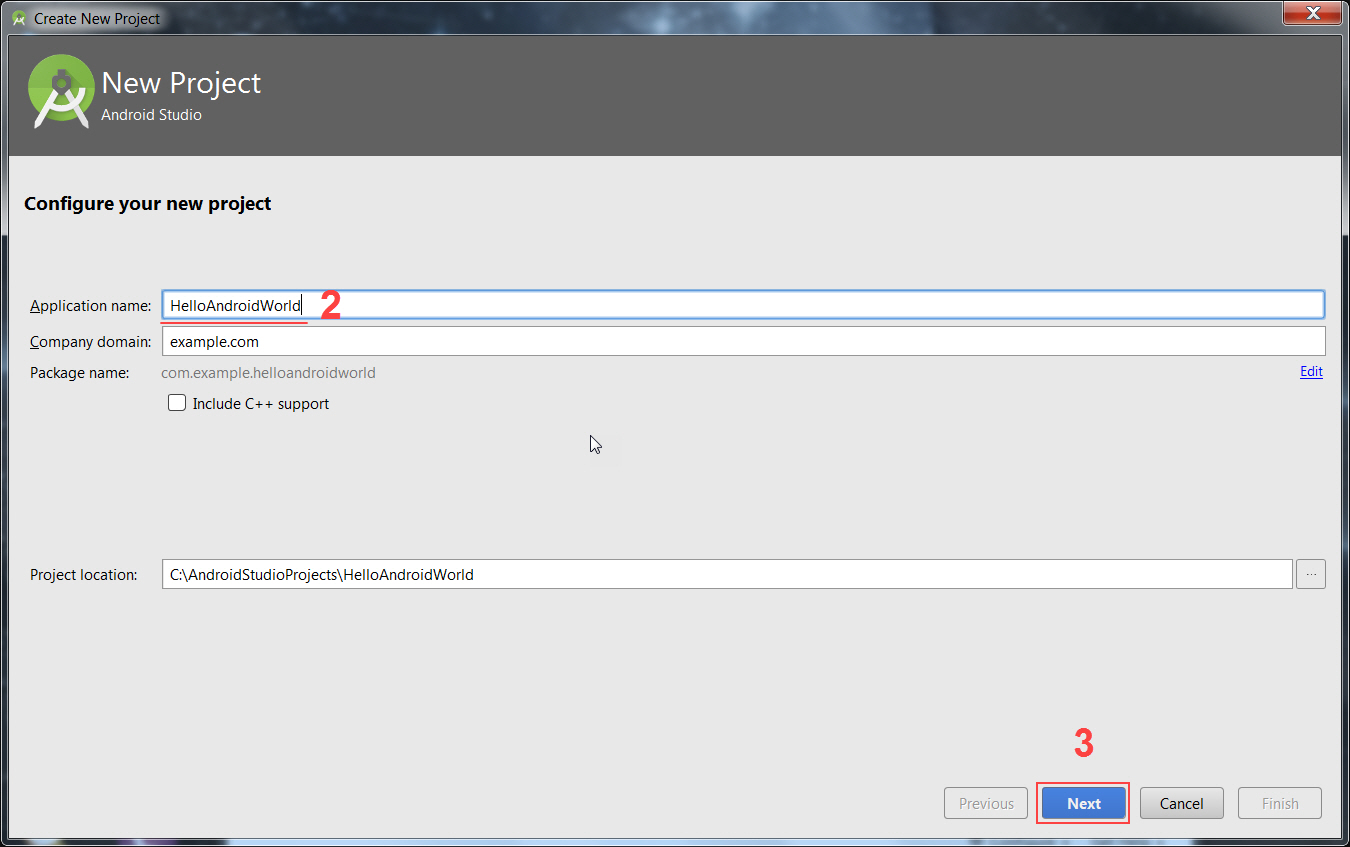
*super.onDestroy();*

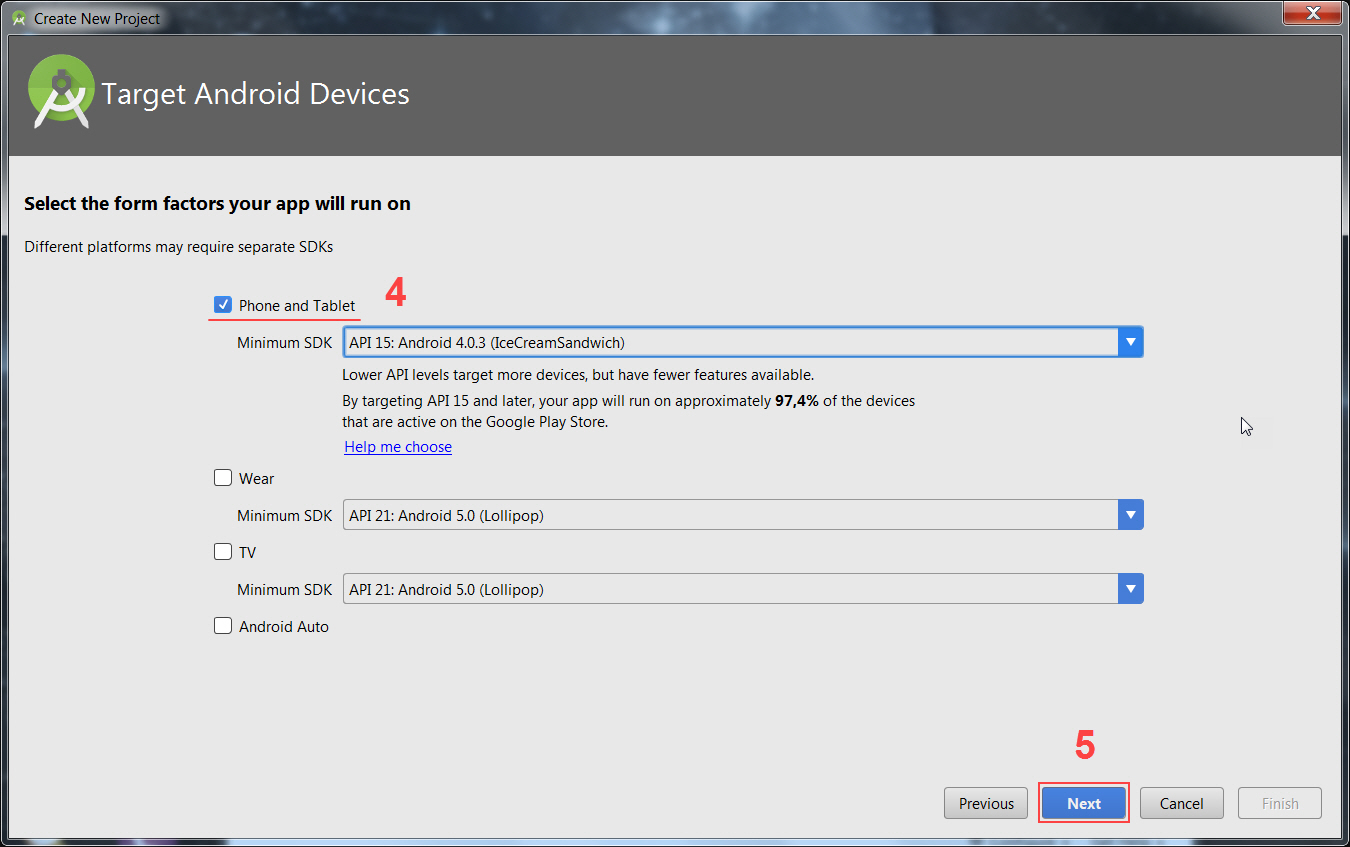
*}*

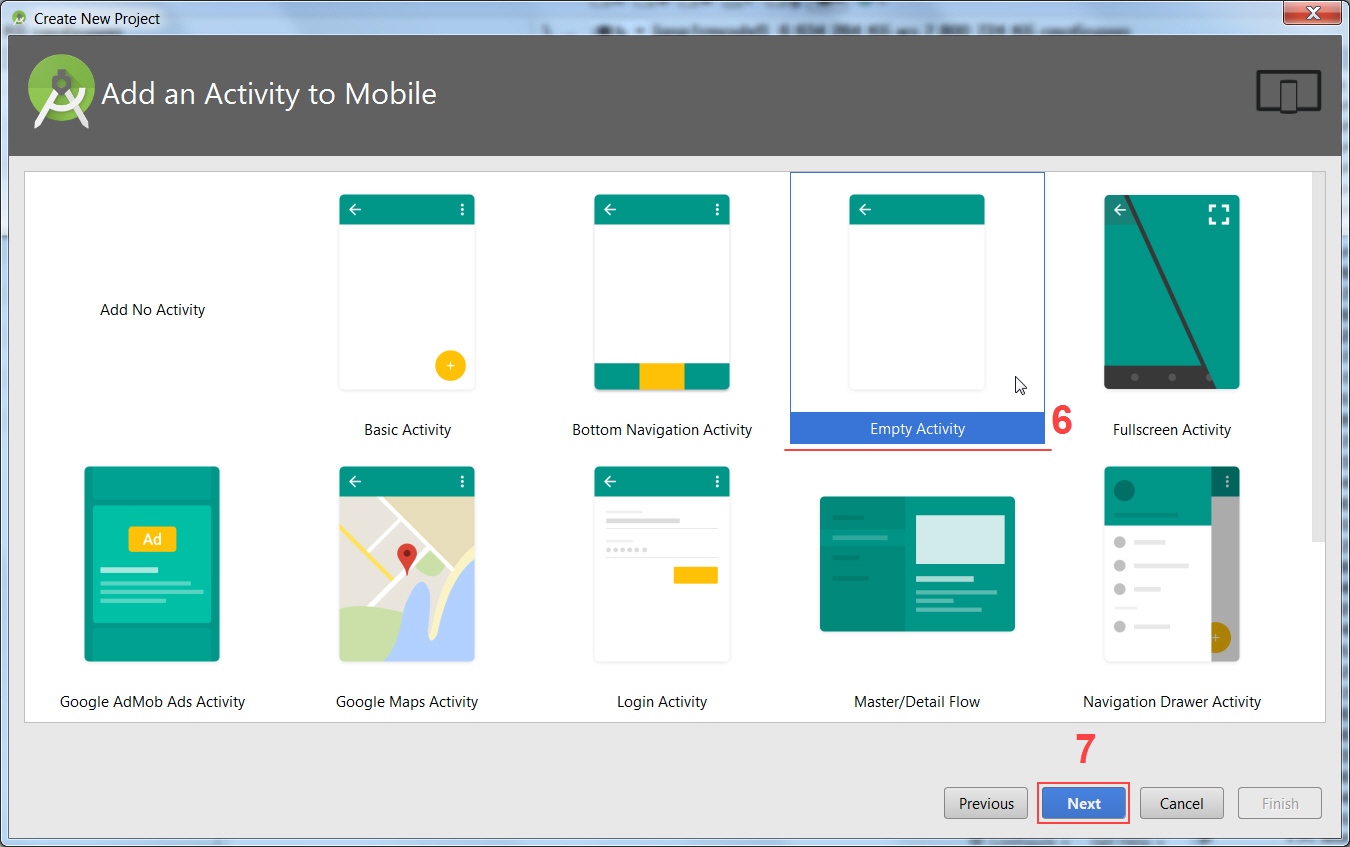
*}*

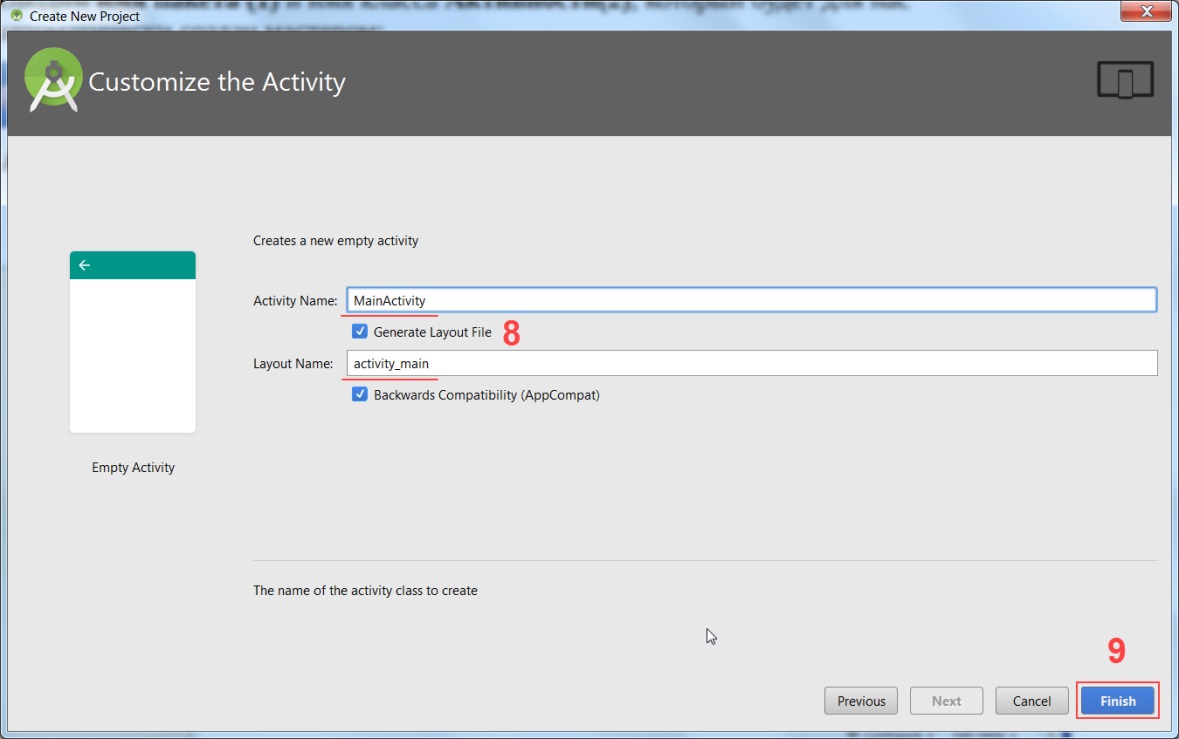
**Создание приложения в Android Studio**











**Отделение ресурсов от кода программы**

Независимо от используемой среды разработки, весьма разумно отделять используемые приложением ресурсы от кода. Внешние ресурсы легче поддерживать, обновлять и контролировать. Такая практика также позволяет описывать альтернативные ресурсы для поддержки вашим приложением различного оборудования и реализовывать локализацию приложения.

Приложения Android используют разнообразные ресурсы из внешних (по отношению к коду) файлов, от простых (строки и цвета) до более сложных (изображения, анимации, визуальные стили). Чрезвычайно полезно также отделять от кода такие важные ресурсы, как разметки экранов (Layout), используемые в Активностях. Android автоматически выбирает наиболее подходящие варианты из дерева ресурсов приложения, содержащие разные значения для разных аппаратных конфигураций, языков и регионов, не требуя при этом ни единой строчки кода.

**Создание ресурсов**

Ресурсы приложения хранятся в каталоге **res** в дереве каталогов проекта. Плагин ADT автоматически создает каталог res с подкаталогами **values**, **layout** и **drawable-\***, в которых хранятся, соответственно: строковые константы, разметка по умолчанию и иконка приложения для разных плотностей пикселей на экране.

Для девяти главных типов ресурсов используются разные подкаталоги каталога **res**, это:

• простые значения

• изображения

• разметка

• анимация

• стили

• меню

• настройки поиска

• XML

• «сырые» данные

При сборке пакета .apk эти ресурсы максимально эффективно компилируются и включаются в пакет.

Для работы с ресурсами внутри кода плагин ADT автоматически генерирует файл класса **R**, содержащий ссылки на все ресурсы. Имена файлов ресурсов могут содержать только латинские буквы в нижнем регистре, подчеркивания и точки.

**Простые значения (values)**

Android поддерживает следующие типы значений: строки, цвета, размеры и массивы (строковые и целочисленные). Эти данные хранятся в виде XML-файла в каталоге **res/values**. Ниже показан пример подобного файла:

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

*<resources>*

*<string name="app\_name">To Do List</string>*

*<color name="app\_background">#FF0000FF</color>*

*<dimen name="default\_border">5px</dimen>*

*<array name="string\_array">*

*<item>Item 1</item>*

*<item>Item 2</item>*

*<item>Item 3</item>*

*</array>*

*<array name="integer\_array">*

*<item>3</item>*

*<item>2</item>*

*<item>1</item>*

*</array>*

*</resources>*

В примере содержатся все доступные типы простых значений, но на самом деле каждый тип ресурсов хранится в отдельном файле, например, res/values/arrays.xml содержит массивы, а res/values/strings.xml – строковые константы.

**Строки**

Строковые константы определяются с помощью тэга <string>, как показано на примере:

*<string name="greeting\_msg">Превед!</string>*

Для выделения текста в строках можно использовать HTML-тэги **<b>**, **<i>** и **<u>**.

Пример:

*<string name="greeting\_msg"><b>Превед</b>, дарлинг!</string>*

При необходимости использования данной строки в коде программы используется

вышеупомянутый класс **R**:

*String greeting = getString(****R.string.greeting\_msg****);*

**Цвета**

Для описания цветов используется тэг <color>. Значения цвета указываются в шестнадцатеричном виде в одном из следующих форматов:

