Лабораторная работа №1

1.1. Отслеживание состояний Активности

1.В имеющемся проекте *HelloAndroidWorld* откройте в редакторе класс *HelloAndroidWorld.java.*

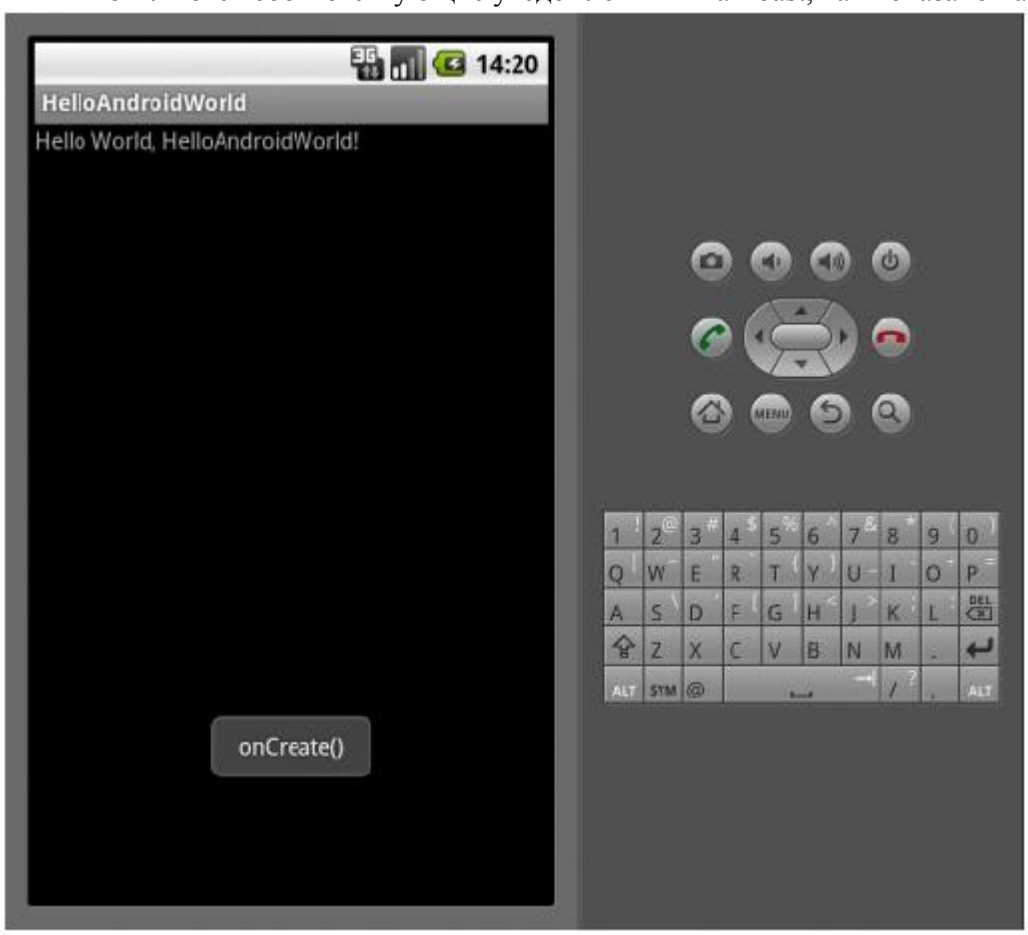
2.Переопределите методы *onPause*, *onStart*, *onRestart* для класса и внесите изменения в метод *onCreate*:

```
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();
}
}
```

Обратите внимание, что в методах, *восстанавливающих* состояние, вызов метода родительского класса производится *до* ваших действий, а в методах, состояние *сохраняющих* – *после* ваших действий.

3.Запустите приложение на исполнение в эмуляторе (Ctrl+F11, Android Project) и убедитесь, что при изменении состояния приложения HelloAndroidWorld на экране эмулятора появляются соответствующие уведомления типа Toast, как показано на снимках:



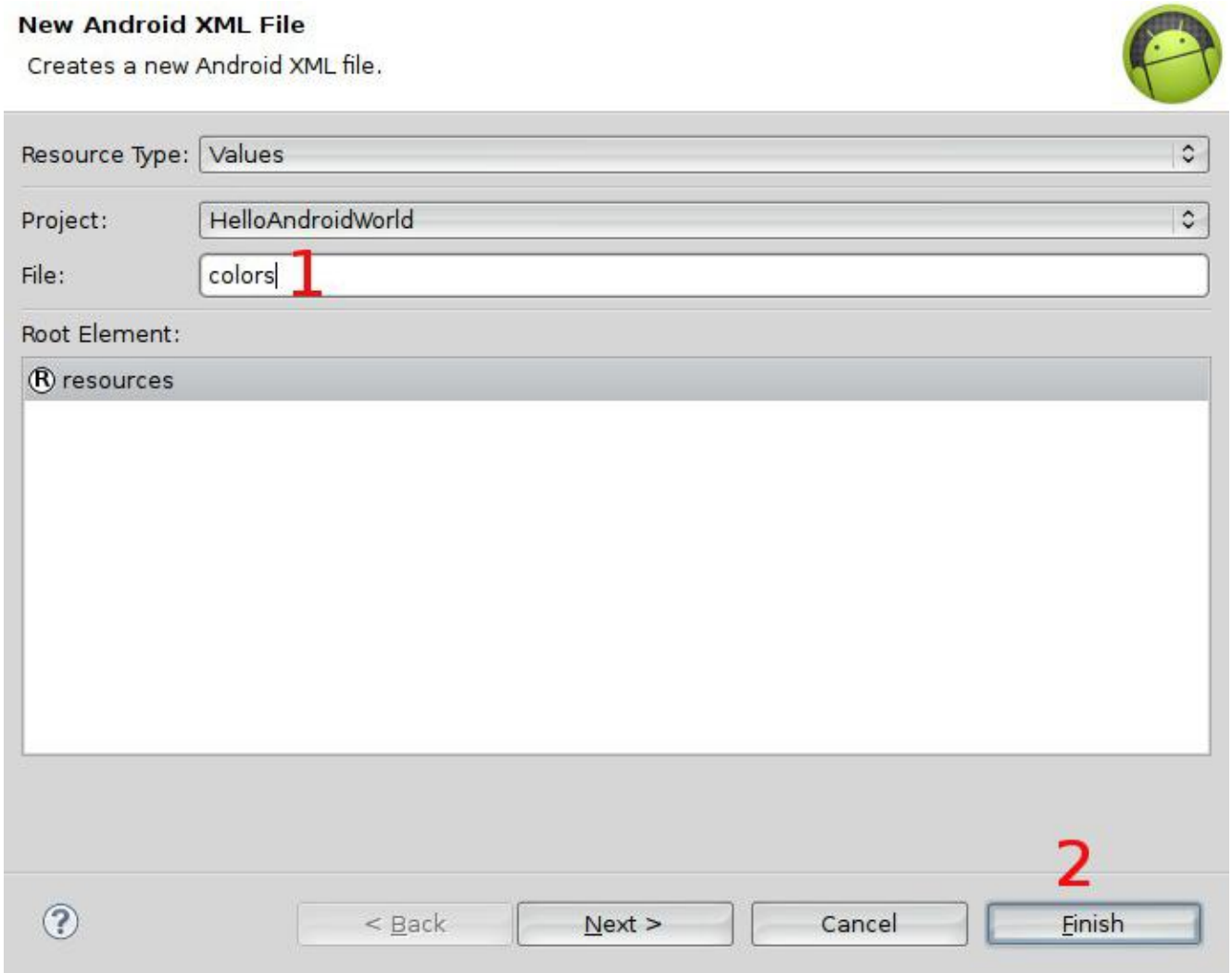
4.Поэкспериментируйте в приложением, чтобы выяснить, при каких условиях вызываются реализованные обработчики событий.