• #RGB

• #ARGB

• #RRGGBB

• #AARRGGBB

В примере показаны описания полупрозрачного красного цвета и непрозрачного зеленого:

*<color name="transparent\_red">#77FF0000</color>*

*<color name="opaque\_green">#0F0</color>*

**Использование внешних ресурсов в коде приложения**

Доступ к ресурсам в коде осуществляется с помощью автоматически генерируемого класса **R**, точнее, его подклассов. Например, класс **R** в нашем проект выглядит так:

*package com.example.helloandroidworld;*

*public final class R {*

*public static final class attr {*

*}*

*public static final class color {*

*public static final int screen\_bkg\_color=0x7f040001;*

*public static final int text\_color=0x7f040002;*

*public static final int view\_bkg\_color=0x7f040000;*

*}*

*public static final class drawable {*

*public static final int ic\_launcher=0x7f020000;*

*}*

*public static final class layout {*

*public static final int main=0x7f030000;*

*}*

*public static final class string {*

*public static final int app\_name=0x7f050001;*

*public static final int hello=0x7f050000;*

*}*

*}*

Члены классов с именами, соответствующими ресурсам, являются идентификаторами в таблице ресурсов, а не самими экземплярами ресурсов. Некоторые методы и конструкторы могут принимать в качестве параметров идентификаторы ресурсов, в этом случае их можно использовать напрямую:

*setContentView(****R.layout.main****);*

*Toast.makeText(this,* ***R.string.awesome\_error****, Toast.LENGTH\_LONG).show();*

В случае, если необходим экземпляр ресурса, требуется доступ к таблице ресурсов, осуществляемый с помощью экземпляра класса Resources. Этот класс содержит геттеры для всех видов ресурсов, при этом в качестве параметров используются идентификаторы ресурсов из класса **R**:

*// Получаем доступ к таблице ресурсов*

*Resources r = getResources();*

*// и получаем необходимые экземпляры ресурсов*

*CharSequence greetingMsg = r.getText(R.string.greeting\_message);*

*Drawable icon = r.getDrawable(R.drawable.app\_icon);*

*int opaqueBlue = r.getColor(R.color.opaque\_blue);*

*float borderWidth = r.getDimension(R.dimen.standard\_border);*

*String[] stringArray = r.getStringArray(R.array.string\_array);*

*int[] intArray = r.getIntArray(R.array.integer\_array);*

**Использование ресурсов внутри ресурсов**

Для обращение к одному ресурсу внутри описания другого ресурса используется следующая нотация:

*attribute="@[packagename:]resourcetype/resourceidentifier"*

Имя пакета используется только при обращении к ресурсам из другого пакета, для обращения к своим ресурсам его указывать не требуется. В нашем случае такую нотацию можно увидеть, например, при описании элемента разметки TextView в файле **res/layout/main.xml**:

*<TextView*

*android:layout\_width="fill\_parent"*

*android:layout\_height="wrap\_content"*

*android:background="@color/view\_bkg\_color"*

*android:text="@string/hello"*

*android:textColor="@color/text\_color" />*

Аналогичным образом осуществляется доступ к системным ресурсам, в качестве именипакета при этом указывается @android:

*<EditText*

*android:id="@+id/myEditText"*

*android:layout\_width="fill\_parent"*

*android:layout\_height="wrap\_content"*

*android:text="@android:string/ok"*

*android:textColor="@android:color/primary\_text\_light" />*

Обратите внимание на атрибут android:id="@+id/myEditText". Такая запись позволяет присвоить идентификатор вашему компоненту ресурса (в данном случае элементу разметки) и в дальнейшем использовать этот идентификатор для получения экземпляра ресурса.