14:19
Google
SMS/MMS
Телефон
onPause()
Карты
MENU

1.2. Использование значений строк и цветов

1.В имеющемся проекте *HelloAndroidWorld* создайте файл *colors* в каталоге *res/values:*

File → New → Android XML File:



2. Отредактируйте содержимое файла **res/values/colors.xml**:

```
<?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>
```

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

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<string name="hello">Hello, Android World!</string>
<string name="app_name">HelloAndroidWorld</string>
```

```
</resources>
```

4.Внесите изменения в файл разметки **res/layout/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>
```

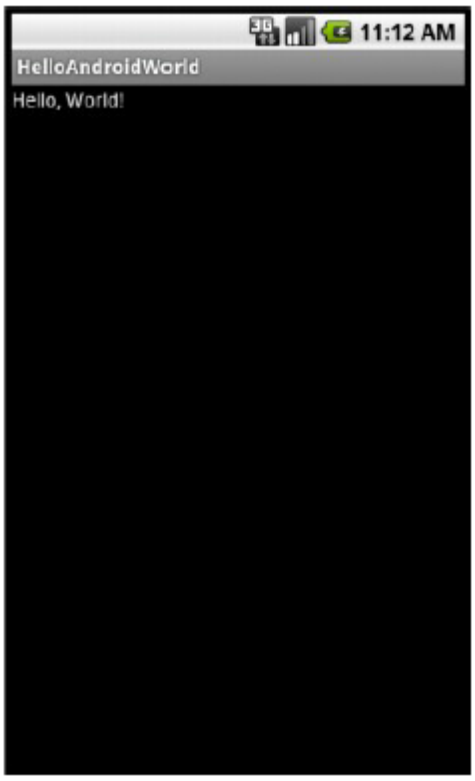
5.Сохраните все несохраненные файлы и запустите приложение:

6.Убедитесь, что внешний вид приложения ~~стал невыносимо мерзким~~ изменился в соответствии с определенными нами ресурсами.

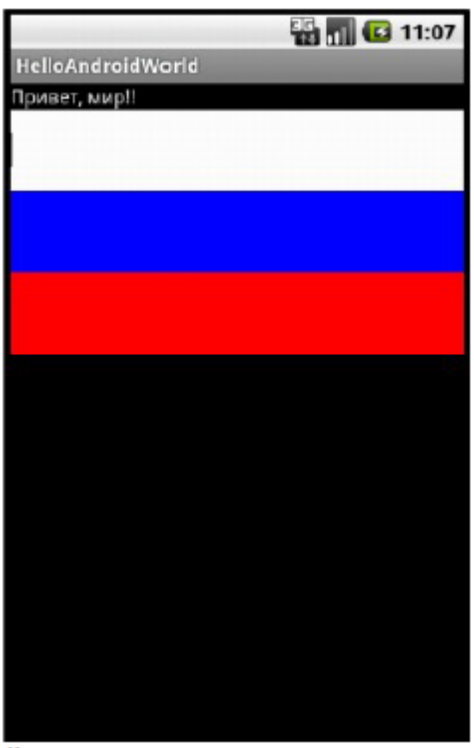
7.Поэкспериментируйте со значениями цветов, прозрачностью и HTML-тэгами в строковых константах для изменения внешнего вида приложения.

1.3. Локализация приложения

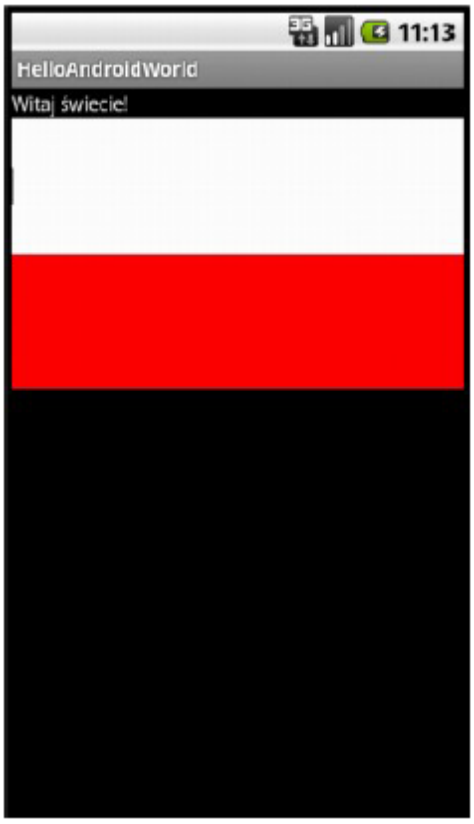
1.Измените ресурсы приложения *HelloAndroidWorld* так, чтобы оно выглядело следующим образом (на виртуальном устройстве должен быть установлен английский язык):



2.Добавьте нужные подкаталоги и ресурсы в каталог **res** проекта *HelloAndroidWorld*, чтобы при настройке русского языка приложение меняло свой вид на следующий:



3.Проделайте те же действия для польского языка:



4.Восстановите языковые настройки на виртуальном устройстве.

Лабораторная работа №2

2.1. «Использование анимации»

1.Создайте новый проект **AnimSample**.

2.В каталоге **res** создайте каталог **anim**, а в нем файл с именем **ship_anim.xml**, описывающий анимацию. Отредактируйте файл, чтобы он имел следующее содержимое:

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android" >
<rotate
android:duration="3333"
android:fromDegrees="0"
android:pivotX="50%"
android:pivotY="50%"
android:repeatCount="infinite"
android:repeatMode="reverse"
android:toDegrees="1080" />
<translate
android:duration="1900"
android:fromXDelta="-50%p"
android:repeatCount="infinite"
android:repeatMode="reverse"
android:toXDelta="50%p" />
<translate
android:duration="1300"
android:fromYDelta="-50%p"
android:repeatCount="infinite"
android:repeatMode="reverse"
android:startOffset="123"
android:toYDelta="50%p" />
<alpha
android:duration="500"
android:fromAlpha="1.0"
android:repeatCount="infinite"
android:repeatMode="reverse"
android:toAlpha="0.3" />
<scale
android:duration="10000"
android:fromXScale="0.0"
android:fromYScale="0.0"
android:pivotX="50%"
android:pivotY="50%"
android:repeatCount="infinite"
android:repeatMode="reverse"
android:toXScale="2.5"
android:toYScale="2.5" />
</set>
```

3.Добавьте рисунок, к которому будет применяться анимация (файл **lander_plain.png**), в каталог **res/drawable-mdpi**.

4.В файле разметки **res/layout/main.xml** замените элемент **TextView** на **ImageView** со следующими атрибутами:

```
<ImageView
android:id="@+id/shipView"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:src="@drawable/lander_plain" />
```

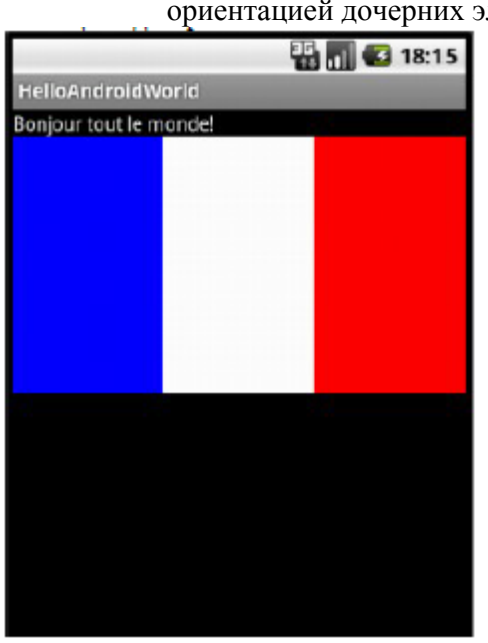
5.В конце метода **onCreate** Активности (она в этом проекте одна) добавьте следующие строки:

```
ImageView ship = (ImageView) findViewById(R.id.shipView);
Animation shipAnim = AnimationUtils.loadAnimation(this,
R.anim.ship_anim);
ship.startAnimation(shipAnim);
```

6.Запустите проект.

2.2. «Использование LinearLayout»

1.Продолжите локализацию приложения HelloAndroidWorld, теперь для французского языка. Для рисования флага используйте вложенный LinearLayout с вертикальной ориентацией дочерних элементов:



2.Обратите внимание, что при размещении элементов *внутри вложенного LinearLayout* удобно указывать атрибут **android:layout_weight="1"**, в этом случае дочерние виджеты будут размещены по горизонтали равномерно.

3.После получения нужного результата верните стандартные языковые настройки в виртуальном устройстве.

2.3. «Использование RelativeLayout»

RelativeLayout является очень полезным вариантом разметки, позволяющим создавать пользовательский интерфейс без избыточного количества вложенных элементов *ViewGroup*.

1.Создайте новый проект с именем **RelativeLayoutSample**.

2.Добавьте нужные строки в файл **res/values/strings.xml** и удалите строку с именем **hello**:

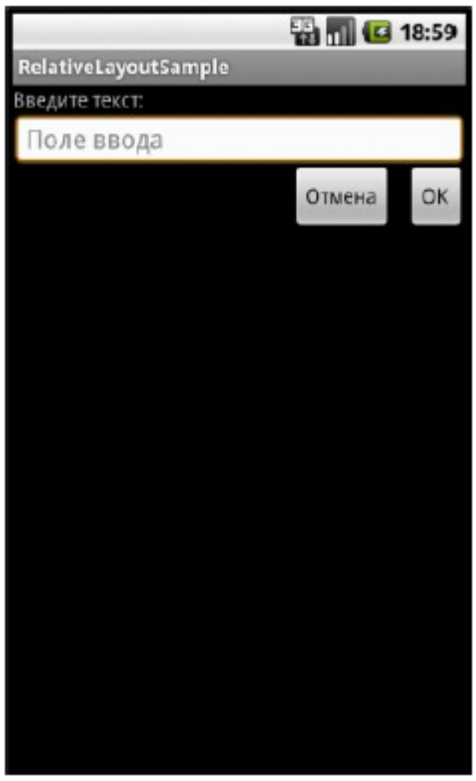
```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<string name="label_text">Введите текст:</string>
<string name="entry_hint">Поле ввода</string>
<string name="app_name">RelativeLayoutSample</string>
</resources>
```

3.Отредактируйте файл разметки **res/layout/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>
```

4.Запустите приложение:



5. 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" >

    . . .
```

6.Запустите приложение с новой темой:

7.Мы использовали для текста кнопок в разметке стандартные строковые значения, которые предоставляет Android. Поэкспериментируйте с настройками языка в виртуальном устройстве и обратите внимание, как при запуске приложения меняются надписи на *кнопках*. Очевидно, что использование стандартных строковых значений позволяет минимизировать затраты на локализацию приложений.

2.4. «Использование TabWidget»

«Табулированная» разметка позволяет создавать UI, содержание вкладки. В этом случае используются такие виджеты, как *TabHost* и *TabWidget*.

TabHost является корневым узлом в разметке, содержащим *TabWidget* для отображения «вкладок», и FrameLayout для соответствующего им контента.

Отображение контента вкладок можно реализовать двумя способами:

•описав Представление (View) для содержимого каждой вкладки внутри *одной и той же* Активности;

•используя вкладки для переключения между *различными* Активностями.

Выбор конкретной реализации зависит от потребностей разработчика, но обычно более предпочтительным является второй вариант, так в этом случае разные вкладки обрабатываются разными Активностями, а не одной (достаточно громоздкой), что позволяет делать код более ясным и управляемым. В данной лабораторной работе мы используем вариант реализации с независимыми Активностями, поэтому для реализации трех вкладок нам потребуются четыре Активности (три для вкладок и одна «главная).

1.Создайте новый проект с именем **TabWidgetSample**.

2.Опишите нужные строки в файле **res/values/strings.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<string name="app_name">TabWidgetSample</string>
<string name="tab1_indicator">Students</string>
<string name="tab2_indicator">Teachers</string>
<string name="tab3_indicator">Classes</string>
<string name="tab1_content">This is Students tab</string>
<string name="tab2_content">This is Teachers tab</string>
<string name="tab3_content">This is Classes tab</string>
</resources>
```