Ссылки на текущие визуальные стили позволяют использовать действующие в настоящий момент атрибуты стилей, вместо того, чтобы заново их определять. Для указания ссылки на такой ресурс используется символ ? Вместо @:

*<EditText*

*android:id="@+id/myEditText"*

*android:layout\_width="fill\_parent"*

*android:layout\_height="wrap\_content"*

*android:hint="@string/edit\_text\_hint"*

*android:backgroundColor="?android:backgroundColor" />*

**Локализация приложения с помощью внешних ресурсов**

Одно из основных преимуществ использования внешних по отношению к коду приложения ресурсов – возможность использования механизма автоматического выбора ресурсов. Пользуясь описанным ниже механизмом, можно создавать индивидуальные ресурсы для различных аппаратных конфигураций, языков, регионов и т. д. Android во время выполнения приложения сам выберет наиболее подходящие ресурсы.

Для индивидуальной настройки приложения доступны следующие возможности:

• **MCC** (Mobile Country Code) и **MNC** (Mobile Network Code)

• **Язык и регион.** Например, en-rUS для английского языка в американском регионе

(маленькая **r** – от «**r**egion»), ru для русского

• **Размер экрана** (*small, medium* или *large*)

• **«Широкоформатность» экрана** (*long* или *notlong*)

• **Ориентация экрана** (*port, land* или *square*)

• **Плотность пикселей на экране** (*ldpi, mdpi, hdpi* или *nodpi*)

• **Тип сенсорного экрана** (*notouch, stylus* или *finger*)

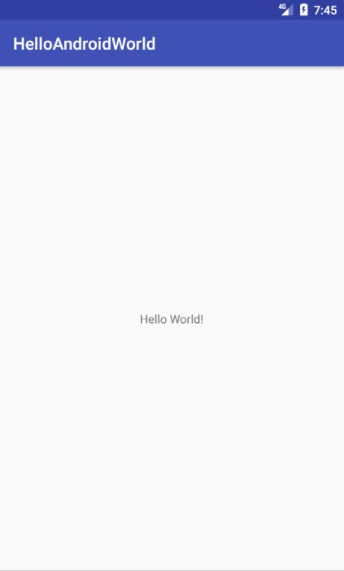
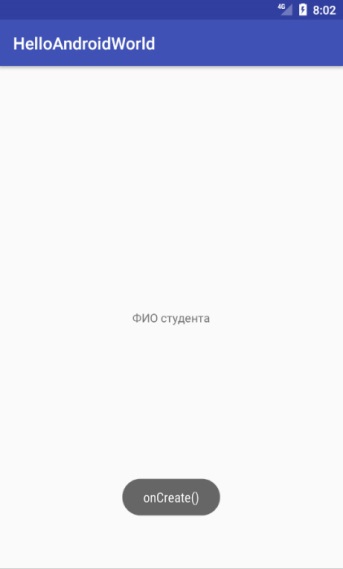
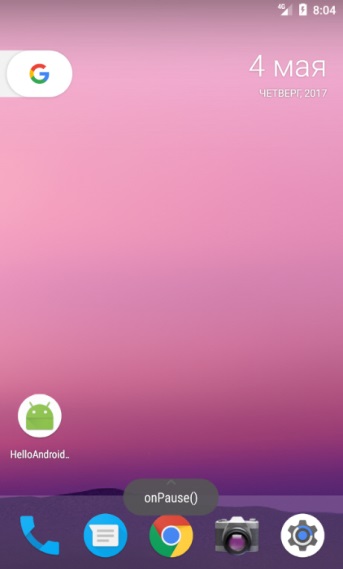
• **Доступность клавиатуры** (*keysexposed, keyshidden* или *keyssoft* )

• **Тип ввода** (*nokeys, qwerty* или *12key* )

• **Способ навигации** (*nonav, dpad, trackball* или *wheel* )

Альтернативные ресурсы располагаются в подкаталогах каталога **res**, при этом используются модификаторы стандартных имен подкаталогов с ресурсами. Например, файл, содержащий строковые константы для русского языка, будет располагаться по следующему пути: **res/values-ru/strings.xml** Модификатором в данном случае является суффикс **-ru**, добавленный к имени каталога **values**.

**Пример выполнения задания**

**Задание**

1. В проекте HelloAndroidWorld откройте в редакторе класс HelloAndroidWorld.java.
2. Переопределите методы onPause, onStart, onRestart для класса и внесите изменения в метод onCreate:

*package com.example.helloandroidworld;*

*import android.app.Activity;*

*import android.os.Bundle;*

*import android.widget.Toast;*

*public class HelloAndroidWorld extends Activity {*

*/\*\* Called when the activity is first created. \*/*

*@Override*

*public void onCreate(Bundle savedInstanceState) {*

*super.onCreate(savedInstanceState);*

*setContentView(R.layout.main);*

*Toast.makeText(this, "onCreate()", Toast.LENGTH\_LONG).show();*

*}*

*@Override*

*protected void onPause() {*

*Toast.makeText(this, "onPause()", Toast.LENGTH\_LONG).show();*

*super.onPause();*

*}*

*@Override*

*protected void onRestart() {*

*super.onRestart();*

*Toast.makeText(this, "onRestart()", Toast.LENGTH\_LONG).show();*

*}*

*@Override*

*protected void onStart() {*

*super.onStart();*

*Toast.makeText(this, "onStart()", Toast.LENGTH\_LONG).show();*

*}*

*}*

1. В проекте HelloAndroidWorld отредактировать файл **colors.xml** в каталоге **res/values**:

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

*<resources>*

*<color name="view\_bkg\_color">#FF0</color>*

*<color name="screen\_bkg\_color">#F88</color>*

*<color name="text\_color">#8004</color>*

*</resources>*

1. Отредактируйте содержимое файла **res/values/strings.xml**:

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

*<resources>*

*<string name="hello">Полное ФИО студента</string>*

*<string name="app\_name">HelloAndroidWorld</string>*

*</resources>*

1. Внесите изменения в файл разметки **res/layout/activity\_main.xml:**

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

*<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"*

*android:layout\_width="fill\_parent"*

*android:layout\_height="fill\_parent"*

*android:orientation="vertical"*

***android:background="@color/screen\_bkg\_color">***

*<TextView*

*android:layout\_width="fill\_parent"*

*android:layout\_height="wrap\_content"*

***android:text="@string/hello"***

***android:background="@color/view\_bkg\_color"***

***android:textColor="@color/text\_color"/>***

*</LinearLayout>*

1. Сохранить все несохраненные файлы и запустить приложение.
2. Убедиться, что внешний вид приложения изменился в соответствии с определенными нами ресурсами.
3. Изменить значения цветов, прозрачность и HTML-тэги в строковых константах для изменения внешнего вида приложения.
4. Изменить ресурсы таким образом, чтобы приложение отображало два варианта интерфейса на русском языке и на английском в зависимости от выбранного языка в системе.

**Контрольные вопросы**

1. Дайте определение понятию «Активность».
2. Перечислите и охарактеризуйте состояния Активности.
3. Перечислите методы-обработчики, вызываемые при переходе между состояниями Активности.
4. Дайте определение понятию «Стек Активностей».
5. Опишите структуру «Стека Активностей».
6. Дайте определение понятия «Ресурс» в Android приложении.
7. Перечислите и охарактеризуйте основные типы ресурсов.
8. В каком виде хранятся простые ресурсы?
9. Какие форматы хранения цвета используются в ОС Android?
10. Перечислите каталоги хранения основных типов ресурсов.

**Содержание отчёта**

1. Ф.И.О., группа, название лабораторной работы.
2. Цель работы.
3. Описание проделанной работы.
4. Результаты выполнения лабораторной работы.
5. Выводы.

Отчет и исходные коды запаковать в архив с названием по следующему шаблону:

PMS-Lab1-«группа, аббревиатуру на латинице»-«Фамилия на латинице»

Пример: PMS-Lab1-11VS-Ivanov.zip

**Лабораторная работа №2**

«Использование элементов пользовательского интерфейса. Использование ListView»

**Цель работы**

Ознакомиться с различными типами элементов интерфейса. Научиться использовать элементы пользовательского интерфейса в разрабатываемых приложениях. Ознакомиться с использованием адаптеров в Android. Научиться использовать элемент ListView в разрабатываемых приложениях.

**Теоретическая часть**

**Основные понятия и связи между ними**

• Представления (View) являются базовым классом для все визуальных элементов UI (элементов управления (Control) и виджетов (Widget)). Все эти элементы, в том числе и разметка (Layout), являются расширениями класса View.

• Группы Представлений (ViewGroup) – потомки класса View; могут содержать в себе несколько дочерних Представлений. Расширение класса ViewGroup используется для создания сложных Представлений, состоящих из взаимосвязанных компонентов. Класс ViewGroup также является базовым для различных Разметок (Layout).

• Активности (Activity) – отображаемые экраны или окна (с точки зрения построения UI), являются «андроидными эквивалентами» форм. Для отображения UI Активности используют Представления (как правило, Разметки (Layout)).

Для создания приложений с уникальным интерфейсом разработчику иногда приходится расширять и модифицировать стандартные Представления, комбинируя их со стандартными.