3.Отредактируйте файл **res/layout/main.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<TabHost xmlns:android="http://schemas.android.com/apk/res/android"
android:id="@android:id/tabhost"
android:layout_width="fill_parent"
android:layout_height="fill_parent">
<LinearLayout
android:orientation="vertical"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:padding="5dp">
```

```
<TabWidget
android:id="@android:id/tabs"
android:layout_width="fill_parent"
android:layout_height="wrap_content" />
<FrameLayout
android:id="@android:id/tabcontent"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:padding="5dp" />
</LinearLayout>
</TabHost>
```

4.Создайте три Активности с именами **StudentsActivity**, **TeachersActivity** и **ClassesActivity**.

5.В каждой из созданных Активностей переопределите метод **onCreate** следующим образом (внеся соответствующие изменения в имя ресурса *tab**X**_content*):

```
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
Resources res = getResources();
String contentText = res.getString(R.string.tab2_content);
TextView textView = new TextView(this);

        textView.setText(contentText);
setContentView(textView);
}
```

Обратите внимание на то, что в каждой Активности используются *собственные* строковые ресурсы.

6.Замените базовый класс для **TabWidgetSampleActivity** с **Activity** на **TabActivity** и переопределите метод **onCreate** следующим образом:

```
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.main);
Resources res = getResources();
String tab1Indicator = res.getString(R.string.tab1_indicator);
String tab2Indicator = res.getString(R.string.tab2_indicator);
String tab3Indicator = res.getString(R.string.tab3_indicator);
TabHost tabHost = getTabHost();
TabHost.TabSpec spec;
Intent intent;
intent = new Intent().setClass(this, StudentsActivity.class);
spec = tabHost.newTabSpec("students").setIndicator(tab1Indicator)
.setContent(intent);
tabHost.addTab(spec);
intent = new Intent().setClass(this, TeachersActivity.class);
spec = tabHost.newTabSpec("teachers").setIndicator(tab2Indicator)
.setContent(intent);
tabHost.addTab(spec);
intent = new Intent().setClass(this, ClassesActivity.class);
spec = tabHost.newTabSpec("class").setIndicator(tab3Indicator)
.setContent(intent);
tabHost.addTab(spec);
```

```
tabHost.setCurrentTab(1);
}
```

В этом методе мы используем для запуска нужных Активностей так называемые *явные* **Намерения** (**Intent**). Намерения и их использование будут рассмотрены позже.

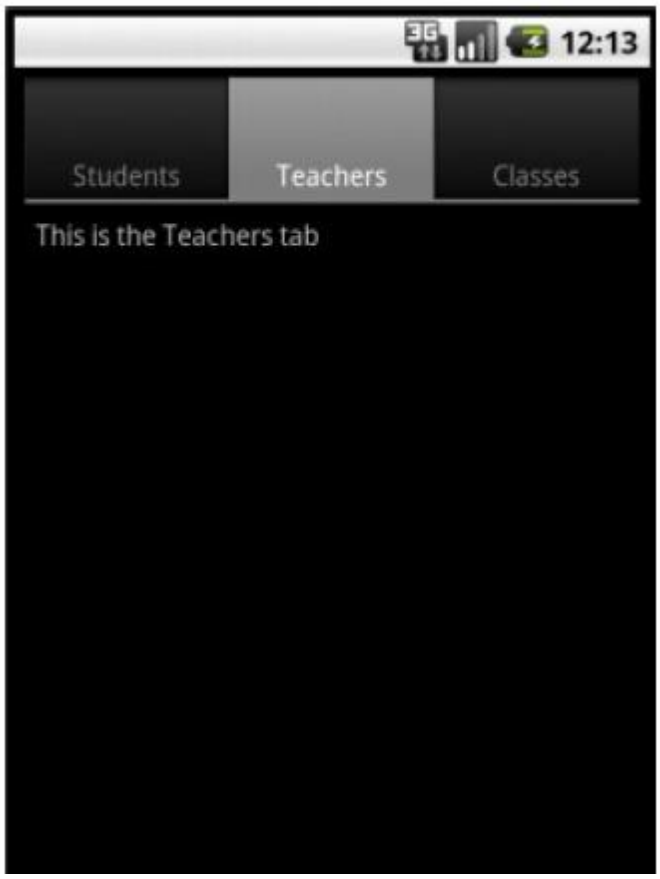
7.При попытке запуска проекта на выполнение произойдет аварийное завершение работы приложения, так как Активности, созданные для отображения контента вкладок неизвестны системе, потому что не описаны в Манифесте приложения. Требуется отредактировать файл **AndroidManifest.xml** так, чтобы все используемые Активности были в нем описаны. Кроме того, для улучшения внешнего вида интерфейса имеет смысл изменить используемую тему оформления на такую, где нет *строки заголовка* : Узел <application> должен выглядеть следующим образом:

```
<application
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@android:style/Theme.NoTitleBar" >
    <activity
        android:name=".TabWidgetSampleActivity"
        android:label="@string/app_name" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
    <activity android:name=".StudentsActivity" />
    <activity android:name=".TeachersActivity" />
    <activity android:name=".ClassesActivity" />
</application>
```

```
<application
android:icon="@drawable/ic_launcher"
android:label="@string/app_name"
android:theme="@android:style/Theme.NoTitleBar" >
<activity
android:name=".TabWidgetSampleActivity"
android:label="@string/app_name" >
<intent-filter>
<action android:name="android.intent.action.MAIN" />
<category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>
<activity android:name=".StudentsActivity" />
<activity android:name=".TeachersActivity" />
<activity android:name=".ClassesActivity" />
</application>
```

8.Запустите приложение и поэкспериментируйте с вкладками:



9.Локализуйте приложение с помощью индивидуальных для разных языков строковых ресурсов.

Лабораторная работа №3

3.1. «Использование WebView»

В данной лабораторной работе рассматривается использование виджета web-браузера и применяется итеративный подход к созданию приложения.

1.Создайте новый проект с именем **WebViewSample**.

2.Отредактируйте файл **res/layout/main.xml**:

```
<?xml version="1.0" encoding="utf-8"?>
<WebView xmlns:android="http://schemas.android.com/apk/res/android"
android:id="@+id/webview"
android:layout_width="fill_parent"
android:layout_height="fill_parent" />
```

3.Добавьте в конец метода **onCreate** Активности **WebViewSampleActivity** следующие строки:

```
WebView webView = (WebView) findViewById(R.id.webview);
webView.getSettings().setJavaScriptEnabled(true);
webView.loadUrl("http://www.ya.ru");
```

4.Запустите приложение:

5.Очевидно, что ~~в супе чего-то не хватает~~ у приложения отсутствуют полномочия на доступ к сети. Добавим нужные полномочия в Манифест приложения, добавив элемент <*uses-permission*> внутри корневого узла <*mainfest*>:

<uses-permission android:name=*"android.permission.INTERNET"*/>

6.Запустим проект (нижняя часть картинки отрезана):

7.Продолжим улучшать приложение. Для увеличения полезной площади экрана запретим показ заголовка, для этого укажем соответствующую тему в файле Манифеста:

<activity
android:name=".WebViewSampleActivity"
android:label="@string/app_name"
android:theme="@android:style/Theme.NoTitleBar" >

8.Запустим проект и убедимся том, что (первая) цель достигнута.

9.В настоящий момент ссылки, ведущие за пределы сайта www.ya.ru обслуживаются стандартным web-браузером, а не нашим WebView, так как оно не в состоянии обработать эти запросы и виджет webView автоматически посылает системе соответствующее *Намерение* (*Intent*), обрабатываемое стандартным браузером. У этой проблемы есть два решения:

•Добавить в Манифест нужный *Фильтр Намерений* (*Intent Filter*)

•Переопределить внутри нашей Активности класс *WebViewClient*, чтобы приложение могло обрабатывать свои *собственные запросы* на отображение web-ресурсов с помощью имеющегося виджета WebView.

Второй вариант является более приемлемым еще и потому, что в этом случае не будет рассматриваться системой как альтернативный web-браузер, что произошло бы в случае добавления *Фильтра Намерений* в Манифест.

10.Вынесем объект *webView* из метода *onCreate* и сделаем его членом класса, после чего добавим внутри Активности новый класс *WebViewSampleClient*, расширяющий класс *WebViewClient* , после чего установим свой обработчик запросов на отображение web-ресурсов:

```
public class WebViewSampleActivity extends Activity {
WebView webView;
@Override
public void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);
setContentView(R.layout.main);
webView = (WebView) findViewById(R.id.webview);
webView.getSettings().setJavaScriptEnabled(true);
webView.loadUrl("http://www.ya.ru");
webView.setWebViewClient(new WebViewSampleClient());
}
private class WebViewSampleClient extends WebViewClient {
@Override
public boolean shouldOverrideUrlLoading(WebView view, String url) {
view.loadUrl(url);
return true;
}
}
}
```

Запустим приложение и убедимся, что оставшийся серьезный недостаток – «неправильная» обработка нажатия кнопки «назад» устройства, приложение при этом заканчивает свою работу. Решение этой проблемы простое и прямолинейное: сделаем свой обработчик событий нажатия на кнопки, в котором будет реализовано только одно действие: если была нажата кнопка «назад», виджет WebView получит команду вернуться на предыдущую страницу (если у него есть такая возможность). Переопределим метод **onKeyDown** в Активности:

```
@Override
public boolean onKeyDown(int keyCode, KeyEvent event) {
if ((keyCode == KeyEvent.KEYCODE_BACK) && webView.canGoBack()) {
webView.goBack();
return true;
}
return super.onKeyDown(keyCode, event);
}
```

13.Запустим приложение и убедимся, что цель достигнута.

Лабораторная работа №4

4.1. «Использование ListView»

1.Создайте новый Android проект **ListViewSample**.

2.В каталоге **res/values** создайте файл **arrays.xml** со следующим содержимым:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<string-array name="stations">
<item>Вокзальна</item>
<item>Університет</item>
…

<item>Хрещатик</item>
</string-array>
</resources>
```

3.В каталоге res/layout создайте 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>
```

4.Модифицируйте метод onCreate вашей 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();
}
```

5.Измените базовый класс Активности с **Activity** на **ListActivity**.

6.Запустите приложение.

7.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*. Добавьте следующий код в *нужное место*:

```
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();
}
});
```

8.Запустите приложение и «покликайте» по станциям метро.

Лабораторная работа №5

«Использование управляющих элементов в пользовательском интерфейсе»

Цель лабораторной работы – научиться использовать в интерфейсе пользователя различные управляющие элементы: кнопки с изображениями, радиокнопки, чекбоксы и пр.

5.1. Подготовка

1.Создайте новый проект **ControlsSample**.

2.Отредактируйте файл **res/layout/main.xml** так, чтобы остался только корневой элемент **LinearLayout**. В него в дальнейшем будут добавляться необходимые дочерние элементы:

```
<?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" >
</LinearLayout>
```

5.2. 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.

1.Скопируйте нужные изображения кнопки в каталог **res/drawable-mdpi**, для обновления списка содержимого каталога в **Eclipse** можно использовать кнопку **F5**.

2.В этом же каталоге создайте 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.Добавьте 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.Обратите внимание на атрибут android:onClick="onButtonClicked", указывающий, какой метод из Активности будет использоваться как обработчик нажатия на данную кнопку. Добавьте этот метод в Активность:

```
public void onButtonClicked(View v) {
Toast.makeText(this, "Кнопка нажата", Toast.LENGTH_SHORT).show();
}
```

5.Запустите приложение и посмотрите, как изменяется изображение кнопки в разных состояниях, а также как функционирует обработчик нажатия на кнопку.

5.3. Использование виджета CheckBox

1.Добавьте 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. Добавьте 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();
}
}
```

3.Запустите приложение и посмотрите на поведение чекбокса в разных ситуациях.

5.4. Использование виджета ToggleButton

Данный виджет хорошо подходит в качестве альтернативы радиокнопкам и чекбоксам, когда требуется переключаться между двумя взаимоисключающими состояниями, например, *Включено/Выключено*.

1.Добавьте 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. Добавьте 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();
}
}
```

3.Запустите приложение и проверьте его функционирование.

5.5. Использование виджета RadioButton

Радиокнопки используются для выбора между различными взаимоисключающими

вариантами. Для создания *группы радиокнопок* используется элемент *RadioGroup*, внутри которого располагаются элементы *RadioButton*.