**Представления (View)**

Android предоставляет разработчику возможность использования богатого набора готовых Представлений:

• **TextView.** Стандартный элемент, предназначенный для вывода текста. Поддерживает многострочное отображение, форматирование и автоматический перенос.

• **EditText.** Редактируемое поле для ввода текста. Поддерживает многострочный ввод, перенос слов на новую строку и и текст подсказки.

• **ListView.** Группа представлений (ViewGroup), которая формирует вертикальный список элементов, отображая их в виде строк внутри списка. Простейший объект ListView использует TextView для вывода на экран значений toString(), принадлежащих элементом массива.

• **Spinner.** Составной элемент, отображающий TextView в сочетании с соответствующим ListView, которое позволяет выбрать элемент списка для отображения в текстовой строке. Сама строка состоит из объекта TextView и кнопки, при нажатии на которую всплывает диалог выбора. Внешне этот элемент напоминает тэг <SELECT> в HTML.

• **Button.** Стандартная кнопка, которую можно нажимать.

• **CheckBox.** Кнопка, имеющая два состояния. Представлена в виде отмеченного или неотмеченного флажка («галки»).

• **RadioButton.** «Радиокнопка», позволяет выбрать только один из нескольких вариантов.

• **ViewFlipper.** Группа представлений (ViewGroup), позволяющая определить набор элементов и горизонтальную строку, в которой может выводиться только одно Представление (View). При этом переходы между отображающимися элементами осуществляются с помощью анимации.

Android предлагает и более продвинутые реализации Представлений, такие, как элементы для выбора даты и времени, поля ввода с автоматическим дополнением, галереи, вкладки и даже карты (MapView).

Более полный список поддерживаемых системой Представлений можно увидеть по адресу http://developer.android.com/guide/tutorials/views/index.html

Кроме готовых Представлений, разработчик, при необходимости, может создавать собственные, расширяя класс View или его подклассы.

**Разметка (Layout)**

Разметка (Layout) является расширением класса ViewGroup и используется для размещения дочерних компонентов на экране устройства. Используя вложенные Разметки, можно создавать пользовательские интерфейсы любой сложности.

Наиболее часто используемые виды Разметки:

• **FrameLayout.** Самая простая разметка, прикрепляет каждое новое дочернее Представление к левому верхнему углу экрана, накладывая новый элемент на предыдущий, заслоняя его.

• **LinearLayout.** Помещает дочерние Представления в горизонтальный или вертикальный ряд. Вертикальная разметка представляет собой колонку, а горизонтальная – строку с элементами. Данная разметка позволяет задавать не только размеры, но и «относительный вес» дочерних элементов, благодаря чему можно гибко контролировать их размещение на экране.

• **RelativeLayout.** Наиболее гибкий среди стандартных видов разметки. Позволяет указывать позиции дочерних Представлений относительно границ свободного пространства и других Представлений.

• **TableLayout.** Позволяет размещать дочерние Представления внутри ячеек «сетки», состоящей из строк и столбцов. Размеры ячеек могут оставаться постоянными или автоматически растягиваться при необходимости.

• **Gallery.** Представляет элементы в виде прокручиваемого горизонтального списка (обычно графические элементы).

Актуальную информацию о свойствах и возможностям разных видов разметки можно получить по адресу http://developer.android.com/guide/topics/ui/layout-objects.html

Наиболее предпочтительный способ реализации разметки экрана – использование внешних ресурсов: XML-файлов, описывающих размещение элементов на экране и их параметры.

Мы уже работали с содержимым файла res/layout/main.xml для изменения внешнего вида приложения HelloAndroidWorld, а теперь подробнее рассмотрим его содержимое:

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

*<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"*

*android:layout\_width="fill\_parent"*

*android:layout\_height="fill\_parent"*

*android:orientation="vertical" >*

*<TextView*

*android:layout\_width="fill\_parent"*

*android:layout\_height="wrap\_content"*

*android:text="@string/hello"*

*android:textColor="@color/text\_color" />*

*</LinearLayout>*

Корневым элементом нашей разметки является *<LinearLayout>*, имеющий один дочерний элемент *<TextView>*. Атрибуты *<LinearLayout>* определяют пространство имен «android» (*xmlns:android="http://schemas.android.com/apk/res/android"*), ширину (*android:layout\_width*), высоту (*android:layout\_height*) и ориентацию (*android:orientation*), которая определяет способ размещения дочерних элементов внутри *LinearLayout*: вертикально или горизонтально. Обратите внимание на относительное (относительно размеров родительского элемента), а не абсолютное (в пикселях) задание размеров (ширины и высоты). Такой способ определения размеров является предпочтительным и позволяет создавать дизайн приложений, слабо зависящий от размеров экрана устройства.

Элемент <TextView> в нашем случае имеет следующие атрибуты: ширину и высоту, а также ссылки на строку с именем «hello» и цвет, используемый для отображения текста «text\_color».

**Адаптеры в Android**

Адаптеры в Android являются связующими классами между данными приложения и Представлениями. Адаптер отвечает за создание дочерних Представлений, отображающими каждый элемент внутри родительского виджета, а также обеспечивает доступ к исходным данным, используемым приложением. Представления, использующие привязку к Адаптеру, должны быть потомками абстрактного класса AdapterView.

Android содержит набор стандартных Адаптеров, которые доставляют данные в стандартные виджеты пользовательского интерфейса. Двумя наиболее полезными и часто используемыми Адаптерами являются ArrayAdapter и SimpleCursorAdapter.

**ArrayAdapter** использует механизм обобщенных типов (generics) языка Java для привязки родительского класса AdapterView к массиву объектов указанного типа. По умолчанию ArrayAdapter использует метод toString() для каждого элемента в массиве, чтобы создать и заполнить текстовыми данными виджеты TextView.

**SimpleCursorAdapter** привязывает указанное в разметке Представление к столбцам Курсора, ставшего результатом запроса к СУБД или Контент-Провайдеру. Для его использования требуется описать разметку в формате XML, а затем привязать каждый столбец к Представлениям из этой разметки. Адаптер создаст Представления для каждой записи из Курсора и наполнит их данными из соответствующих столбцов.

**SimpleAdapter** позволяет привязать ListView к списку ArrayList, содержащему объекты типа Map (ассоциативные массивы, содержащие пары «ключ-значение»). Для каждого такого объекта при отображении используется один элемент из ListView. Как и для SimpleCursorAdapter, для отображения применяется XML-разметка, к элементам которой привязываются члены каждого объекта типа Map.

**Использование Адаптеров для привязки данных**

Чтобы применить Адаптер, необходимо вызвать из Представления (потомка абстрактного класса *AdapterView*) метод *setAdapter()* (или его более конкретизированный вариант).

Пример использования *ArrayAdapter*:

*ArrayList<String> myStringArray = new ArrayList<String>();*

*ArrayAdapter<String> myAdapterInstance;*

*int layoutID = android.R.layout.simple\_list\_item\_1;*

*myAdapterInstance = new ArrayAdapter<String>(this, layoutID, myStringArray);*

*myListView.setAdapter(myAdapterInstance);*

Пример использования SimpleCursorAdapter:

*String uriString = "content://contacts/people/";*

*Cursor myCursor = managedQuery(Uri.parse(uriString), null, null, null, null);*

*String[] fromColumns = new String[] {People.NUMBER, People.NAME};*

*int[] toLayoutIDs = new int[] { R.id.nameTextView, R.id.numberTextView};*

*SimpleCursorAdapter myAdapter;*

*myAdapter = new SimpleCursorAdapter(this, R.layout.simplecursorlayout, myCursor, fromColumns, toLayoutIDs);*

*myListView.setAdapter(myAdapter);*

**Задание**

1. Выполнить размещение элементов пользовательского интерфейса на одной из activity приложения, используя LinearLayout. Необходимо обратить внимание, что при размещении элементов внутри вложенного LinearLayout удобно указывать атрибут *android:layout\_weight*="1", в этом случае дочерние виджеты будут размещены по горизонтали (или вертикали) равномерно
2. Создать activity с использованием RelativeLayout.
3. Добавить нужные строки в файл **res/values/strings.xml**

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

*<resources>*

***<string name="label\_text">Введите текст:</string>***

***<string name="entry\_hint">Поле ввода</string>***

*<string name="app\_name"> HelloAndroidWorld </string>*

*</resources>*

1. Отредактировать файл разметки **res/layout/activity\_main.xml** так, чтобы он имел следующее содержимое:

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

*<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"*

*android:layout\_width="fill\_parent"*

*android:layout\_height="fill\_parent" >*

*<TextView*

*android:id="@+id/label"*

*android:layout\_width="fill\_parent"*

*android:layout\_height="wrap\_content"*

*android:text="@string/label\_text" />*

*<EditText*

*android:id="@+id/entry"*

*android:layout\_width="fill\_parent"*

*android:layout\_height="wrap\_content"*

*android:layout\_below="@id/label"*

*android:background="@android:drawable/editbox\_background"*

*android:hint="@string/entry\_hint" />*

*<Button*

*android:id="@+id/ok"*

*android:layout\_width="wrap\_content"*

*android:layout\_height="wrap\_content"*

*android:layout\_alignParentRight="true"*

*android:layout\_below="@id/entry"*

*android:layout\_marginLeft="10dip"*

*android:text="@android:string/ok" />*

*<Button*

*android:layout\_width="wrap\_content"*

*android:layout\_height="wrap\_content"*

*android:layout\_alignTop="@id/ok"*

*android:layout\_toLeftOf="@id/ok"*

*android:text="@android:string/cancel" />*

*</RelativeLayout>*

1. Отредактируйте файл AndroidManifest.xml, чтобы изменить тему, используемую приложением. Для это в узел <application> внесите следующие изменения**:**

*<application*

*android:icon="@drawable/ic\_launcher"*

*android:label="@string/app\_name"*

***android:theme="@android:style/Theme.Light"*** *>*

1. Создать activity с использованием TabWidget
2. Создать activity с использованием WebView
3. Создать новый Android проект ListViewSample
4. В каталоге **res/values** создать файл **arrays.xml** со следующим содержимым:

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

*<resources>*

*<string-array name="stations">*

*<item>Уручье</item>*

*<item>Борисовский тракт</item>*

*<item>Восток</item>*

*<item>Московская</item>*

*<item>Парк Челюскинцев</item>*

*<item>Академия наук</item>*

*<item>Площадь Якуба Коласа</item>*

*<item>Площадь Победы</item>*

*<item>Октябрьская</item>*

*<item>Площадь Ленина</item>*

*<item>Институт Культуры</item>*

*<item>Грушевка</item>*

*<item>Михалово</item>*

*<item>Петровщина</item>*

*<item>Малиновка</item>*

*<item>Каменная Горка</item>*

*<item>Кунцевщина</item>*

*<item>Спортивная</item>*

*<item>Пушкинская</item>*

*<item>Молодёжная</item>*

*<item>Фрунзенская</item>*

*<item>Немига</item>*

*<item>Купаловская</item>*

*<item>Первомайская</item>*

*<item>Пролетарская</item>*

*<item>Тракторный завод</item>*

*<item>Партизанская</item>*

*<item>Автозаводская</item>*

*<item>Могилёвская</item>*

*</string-array>*

*</resources>*

1. В каталоге **res/layout** создать файл **list\_item.xml** со следующим содержимым:

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

*<TextView xmlns:android="http://schemas.android.com/apk/res/android"*

*android:layout\_width="fill\_parent"*

*android:layout\_height="fill\_parent"*

*android:padding="10dp"*

*android:textSize="16sp" >*

*</TextView>*

1. Модифицировать метод onCreate основной Активности**:**

*@Override*

*public void onCreate(Bundle savedInstanceState) {*

*super.onCreate(savedInstanceState);*

*Resources r = getResources();*

*String[] stationsArray = r.getStringArray(R.array.stations);*

*ArrayAdapter<String> aa = new ArrayAdapter<String>(this, R.layout.list\_item, stationsArray);*

*setListAdapter(aa);*

*ListView lv = getListView();*

*}*

1. Изменить базовый класс Активности с **Activity** на **ListActivity**.
2. Для реакции на клики по элементам списка требуется добавить обработчик такого события, с помощью метода setOnItemClickListener. В качестве обработчика будет использоваться анонимный объект класса OnItemClickListener. Добавить следующий код:

*lv.setOnItemClickListener(new OnItemClickListener() {*

*public void onItemClick(AdapterView<?> parent, View v, int position, long id) {*

*CharSequence text = ((TextView) v).getText();*

*int duration = Toast.LENGTH\_LONG;*

*Context context = getApplicationContext();*

*Toast.makeText(context, text, duration).show();*

*}*

*});*

1. Запустить приложение и «покликать» по станциям метро.

**Контрольные вопросы**

1. Дайте определение понятиям «Представление» и «Группа Представлений».
2. Перечислите и охарактеризуйте основные Представления в ОС Android.
3. Перечислите и охарактеризуйте основные виды Разметки в ОС Android.
4. В каком виде реализуется разметка экрана Android приложения?
5. Возможно ли использование нескольких видов разметки на одной Активности?
6. Дайте определение понятию «Адаптер» в Android приложении.
7. Перечислите и охарактеризуйте основные виды Адаптеров.
8. Какой стандартный адаптер используется для работы с результатом запроса СУБД?
9. Охарактеризуйте Представление ListView.
10. В каком виде храниться внешний ресурс, содержащий массив данных?

**Содержание отчёта**

1. Ф.И.О., группа, название лабораторной работы.
2. Цель работы.
3. Описание проделанной работы.
4. Результаты выполнения лабораторной работы.
5. Выводы.

Отчет и исходные коды запаковать в архив с названием по следующему шаблону:

PMS-Lab1-«группа, аббревиатуру на латинице»-«Фамилия на латинице»

Пример: PMS-Lab1-11VS-Ivanov.zip

**Лабораторная работа №3**

«Использование управляющих элементов в пользовательском интерфейсе. Работа с базами данных в Android»

**Цель работы**

Научиться использовать в интерфейсе пользователя различные управляющие элементы: кнопки с изображениями, радиокнопки, чекбоксы и другие. Научиться взаимодействовать с базой данных в Android приложении.

**Теоретическая часть**

**Использование графической кнопки**

Для использования изображения вместо текста на кнопке потребуются три изображения для трех состояний кнопки: обычное, выбранное («в фокусе») и нажатое. Все эти три изображения с соответствующими состояниями описываются в одном XML файле, который используется для создания такой кнопки. Для этого необходимо:

1. Скопировать нужные изображения кнопки в каталог res/drawable-mdpi.

2. В этом же каталоге создать файл smile\_button.xml, описывающий, какие изображения в каких состояниях кнопки нужно использовать:

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

*<selector xmlns:android="http://schemas.android.com/apk/res/android">*

*<item android:drawable="@drawable/smile\_pressed"*

*android:state\_pressed="true"/>*

*<item android:drawable="@drawable/smile\_focused"*

*android:state\_focused="true"/>*

*<item android:drawable="@drawable/smile\_normal" />*

*</selector>*

3. Добавить элемент Button внутри LinearLayout в файле разметки res/layout/main.xml:

*<Button*

*android:id="@+id/button"*

*android:layout\_width="wrap\_content"*

*android:layout\_height="wrap\_content"*

*android:background="@drawable/smile\_button"*

*android:onClick="onButtonClicked"*

*android:padding="10dp" />*

4. Необходимо обратить внимание на атрибут android:onClick="onButtonClicked", указывающий, какой метод из Активности будет использоваться как обработчик нажатия на данную кнопку. Добавить этот метод в Активность:

*public void onButtonClicked(View v) {*

*Toast.makeText(this, "Кнопка нажата", Toast.LENGTH\_SHORT).show();*

*}*

**Использование виджета CheckBox**

Для использования виджета CheckBox необходимо:

1. Добавить элемент CheckBox внутри LinearLayout в файле разметки res/layout/main.xml:

*<CheckBox*

*android:id="@+id/checkbox"*

*android:layout\_width="wrap\_content"*

*android:layout\_height="wrap\_content"*

*android:onClick="onCheckboxClicked"*

*android:text="Выбери меня" />*

2. Атрибут android:onClick="onCheckboxClicked" определяет, какой метод из Активности будет использоваться как обработчик нажатия на виджет. Добавить этот метод в Активность:

*public void onCheckboxClicked(View v) {*

*if (((CheckBox) v).isChecked()) {*

*Toast.makeText(this, "Отмечено", Toast.LENGTH\_SHORT).show();*

*} else {*

*Toast.makeText(this, "Не отмечено", Toast.LENGTH\_SHORT).show();*

*}*

*}*

**Использование виджета ToggleButton**

Данный виджет хорошо подходит в качестве альтернативы радиокнопкам и чекбоксам, когда требуется переключаться между двумя взаимоисключающими состояниями, например, Включено/Выключено. Для использования данного виджета необходимо:

1. Добавить элемент ToggleButton внутри LinearLayout в файле разметки res/layout/main.xml:

*<ToggleButton android:id="@+id/togglebutton"*

*android:layout\_width="wrap\_content"*

*android:layout\_height="wrap\_content"*

*android:textOn="Звонок включен"*

*android:textOff="Звонок выключен"*

*android:onClick="onToggleClicked"/>*

2. Атрибут android:onClick="onToggleClicked" определяет, какой метод из Активности будет использоваться как обработчик нажатия на виджет. Добавить этот метод в Активность:

*public void onToggleClicked(View v) {*

*if (((ToggleButton) v).isChecked()) {*

*Toast.makeText(this, "Включено", Toast.LENGTH\_SHORT).show();*

*} else {*

*Toast.makeText(this, "Выключено", Toast.LENGTH\_SHORT).show();*

*}*

*}*

**Использование виджета RadioButton**

Радиокнопки используются для выбора между различными взаимоисключающими вариантами. Для создания группы радиокнопок используется элемент RadioGroup, внутри которого располагаются элементы RadioButton. Для использования данного виджета необходимо:

1. Добавить следующие элементы разметки внутри LinearLayout в файле res/layout/main.xml:

*<RadioGroup*

*android:layout\_width="fill\_parent"*

*android:layout\_height="wrap\_content"*

*android:orientation="vertical" >*

*<RadioButton*

*android:id="@+id/radio\_dog"*

*android:layout\_width="wrap\_content"*

*android:layout\_height="wrap\_content"*

*android:onClick="onRadioButtonClicked"*

*android:text="Собака" />*

*<RadioButton*

*android:id="@+id/radio\_cat"*

*android:layout\_width="wrap\_content"*

*android:layout\_height="wrap\_content"*

*android:onClick="onRadioButtonClicked"*

*android:text="Кошка" />*

*<RadioButton*

*android:id="@+id/radio\_rabbit"*

*android:layout\_width="wrap\_content"*

*android:layout\_height="wrap\_content"*

*android:onClick="onRadioButtonClicked"*

*android:text="Кролик" />*

*</RadioGroup>*

2. Добавить метод onRadioButtonClicked в Активность:

*public void onRadioButtonClicked(View v) {*

*RadioButton rb = (RadioButton) v;*

*Toast.makeText(this, "Выбрано животное: " + rb.getText(),*

*Toast.LENGTH\_SHORT).show();*

*}*

**Использование виджета EditText**

Виджет EditText используется для ввода текста пользователем. Установленный для этого виджета обработчик нажатий на кнопки будет показывать введенный текст с помощью Toast. Для этого необходимо:

1. Добавить элемент EditText внутри LinearLayout в файле разметки res/layout/main.xml:

*<EditText*

*android:id="@+id/user\_name"*

*android:layout\_width="fill\_parent"*

*android:layout\_height="wrap\_content"*

*android:hint="Введите имя"/>*

2. Для обработки введенного пользователем текста добавить следующий код в конце метода onCreate. Следует обратить внимание, этот обработчик, в отличие от предыдущих, использованных нами, возвращает значение true или false. Семантика этих значений традиционна: true означает, что событие (event) обработано и больше никаких действий не требуется, false означает, что событие не обработано этим обработчиком и будет передано следующим обработчикам в цепочке. В нашем случае реагирование происходит только на нажатие (ACTION\_DOWN) кнопки Enter (KEYCODE\_ENTER):

*final EditText userName = (EditText) findViewById(R.id.user\_name);*

*userName.setOnKeyListener(new View.OnKeyListener() {*

*@Override*

*public boolean onKey(View v, int keyCode, KeyEvent event) {*

*if ((event.getAction() == KeyEvent.ACTION\_DOWN)*

*&& (keyCode == KeyEvent.KEYCODE\_ENTER)) {*

*Toast.makeText(getApplicationContext(),*

*userName.getText(), Toast.LENGTH\_SHORT).show();*

*return true;*

*}*

*return false;*

*}*

*});*

**Работа с БД**

Механизм работы с базами данных в Android позволяет хранить и обрабатывать структурированную информацию. Любое приложение может создавать свои собственные базы данных, над которыми оно будет иметь полный контроль.