1.Добавьте 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="Собачка" />
<RadioButton
android:id="@+id/radio_cat"
android:layout_width="wrap_content"
```

```
android:layout_height="wrap_content"
android:onClick="onRadioButtonClicked"
android:text="Кошечка" />
<RadioButton
android:id="@+id/radio_rabbit"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:onClick="onRadioButtonClicked"
android:text="Кролик" />
</RadioGroup>
```

2.Добавьте метод **onRadioButtonClicked** в Активность:

```
public void onRadioButtonClicked(View v) {
RadioButton rb = (RadioButton) v;
Toast.makeText(this, "Выбрано животное: " + rb.getText(),
Toast.LENGTH_SHORT).show();
}
```

3.Проверьте работу приложения.

5.6. 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*.

1.Добавьте 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"/>
```

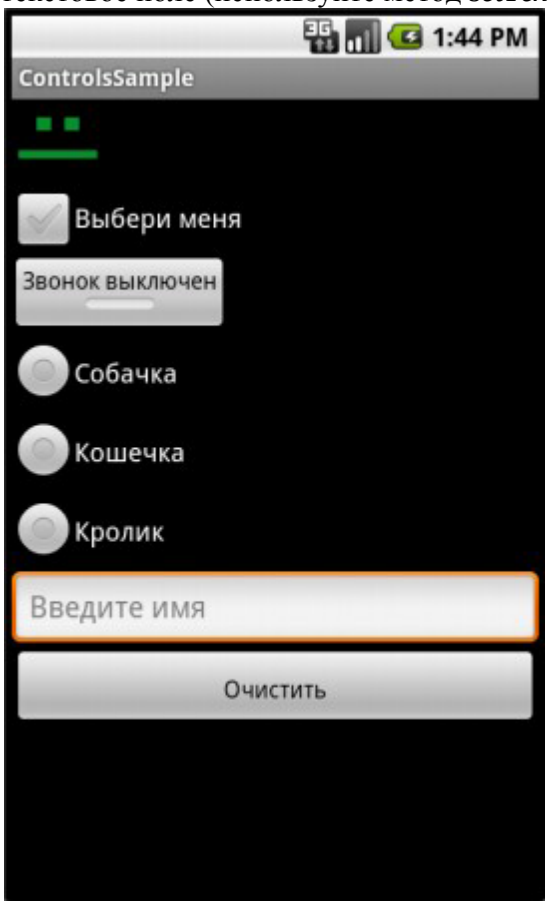
2.Для обработки введенного пользователем текста добавьте следующий код в конце метода **onCreate**. Обратите внимание, этот обработчик, в отличие от предыдущих, использованных нами, *возвращает* значение **true** или **false**. Семантика этих значений традиционна: *true* означает, что событие (*event*) 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;
}
});
```

3.Запустите приложение и проверьте его работу.

4.Добавьте кнопку **«Очистить»** в разметку и напишите обработчик, очищающий текстовое поле (используйте метод *setText* виджета *EditText*):



5.Проверьте работу приложения.

Лабораторная работа №6

6.1. «Вызов Активности с помощью явного намерения и получение результатов работы»

1.Создайте новый проект **MetroPicker**.

2.Добавьте вспомогательную Активность **ListViewActivity** для отображения и выбора станций метро, в качестве заготовки используйте результаты лабораторной работы **«Использование ListView» .**

3.Отредактируйте файл разметки **res/layout/main.xml**: добавьте кнопку выбора станции метро, присвоив идентификаторы виджетам *TextView* и *Button* для того, чтобы на них можно было ссылаться в коде.

4.Установите обработчик нажатия на кнопку в главной Активности для вызова списка станции и выбора нужной станции.

5.Напишите нужный обработчик для установки выбранной станции метро в виджет *TextView* родительской Активности (метод *setText* виджета *TextView* позволяет установить отображаемый текст). Не забудьте обработать ситуацию, когда пользователь нажимает кнопку «Назад» (в этом случае «никакой станции не выбрано» и главная Активность должна известить об этом пользователя).

6.Убедитесь в работоспособности созданного приложения, проверив реакцию различные действия потенциальных пользователей.

6.2. «Использование неявных Намерений»

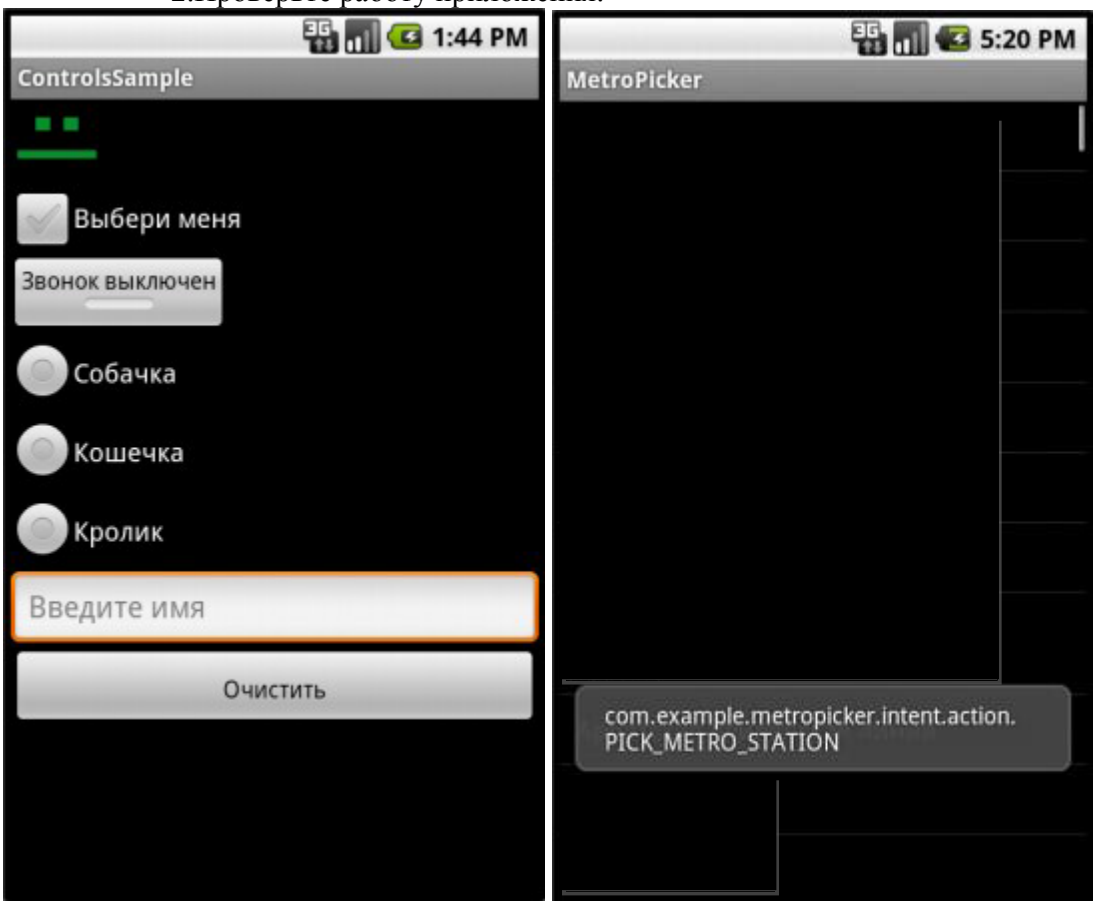
1.Измените проект **MetroPicker** так, чтобы для запуска Активности **ListViewActivity** использовалось неявное Намерение с действием (*action*) , определенным в вашем приложении и имеющем значение "**com.example.metropicker.intent.action.PICK_METRO_STATION**".

2.Проверьте работу приложения.

6.3. «Получение данных из Намерения»

1. Модифицируйте методы onCreate двух ваших Активностей из предыдущей лабораторной работы так, чтобы с помощью Toast они показывали действие вызвавшего их Намерения .

2.Проверьте работу приложения:



Лабораторная работа №7

7.1. «Использование SharedPreferences для сохранения состояния»