В Android используется библиотека SQLite, представляющую из себя реляционную СУБД, обладающую следующими характерными особенностями: свободно распространяемая (open source), поддерживающая стандартный язык запросов и транзакции, легковесная, одноуровневая (встраиваемая), отказоустойчивая.

**Курсоры (Cursor) и ContentValues**

Запросы к СУБД возвращают объекты типа Cursor. Для экономии ресурсов используется подход, когда при извлечении данных не возвращаются копии их значений из СУБД, а создается Cursor, обеспечивающий навигацию и доступ к запрошенному набору исходных данных. Методы объекта Cursor предоставляют различные возможности навигации, назначение которых, как правило, понятно из названия:

• moveToFirst

• moveToNext

• moveToPrevious

• getCount

• getColumnIndexOrThrow

• getColumnName

• getColumnNames

• moveToPosition

• getPosition

При добавлении данных в таблицы СУБД применяются объекты класса ContentValues. Каждый такой объект содержит данные одной строки в таблице и, по сути, является ассоциативным массивом с именами столбцов и соответствующими значениями.

**Работа с СУБД SQLite**

При создании приложений, использующих СУБД, во многих случаях удобно применять инструменты, называющиеся ORM (Object-Relationship Mapping), отображающие данные из одной или нескольких таблиц на объекты используемого языка программирования. Кроме того, ORM позволяют абстрагироваться от конкретной реализации и структуры таблиц и берут на себя обязанности по взаимодействию с СУБД. К сожалению, в силу ограниченности ресурсов мобильной платформы ORM в настоящий момент в Android практически не применяется. Тем не менее, разумным подходом при разработке приложения будет инкапсуляция всех взаимодействий с СУБД в одном классе, методы которого будут предоставлять необходимые услуги остальным компонентам приложения.

Хорошей практикой является создание вспомогательного класса, берущего на себя работу с СУБД. Данный класс обычно инкапсулирует взаимодействия с базой данных, предоставляя интуитивно понятный строго типизированный способ удаления, добавления и изменения объектов. Такой Адаптер базы данных также должен обрабатывать запросы к БД и переопределять методы для открытия, закрытия и создания базы данных. Его также обычно используют как удобное место для хранения статических констант, относящихся к базе данных, таких, например, как имена таблиц и полей. Ниже показан пример каркаса для реализации подобного Адаптера:

*public class SampleDBAdapter {*

*private static final String DATABASE\_NAME = "SampleDatabase.db";*

*private static final String DATABASE\_TABLE = "SampleTable";*

*private static final int DATABASE\_VERSION = 1;*

*// Имя поля индекса для*

*public static final String KEY\_ID = "\_id";*

*// Название и номер п/п (индекс) каждого поля*

*public static final String KEY\_NAME = "name";*

*public static final int NAME\_COLUMN = 1;*

*// Для каждого поля опишите константы аналогичным образом...*

*// SQL-запрос для создания БД*

*private static final String DATABASE\_CREATE = "create table "*

*+ DATABASE\_TABLE + " (" + KEY\_ID*

*+ " integer primary key autoincrement, " + KEY\_NAME*

*+ " text not null);";*

*// Переменная для хранения объекта БД*

*private SQLiteDatabase db;*

*// Контекст приложения для*

*private final Context context;*

*// Экземпляр вспомогательного класса для открытия и обновления БД*

*private myDbHelper dbHelper;*

*// Конструктор*

*public SampleDBAdapter(Context \_context) {*

*context = \_context;*

*dbHelper = new myDbHelper(context, DATABASE\_NAME, null,*

*DATABASE\_VERSION);*

*}*

*// «Открывашка» БД*

*public SampleDBAdapter open() throws SQLException {*

*try {*

*db = dbHelper.getWritableDatabase();*

*} catch (SQLiteException e) {*

*db = dbHelper.getReadableDatabase();*

*}*

*return this;*

*}*

*// Метод для закрытия БД*

*public void close() {*

*db.close();*

*}*

*// Метод для добавления данных, возвращает индекс*

*// свежевставленного объекта*

*public long insertEntry(SampleObject \_SampleObject) {*

*// Здесь создается объект ContentValues, содержащий*

*// нужные поля и производится вставка*

*return index;*

*}*

*// Метод для удаления строки таблицы по индексу*

*public boolean removeEntry(long \_rowIndex) {*

*return db.delete(DATABASE\_TABLE, KEY\_ID + "=" + \_rowIndex, null) > 0;*

*}*

*// Метод для получения всех данных.*

*// Возвращает курсор, который можно использовать для*

*// привязки к адаптерам типа SimpleCursorAdapter*

*public Cursor getAllEntries() {*

*return db.query(DATABASE\_TABLE, new String[] { KEY\_ID, KEY\_NAME },*

*null, null, null, null, null);*

*}*

*// Возвращает экземпляр объекта по индексу*

*public SampleObject getEntry(long \_rowIndex) {*

*// Получите курсор, указывающий на нужные данные из БД*

*// и создайте новый объект, используя этими данными*

*// Если ничего не найдено, верните null*

*return objectInstance;*

*}*

*// Изменяет объект по индексу*

*// Увы, это не ORM :(*

*public boolean updateEntry(long \_rowIndex, SampleObject \_SampleObject) {*

*// Создайте объект ContentValues на основе свойств SampleObject*

*// и используйте его для обновления строки в таблице*

*return true; // Если удалось обновить, иначе false :)*

*}*

*// Вспомогательный класс для открытия и обновления БД*

*private static class myDbHelper extends SQLiteOpenHelper {*

*public myDbHelper(Context context, String name,*

*CursorFactory factory, int version) {*

*super(context, name, factory, version);*

*}*

*// Вызывается при необходимости создания БД*

*@Override*

*public void onCreate(SQLiteDatabase \_db) {*

*\_db.execSQL(DATABASE\_CREATE);*

*}*

*// Вызывается для обновления БД, когда текущая версия БД*

*// в приложении новее, чем у БД на диске*

*@Override*

*public void onUpgrade(SQLiteDatabase \_db, int \_oldVersion,*

*int \_newVersion) {*

*// Выдача сообщения в журнал, полезно при отладке*

*Log.w("TaskDBAdapter", "Upgrading from version " + \_oldVersion*

*+ " to " + \_newVersion*

*+ ", which will destroy all old data");*

*// Обновляем БД до новой версии.*

*// В простейшем случае убиваем старую БД*

*// и заново создаем новую.*

*// В реальной жизни стоит подумать о пользователях*

*// вашего приложения и их реакцию на потерю старых данных.*

*\_db.execSQL("DROP TABLE IF EXISTS " + DATABASE\_TABLE);*

*onCreate(\_db);*

*}*

*}*

*}*

**Работа с СУБД без адаптера**

При нежелании использовать класс SQLiteOpenHelper (и вообще адаптер БД) можно воспользоваться методом openOrCreateDatabase контекста приложения:

*private static final String DATABASE\_NAME = "myDatabase.db";*

*private static final String DATABASE\_TABLE = "mainTable";*

*private static final String DATABASE\_CREATE = "create table "*

*+ DATABASE\_TABLE + " ( \_id integer primary key autoincrement,"*

*+ "column\_one text not null);";*

*SQLiteDatabase myDatabase;*

*private void createDatabase() {*

*myDatabase = openOrCreateDatabase(DATABASE\_NAME, MODE\_PRIVATE, null);*

*myDatabase.execSQL(DATABASE\_CREATE);*

*}*

В этом случае можно работать с базой данных с помощью, например, метода execSQL экземпляра БД, как показано на примере выше.

**Особенности работы с БД в Android**

При работе с базами данных в Android следует избегать хранения BLOB'ов с таблицах из-за резкого падения эффективности работы.

Как показано в примере адаптера БД, для каждой таблицы рекомендуется создавать автоикрементное поле **\_id**, которое будет уникальным индексом для строк. Если же планируется делегировать доступ к БД с помощью контент-провайдеров, такое поле является обязательным.

**Выполнение запросов для доступа к данным**

Для повышения эффективности использования ресурсов мобильной платформы запросы к БД возвращают объект типа Cursor, используемый в дальнейшем для навигации и получения значений полей. Выполнение запросов осуществляется с помощью метода query экземпляра БД, параметры которого позволяют гибко управлять критериями выборки:

*// Получить поля индекса, а также первое и третье из таблицы, без дубликатов*

*String[] result\_columns = new String[] {KEY\_ID, KEY\_COL1, KEY\_COL3};*

*Cursor allRows = myDatabase.query(true, DATABASE\_TABLE, result\_columns,*

*null, null, null, null, null, null);*

*// Получить все поля для строк, где третье поле равно требуемому значению,*

*// результат отсортировать по пятому полю*

*String where = KEY\_COL3 + "=" + requiredValue;*

*String order = KEY\_COL5;*

*Cursor myResult = myDatabase.query(DATABASE\_TABLE, null, where,*

*null, null, null, order);*

**Доступ к результатам с помощью курсора**

Для получения результатов запроса необходимо установить курсор на нужную строку с помощью методов вида moveTo**Местоположение**, перечисленных выше. После этого используются типизированные методы getТип, получающие в качестве параметра индекс (номер) поля в строке. Как правило, эти значения являются статическими константами адаптера БД:

*String columnValue = myResult.getString(columnIndex);*

В примере ниже показано, как можно просуммировать все поля (типа float) из результатов выполнения запроса (и получить среднюю сумму счета):

*int KEY\_AMOUNT = 4;*

*Cursor myAccounts = myDatabase.query("my\_bank\_accounts", null, null,*

*null, null, null, null);*

*float totalAmount = 0f;*

*// Убеждаемся, что курсор непустой*

*if (myAccounts.moveToFirst()) {*

*// Проходимся по каждой строке*

*do {*

*float amount = myAccounts.getFloat(KEY\_AMOUNT);*

*totalAmount += amount;*

*} while(myAccounts.moveToNext());*

*}*

*float averageAmount = totalAmount / myAccounts.getCount();*

**Изменение данных в БД**

В классе SQLiteDatabase, содержащем методы для работы с БД, имеются методы insert, update и delete, которые инкапсулируют операторы SQL, необходимые для выполнения соответствующих действий. Кроме того, метод execSQL позволяет выполнить любой допустимый код SQL(если вы, например, захотите увеличить долю ручного труда при создании приложения). Следует помнить, что при наличии активных курсоров после любого изменения данных следует вызывать метод refreshQuery для всех курсоров, которые имеют отношение к изменяемым данным (таблицам).

**Вставка строк**

Метод insert хочет получать (кроме других параметров) объект ContentValues, содержащий значения полей вставляемой строки и возвращает значение индекса:

*ContentValues newRow = new ContentValues();*

*// Выполним для каждого нужного поля в строке*

*newRow.put(COLUMN\_NAME, columnValue);*

*db.insert(DATABASE\_TABLE, null, newRow);*

**Обновление строк**

Также используется ContentValues, содержащий подлежащие изменению поля, а для указания, какие именно строки нужно изменить, используется параметр where, имеющий стандартный для SQL вид:

*ContentValues updatedValues = new ContentValues();*

*// Повторяем для всех нужных полей*

*updatedValues.put(COLUMN\_NAME, newValue);*

*// Указываем условие*

*String where = COLUMN\_NAME + "=" + "'Бармаглот'";*

*// Обновляем*

*db.update(DATABASE\_TABLE, updatedValues, where, null);*

Метод update возвращает количество измененных строк. Если в качестве параметра where передать null, будут изменены все строки таблицы.

**Удаление строк**

Выполняет похожим на update образом:

*db.delete(DATABASE\_TABLE, KEY\_ID + "=" + rowId, null);*

**Использование SimpleCursorAdapter**

SimpleCursorAdapter позволяет привязать курсор к ListView. используя описание разметки для отображения строк и полей. Для однозначного определения того, в каких элементах разметки какие поля, получаемые через курсор, следует отображать, используются два массива: строковый с именами полей строк, и целочисленный с идентификаторами элементов разметки:

*Cursor cursor = [ . . . запрос к БД . . . ];*

*String[] fromColumns = new String[] {KEY\_NAME, KEY\_NUMBER};*

*int[] toLayoutIDs = new int[] { R.id.nameTextView, R.id.numberTextView};*

*SimpleCursorAdapter myAdapter;*

*myAdapter = new SimpleCursorAdapter(this, R.layout.item\_layout, cursor,*

*fromColumns, toLayoutIDs);*

*myListView.setAdapter(myAdapter);*

**Задание**

1. Добавить в приложение **графическую кнопку** и проследить, как изменяется изображение кнопки в разных состояниях, а также как функционирует обработчик нажатия на кнопку.
2. Добавить в приложение **чекбокс** и проследить за поведением чекбокса в разных ситуациях.
3. Добавить в приложение виджет **ToggleButton**.
4. Добавить в приложение виджет **RadioButton**:
5. Добавить в приложение виджет **EditText** и кнопку «Очистить» в разметку и написать обработчик, очищающий текстовое поле (используя метод setText виджета EditText).
6. Создать новый проект **SQLTest**, главная Активность которого будет расширять класс **ListActivity**.
7. В методе onCreate главной Активности сделать открытие или создание БД, используя в качестве основы информацию из пункта «Работа с СУБД без адаптера» и метода onUpgrade вспомогательного класса из предшествующего ему пункта. После создания БД создать таблицу с любым полем (плюс индекс), в которую добавьте 3..5 уникальных записей со строковым полем. Индекс должен инкрементироваться автоматически.
8. В этом же методе сделать запрос к таблице, получающий все строки и с помощью имеющегося курсора запишите данные в массив строк.
9. С помощью дополнительной разметки и адаптера ArrayAdapter отобразить полученные из СУБД данные на экране Активности.
10. Добавить обработчик клика на элемент списка, чтобы при клике запрашивался из БД индекс элемента с этой строкой и отображался с помощью Toast.
11. Добавить контекстное меню к элементам списка, содержащее пункт «удалить» и реализовать удаление. После удаления должны производится действия из п.п. 3 и 4 для изменения состава отображаемых данных.
12. Реализовать редактирование записей по нажатию на кнопку, а удаление через контекстное меню. Работа с СУБД должна осуществляться с использованием адаптера. Пример обработчика выбора пункта контекстного меню приведен в соответствующем разделе данного методического руководства.

**Контрольные вопросы**

1. Дайте определение понятию «Управляющий элемент»?
2. Перечислите и охарактеризуйте основные управляющие элементы в ОС Android.
3. В чём заключается отличие управляющих элементов CheckBox и RadioButton?
4. Какими способами можно создавать управляющие элементы и задавать их свойства?
5. Как происходит обработка взаимодействия с Управляющими элементами?
6. Дайте определение понятию «Курсор».
7. Перечислите и охарактеризуйте основные методы объекта Cursor.
8. Опишите механизм взаимодействия приложения с СУБД с использованием адаптера.
9. Какой метод используется для выполнения запроса к БД?
10. Перечислите и охарактеризуйте основные операции с БД?

**Содержание отчёта**

1. Ф.И.О., группа, название лабораторной работы.
2. Цель работы.
3. Описание проделанной работы.
4. Результаты выполнения лабораторной работы.
5. Выводы.

Отчет и исходные коды запаковать в архив с названием по следующему шаблону:

PMS-Lab1-«группа, аббревиатуру на латинице»-«Фамилия на латинице»

Пример: PMS-Lab1-11VS-Ivanov.zip

**Лабораторная работа №4**

«Вызов Активности с помощью явного намерения и получение результатов работы. Использование неявных Намерений. Получение данных из Намерения»

**Цель работы**

Научиться вызывать Активность с использованием явного намерения и получать результаты её работы. Научиться использовать неявные Намерения и получать данные из Намерения.

**Теоретическая часть**

**Намерения в Android**

Намерения (Intent) в Android используются в качестве механизма передачи сообщений, который может работать как внутри одного приложения, так и между приложениями.

Намерения могут применяться для:

• объявления о желании (необходимости) вашего приложения запуска какой-то Активности или Сервиса для выполнения определенных действий.

• Извещения о том, что произошло какое-то событие.

• Явного запуска указанного Сервиса или Активности.

Последний вариант является наиболее часто используемым.

**Использование Намерений для запуска Активностей.**

Чтобы запустить нужную Активность, вызывается метод startActivity(someIntent).

В конструкторе Намерения можно явно указать класс Активности, которую требуется запустить, или действие, которое нужно выполнить. Во втором случае система автоматически подберет нужную Активность, используя механизм, называемый Определением Намерений (Intent Resolution). Метод startActivity находит запускает Активность, наиболее подходящую вашему Намерению.

По окончании работы запущенной таким образом Активности запустившая ее Активность не получает никаких извещений о результатах обработки Намерения. Если требуется получать результаты, используется метод startActivityForResult.

Для явного указания того, какую Активность (конкретный класс в приложении) требуется запустить, создаются Намерения с помощью указания параметров следующих конструктора: текущий Контекст приложения и класс Активности для запуска.

*Intent intent = new Intent(MyActivity.this, MyOtherActivity.class);*

*startActivity(intent);*

**Возвращение результатов работы Активности**

Активность, запущенная с помощью метода startActivity, полностью независима от запустившей ее Активности и, соответственно, завершает свою работу, никому об этом не сообщая. В то же время, вы можете запускать Активности, «связанные» со своим «породителем». Такой способ отлично подходит для ситуаций, когда «дочерняя» Активность должна обработать ввод пользовательских данных и предоставить результаты обработки «родительской» Активности. Запускаемые таким образом (с помощью метода startActivityForResult) Активности должны быть «зарегистрированы» в Манифесте приложения.

В отличие от метода startActivity, метод startActivityForResult требует явного указания еще одного параметра – кода запроса (request code). Этот параметр используется вызывающей Активностью для определения того, какая именно дочерняя Активность завершила работу и (возможно) предоставила результаты:

*private static final int BUY\_BEER = 1;*

*Intent intent = new Intent(this, MyOtherActivity.class);*

*startActivityForResult(intent, BUY\_BEER);*

**Возвращение результатов работы**

Когда дочерняя Активность готова к завершению работы, до вызова метода finish требуется вызвать метод setResult для передачи результатов вызывающей Активности.

Метод setResult получает два параметра: код возврата и сам результат, представленный в виде Намерения.

Код возврата (result code), возвращаемый из дочерней Активности – это, обычно, либо Activity.RESULT\_OK, либо Activity.RESULT\_CANCELED. В случае, если дочерняя Активность завершит работу без вызова метода setResult, код возврата, переданный родительской Активности, будет равен Activity.RESULT\_CANCELED.

В некоторых случаях может потребоваться использовать собственные коды возврата для обработки определенных ситуаций. Метод setResult в качестве кода возврата воспринимает любое целочисленное значение.

Намерение, возвращаемое как результат, часто содержит URI, указывающий на конкретный объект данных, и/или набор дополнительных значений, записанных в свойство Намерения extras, использующихся для передачи дополнительной информации.