1.Модифицируйте методы **onCreate** и **onActivityResult** проекта **MetroPicker** для сохранения выбранной станции метро между запусками приложения.

2.Проверьте работоспособность приложения.

7.2. «Использование SharedPreferences для сохранения настроек»

1.Модифицируйте проект **ControlsSample** так, чтобы состояние управляющих элементов сохранялось и восстанавливалось между запусками приложения.

2.Проверьте работоспособность приложения.

Лабораторная работа №8

8.1.«Создание и использование меню»

Модифицируйте проект **MetroPicker** следующим образом:

1.Добавьте главное меню в Активность, отображающую список станций метро. В меню должен быть один пункт: «вернуться». Меню создайте динамически в коде, без использования строковых ресурсов.

2.Динамически создайте контекстное меню для Представления *TextView,* отображающего выбранную станцию метро главной Активности. Выбор пункта меню должен сбрасывать выбранную станцию.

3.Для главной Активности создайте основное меню из двух пунктов: «сбросить» и «выйти». Реализуйте нужные функции при выборе этих пунктов.

4.Повторите реализацию п.п. 1, 2 и 3 с помощью ресурсов, описывающих меню.

8.2. «Работа с SQLite без класса-адаптера»

Цель данной лабораторной работы – освоить взаимодействие с СУБД SQLite без применения специальных классов-адаптеров.

1.Создайте новый проект **SQLTest**, главная Активность которого будет расширять класс **ListActivity.**

2.В методе *onCreate* главной Активности сделайте 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 создайте таблицу с любым полем (плюс индекс), в которую добавьте 3..5 уникальных записей со строковым полем. Индекс должен инкрементироваться автоматически.

3.В этом же методе сделайте запрос к таблице, получающий все строки и с помощью имеющегося курсора запишите данные в массив строк.

4.С помощью дополнительной разметки и адаптера *ArrayAdapter* отобразите полученные из СУБД данные на экране Активности.

5.Добавьте обработчик клика на элемент списка, чтобы при клике запрашивался из БД индекс элемента с этой строкой и отображался с помощью Toast.

6.Добавьте контекстное меню к элементам списка, содержащее пункт «удалить» и реализуйте удаление. После удаления должны производится действия из п.п. 3 и 4 для изменения состава отображаемых данных.

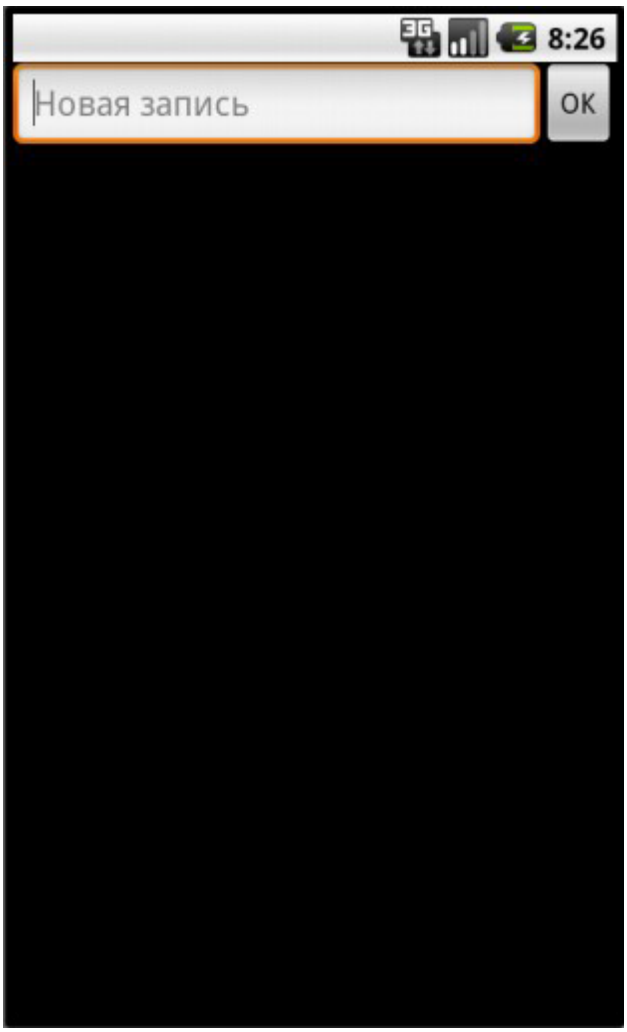
8.3. «Работа с SQLite с классом-адаптером»

Целью работы является создание простого приложения, позволяющего создавать, хранить и удалять короткие заметки.

1.Создайте новый проект **NotesSample.**

2.Для простоты использования все действия с записями будут производиться в рамках одной Активности, поэтому используйте максимально упрощенный интерфейс, как показано на следующей странице. Для реализации можно воспользоваться такой разметкой в файле **res/layout/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" >
<RelativeLayout
android:layout_width="fill_parent"
android:layout_height="wrap_content" >
<Button
android:id="@+id/save_button"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_alignParentRight="true"
android:text="@android:string/ok" />
<EditText
android:id="@+id/edit_text"
android:layout_width="fill_parent"
android:layout_toLeftOf="@id/save_button"
android:layout_height="wrap_content"
android:hint = "Новая запись" />
</RelativeLayout>
<ListView
android:id="@+id/myListView"
android:layout_width="fill_parent"
android:layout_height="wrap_content" />
</LinearLayout>
```



3.Реализуйте сохранение 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.

Лабораторная работа №9

9.1.«Получение списка контактов»

Для чтения информации о контактах используется контент-провайдер *ContactsContract*, точнее, один из его подклассов. Для этой лабораторной работы воспользуемся провайдером **ContactsContract.Contacts**. Для чтения контактов приложению требуются полномочия **READ_CONTACTS**.

1.Добавьте несколько контактов в эмуляторе (поскольку требуется только отображаемое имя контакта, остальные поля можно не заполнять :).

2.Создайте новый проект **ContactsSample**.

3.Выведите имена всех контактов (с помощью ListView), используя для получения информации URI ContactsContract.Contacts.*CONTENT_URI*. Необходимое имя поля для привязки адаптера найдите среди статических констант класса ContactsContract.Contacts.

9.2. «Использование сетевых сервисов»

Целью данной работы является создание простого приложения, получающего курсы иностранных валют по отношению к рублю с сайта ЦБ РФ в формате XML и отображающего данные в виде списка (*ListView*). Для получения данных будет использоваться URL **http://www.cbr.ru/scripts/XML_daily.asp**

Ответ сервера выглядит примерно так:

```
<ValCurs Date="04.04.2012" name="Foreign Currency Market">
<Valute ID="R01010">
<NumCode>036</NumCode>
<CharCode>AUD</CharCode>
<Nominal>1</Nominal>
<Name>Австралийский доллар</Name>
<Value>30,4632</Value>
</Valute>
. . . . . .
</ValCurs>
```

Для каждой валюты (элемент *Valute*) потребуется извлечь значения дочерних элементов *CharCode*, *Nominal*, *Name* и *Value*. Значение атрибута *Date* корневого элемента (*ValCurs*) будет использоваться для изменения заголовка приложения.

1.Создайте новый проект **CurrencyRates**. Главная (и единственная) Активность с таким же именем (CurrencyRates) должна расширять класс **ListActivity**.

2.Отредактируйте Манифест приложения, добавив в него необходимые полномочия и

тему для Активности (android:theme="@android:style/***Theme.Light***").

3.Файл строковых ресурсов **strings.xml** (в каталоге **res/values**) отредактируйте следующим образом:

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



<resources>
<string name="app_name">Курсы ЦБ РФ</string>
<string name="rates_url">http://www.cbr.ru/scripts/XML_daily.asp</string>
</resources>
```

4.Для отображения информации требуется создать разметку для элементов списка. В каталоге **res/layout** создайте файл **item_view.xml** со следующим содержимым:

```
<?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="horizontal" >
<TextView
android:id="@+id/charCodeView"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:background="#FF8"
android:minWidth="45sp"
android:padding="4dp"
android:textColor="#00F"
android:textStyle="bold"
android:gravity="center"
android:shadowDx="8"
android:shadowDy="8"
android:shadowColor="#000"
android:shadowRadius="8"/>
<TextView
android:id="@+id/valueView"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:textColor="#008"
android:background="#FFE"
android:minEms="3"
android:padding="3dp" />
<TextView
android:id="@+id/nominalView"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:padding="3dp" />
<TextView
android:id="@+id/nameView"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:ellipsize="marquee"
android:singleLine="true" />
```

```
</LinearLayout>
```

5.Вся логика приложения будет сосредоточена в классе *CurrencyRates*, поэтому остальные изменения будут касаться только этого класса. Добавьте необходимые

константы:

```
private final static String KEY_CHAR_CODE = "CharCode";
private final static String KEY_VALUE = "Value";
private final static String KEY_NOMINAL = "Nominal";
private final static String KEY_NAME = "Name";
```

6.Тело метода onCreate должно содержать только две строки:

```
super.onCreate(savedInstanceState);
populate();
```

Поскольку *CurrencyRates* является наследником *ListActivity*, вызов *setContentView* не требуется. Метод *populate* будет наполнять *ListView* содержимым с помощью адаптера (*SimpleAdapter*), заполнив его данными, полученными от метода *getData*.

7.Добавьте метод **populate**, в котором создается и настраивается адаптер:

```
private void populate() {
ArrayList<Map<String, String>> data = getData();
String[] from = { KEY_CHAR_CODE, KEY_VALUE, KEY_NOMINAL, KEY_NAME };
int[] to = { R.id.charCodeView, R.id.valueView, R.id.nominalView,
R.id.nameView };
SimpleAdapter sa = new SimpleAdapter(this, data, R.layout.item_view,
from, to);
setListAdapter(sa);
}
```

8.Добавьте метод **getData**. Именно в нем будет создаваться и обрабатываться соединение с сервером, а также анализироваться XML-данные и заполняться список, который будет отображаться адаптером. Метод **getData** объемнее остальных, но ничего сложного в нем нет (стоит отметить, что интерфейсы *Document*, *Element* и *NodeList* должны импортироваться из пакета *org.w3c.dom*):

```
private ArrayList<Map<String, String>> getData() {
ArrayList<Map<String, String>> list =
new ArrayList<Map<String, String>>();
Map<String, String> m;
try {
// Создаем объект URL
URL url = new URL(getString(R.string.rates_url));
// Соединяемся
HttpURLConnection httpConnection =
(HttpURLConnection) url.openConnection();
// Получаем от сервера код ответа
int responseCode = httpConnection.getResponseCode();
// Если код ответа хороший, парсим поток(ответ сервера),
// устанавливаем дату в заголовке приложения и
// заполняем list нужными Map'ами
if (responseCode == HttpURLConnection.HTTP_OK) {
InputStream in = httpConnection.getInputStream();
```

```
DocumentBuilderFactory dbf = DocumentBuilderFactory
.newInstance();

DocumentBuilder db = dbf.newDocumentBuilder();
Document dom = db.parse(in);
Element docElement = dom.getDocumentElement();
String date = docElement.getAttribute("Date");
setTitle(getTitle() + " на " + date);
NodeList nodeList = docElement
.getElementsByTagName("Valute");
int count = nodeList.getLength();
if (nodeList != null && count > 0) {
for (int i = 0; i < count; i++) {
Element entry = (Element) nodeList.item(i);
m = new HashMap<String, String>();
String charCode = entry
.getElementsByTagName(KEY_CHAR_CODE)
.item(0).getFirstChild().getNodeValue();
String value = entry
.getElementsByTagName(KEY_VALUE)
.item(0).getFirstChild().getNodeValue();
String nominal = "за " + entry
.getElementsByTagName(KEY_NOMINAL)
.item(0).getFirstChild().getNodeValue();
String name = entry
.getElementsByTagName(KEY_NAME)
.item(0).getFirstChild().getNodeValue();
m.put(KEY_CHAR_CODE, charCode);
m.put(KEY_VALUE, value);
m.put(KEY_NOMINAL, nominal);
m.put(KEY_NAME, name);
list.add(m);
}
}
} else {
// Сделать извещения об ошибках, если код ответа
// нехороший
}
} catch (MalformedURLException e) {
e.printStackTrace();
} catch (IOException e) {
e.printStackTrace();
} catch (ParserConfigurationException e) {
e.printStackTrace();
} catch (SAXException e) {
e.printStackTrace();
}
return list;
};
```

9.Проверьте работоспособность приложения. В реальной программе следует

отслеживать наличие подключения устройства к сети, отслеживать ошибки соединения, а также получать данные из сети в *отдельном потоке*, чтобы не «замораживать» интерфейс пользователя.