Пример указания обработчика нажатия на кнопку button, устанавливающего результат работы и завершающего текущую Активность:

*Button button = (Button) findViewById(R.id.ok\_button);*

*button.setOnClickListener(new View.OnClickListener() {*

*public void onClick(View v) {*

*Intent result = new Intent();*

*result.putExtra("teacher name", "Mike Varakin");*

*setResult(RESULT\_OK, result);*

*finish();*

*}*

*});*

**Обработка результатов дочерней Активности**

Когда дочерняя Активность завершает работу, в родительской Активности вызывается обработчик onActivityResult, который получает следующие параметры:

• **Request code**. Использованный при запуске дочерней активности код запроса.

• **Result code**. Код возврата.

• **Data**. Намерение, используемое для упаковки возвращаемых данных.

Пример обработчика onActivityResult:

*private static final int SELECT\_VICTIM = 1;*

*private static final int DRINK\_BEER = 2;*

*@Override*

*protected void onActivityResult(int requestCode, int resultCode, Intent data) {*

*super.onActivityResult(requestCode, resultCode, data);*

*switch (requestCode) {*

*case (SELECT\_VICTIM): {*

*if (resultCode == Activity.RESULT\_OK) {*

*String selectedVictim = data.getStringExtra("victim");*

*}*

*break;*

*}*

*case (DRINK\_BEER): {*

*if (resultCode == Activity.RESULT\_OK) {*

*// Обработать результаты заказанного действия*

*}*

*break;*

*}*

*}*

*}*

**Неявные намерения**

Неявные намерения используются для запуска Активностей для выполнения заказанных действий в условиях, когда неизвестно (или безразлично), какая именно Активность (и из какого приложения) будет использоваться.

При создании Намерения, которое в дальнейшем будет передано методу startActivity, необходимо назначить действие (action), которое нужно выполнить, и, возможно, указать URI данных, которые нужно обработать. Также можно передать дополнительную информацию с помощью свойства extras Намерения. Android сам найдет подходящую Активность (основываясь на характеристиках Намерения) и запустит ее. Пример неявного вызова телефонной «звонилки»:

*Intent intent = new Intent(Intent.ACTION\_DIAL, Uri.parse("tel:(123)456-78-90"));*

*startActivity(intent);*

Для определения того, какой именно компонент должен быть запущен для выполнения действий, указанных в Намерениях, Android использует Фильтры Намерений (Intent Filters). Используя Фильтры Намерений, приложения сообщают системе, что они могут выполнять определенные действия (action) с определенными данными (data) при определенных условиях (category) по заказу других компонентов системы.

Для регистрации компонента приложения (Активности или Сервиса) в качестве потенциального обработчика Намерений, требуется добавить элемент **<intent-filter>** в качестве дочернего элемента для нужного компонента в Манифесте приложения. У элемента **<intent-filter>** могут быть указаны следующие дочерние элементы (и соответствующие атрибуты у них):

• **<action>**. Атрибут android:name данного элемента используется для указания названия действия, которое может обслуживаться. Каждый Фильтр Намерений должен содержать не менее одного вложенного элемента <action>. Если не указать действие, ни одно Намерение не будет «проходить» через этот Фильтр. У главной Активности приложения в Манифесте должен быть указан Фильтр Намерений с действием android.intent.action.MAIN

• **<category>**. Сообщает системе, при каких обстоятельствах должно обслуживаться действие (с помощью атрибута android:name). Внутри <intent-filter> может быть указано несколько категорий. Категория android.intent.category.LAUNCHER требуется Активности, которая желает иметь «иконку» для запуска. Активности, запускаемые с помощью метода startActivity, обязаны иметь категорию android.intent.category.DEFAULT

• **<data>**. Дает возможность указать тип данных, которые может обрабатывать компонент. <intent-filter> может содержать несколько элементов <data>. В этом элементе могут использоваться следующие атрибуты:

• *android:host* : имя хоста (например, www.specialist.ru)

• *android:mimetype* : обрабатываемый тип данных (например, text/html)

• *android:path* : «путь» внутри URI (например, /course/android)

• *android:port* : порт сервера (например, 80)

• *android:scheme* : схема URI (например, http)

Пример указания Фильтра Намерений:

*<activity android:name=".MyActivity"*

*android:label="@string/app\_name" >*

*<intent-filter>*

*<action android:name="android.intent.action.SEND" />*

*<category android:name="android.intent.category.DEFAULT" />*

*<data android:mimeType="text/plain" />*

*</intent-filter>*

*</activity>*

При запуске Активности с помощью метода startActivity неявное Намерение обычно подходит только одной Активности. Если для данного Намерения подходят несколько Активностей, пользователю предлагается список вариантов.

Определение того, какие Активности подходят для Намерения, называется Intent Resolution. Его задача – определить наиболее подходящие Фильтры Намерений, принадлежащие компонентам установленных приложений. Для этого используются следующие проверки в указанном порядке:

• **Проверка действий**. После этого шага остаются только компоненты приложений, у которых в Фильтрах Намерений указано действие Намерения. В случае, если действие в Намерении отсутствует, совпадение происходит для всех Фильтров Намерений, у которых указано хотя бы одно действие.

• **Проверка категорий**. Все категории, имеющиеся у Намерения, должны присутствовать в Фильтре Намерений. Если у Намерения нет категорий, то на данном этапе ему соответствуют все Фильтры Намерений, за одним исключением, упоминавшимся выше: Активности, запускаемые с помощью метода startActivity, обязаны иметь категорию android.intent.category.DEFAULT, так как Намерению, использованному в этом случае, по умолчанию присваивается данная категория, даже если разработчик не указал ничего явно. Из этого исключения, в свою очередь, есть исключение: если у Активности присутствуют действие android.intent.action.MAIN и категория android.intent.category.LAUNCHER, ему не требуется иметь категорию android.intent.category.DEFAULT.

• **Проверка данных**. Здесь применяются следующие правила:

• Намерение, не содержащее ни URI, ни типа данных, проходит через Фильтр, если он тоже ничего перечисленного не содержит.

• Намерение, которое имеет URI, но не содержит тип данных (и тип данных невозможно определить по URI), проходит через Фильтр, если URI Намерения совпадает с URI Фильтра. Это справедливо только в случае таких URI, как mailto: или tel:, которые не ссылаются на реальные данные.

• Намерение, содержащее тип данных, но не содержащее URI подходит только для аналогичных Фильтров Намерений.

• Намерение, содержащее и тип данных, и URI (или если тип данных может быть вычислен из URI), проходит этот этап проверки, только если его тип данных присутствует в Фильтре. В этом случае URI должен совпадать, либо(!) у Намерения указан URI вида content: или file:, а у Фильтра URI не указан. То есть, предполагается, что если у компонента в Фильтре указан только тип данных, то он поддерживает URI вида content: или file:.

В случае, если после всех проверок остается несколько приложений, пользователю

предлагается выбрать приложение самому. Если подходящих приложений не найдено, в

выпустившей Намерение Активности возникает Исключение.

**Определение Намерения, вызвавшего запуск Активности**

Поскольку объекты типа Intent служат, в том числе, для передачи информации между компонентами одного или нескольких приложений, может возникнуть необходимость в работе с объектом Намерения, вызвавшим Активность к жизни.

Для получения доступа к этому объекту используется метод getIntent. Пример:

*Intent intent = getIntent();*

Имея данный объект, можно получить доступ к информации, содержащейся в нем:

• метод getAction возвращает действие Намерения

• метод getData возвращает данные Намерения (обычно URI)

• набор методов для разных типов вида get**ТИП**Extra позволяет получить доступ к типизированным значениям, хранящимся в свойстве extras Намерения.

Примеры:

*String action = intent.getAction();*

*Uri data = intent.getData();*

**Задание**

1. Создать новый проект **MetroPicker**.
2. Добавить вспомогательную Активность **ListViewActivity** для отображения и выбора станций метро, в качестве заготовки используйте результаты лабораторной работы «Использование ListView» .
3. Отредактировать файл разметки **res/layout/main.xml**: добавить кнопку выбора станции метро, присвоив идентификаторы виджетам TextView и Button для того, чтобы на них можно было ссылаться в коде.
4. Установить обработчик нажатия на кнопку в главной Активности для вызова списка станции и выбора нужной станции.
5. Написать нужный обработчик для установки выбранной станции метро в виджет TextView родительской Активности (метод setText виджета TextView позволяет установить отображаемый текст). А также обработать ситуацию, когда пользователь нажимает кнопку «Назад» (в этом случае «никакой станции не выбрано» и главная Активность должна известить об этом пользователя).
6. Протестировать созданное приложение, проверив реакцию на различные действия потенциальных пользователей.
7. Изменить проект **MetroPicker** так, чтобы для запуска Активности **ListViewActivity** использовалось неявное Намерение с действием (action) , определенным в приложении и имеющем значение "**com.example.metropicker.intent.action.PICK\_METRO\_STATION**".
8. Модифицировать методы onCreate Активностей так, чтобы с помощью Toast они показывали действие вызвавшего их Намерения.

**Контрольные вопросы**

1. Дайте определение понятию «Намерение» в ОС Android.
2. Для чего могу применяться Намерения?
3. Перечислите и охарактеризуйте методы для запуска Активности.
4. Опишите механизм возврата результата работы Активности.
5. Перечислите и охарактеризуйте параметры, передаваемые обработчику onActivityResult.
6. Дайте определение понятиям «Неявное намерение» и «Фильтр Намерений».
7. Перечислите и охарактеризуйте основные атрибуты Фильтра Намерений.
8. Перечислите и охарактеризуйте этапы проверки соответствия Намерения и Активности.
9. Как определить Намерение, вызвавшее запуск Активности?
10. Как получить доступ к данным, содержащимся в Намерении?

**Содержание отчёта**

1. Ф.И.О., группа, название лабораторной работы.
2. Цель работы.
3. Описание проделанной работы.
4. Результаты выполнения лабораторной работы.
5. Выводы.

Отчет и исходные коды запаковать в архив с названием по следующему шаблону:

PMS-Lab1-«группа, аббревиатуру на латинице»-«Фамилия на латинице»

Пример: PMS-Lab1-11VS-Ivanov.zip