Android Developer Workshop

Training materials and resources

Future Processing

Central Mobile Team

2014

### Android Developer Workshop

[Introduction 1](#__RefHeading__2838_351330546)

[Creating new project 1](#__RefHeading__2840_351330546)

[Activity 2](#__RefHeading__2842_351330546)

[AndroidManifest 4](#__RefHeading__2844_351330546)

[Layouts 5](#__RefHeading__2846_351330546)

[Views 7](#__RefHeading__2848_351330546)

[Map xml file to Activity class 9](#__RefHeading__2850_351330546)

[Shared Preferences 10](#__RefHeading__2852_351330546)

[AsyncTask 11](#__RefHeading__2854_351330546)

[Dialogs 12](#__RefHeading__2964_351330546)

[Exercises 14](#__RefHeading__2858_351330546)

### Creating new project

In this section we explain how to create an Android Project using Eclipse with ADT plugin.

1. File → New → Android Application Project
2. Type Application Name and change Package Name → Next
3. Next
4. Configure your launch icon → Next
5. Configure lunching activity → Next
6. Type Activity and layout name → Next

For more information go to the Android developer page: <http://developer.android.com/training/basics/firstapp/creating-project.html>

### Activity

In this section we want to talk about Activity class in Android. The Activity class is an important part of an application's overall lifecycle, and the way activities are launched and put together is a fundamental part of the platform's application model.

Activity lifecycle:



**onCreate()** - called when the activity is first created. This is where you should do all of your normal static set up: create views, bind data to lists, etc.

**onRestart()** - called after your activity has been stopped, prior to it being started again.

**onStart()** - called when activity is becoming visible to the user.

**onResume()** - called when the activity will start interacting with the user. At this point your activity is at the top of the activity stack, with user input going to it.

**onPause()** - called when the system is about to start resuming a previous activity. This is typically used to commit unsaved changes to persistent data, stop animations and other things that may be consuming CPU, etc.

**onStop()** - called when the activity is no longer visible to the user, because another activity has been resumed and is covering this one. This may happen either because a new activity is being started, an existing one is being brought in front of this one, or this one is being destroyed.

**onDestroy()** - The final call you receive before your activity is destroyed. This can happen either because the activity is finishing (someone called finish() method on it, or because the system is temporarily destroying this instance of the activity to save space).

### AndroidManifest

Activity to be known in Android application must be declared in AndroidManifest.xml.

If you create Activity while creating new project, it should be declared automatically.

Example AndroidManifest file:

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

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

package=*"fp.pl.bestworkshop"*

android:versionCode=*"1"*

android:versionName=*"1.0"* >

<uses-sdk

android:minSdkVersion=*"8"*

android:targetSdkVersion=*"17"* />

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

<application

android:allowBackup=*"true"*

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

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

android:theme=*"@style/AppTheme"* >

<activity

android:name=*"fp.pl.bestworkshop.MainActivity"*

android:configChanges=*"orientation"*

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

<intent-filter>

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

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

</intent-filter>

</activity>

</application>

</manifest>

In manifest you must also declare permissions for the application. For example:

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

This is permission for internet connection.

You can add permission by writing in xml or in manifest editor.

For more information go to the Android developer page: <http://developer.android.com/reference/android/app/Activity.html>

### Layouts

A layout defines the visual structure for a user interface, such as the UI for an activity. You can declare a layout in two ways:

* Declare UI elements in XML. Android provides a straightforward XML vocabulary that corresponds to the View classes and subclasses, such as those for widgets and layouts. You can use xml editor to drag and drop views to the Activity.
* Instantiate layout elements at runtime. Your application can create View and ViewGroup objects (and manipulate their properties) programmatically.

Example of layout declared in xml file:

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

xmlns:tools=*"http://schemas.android.com/tools"*

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

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

android:orientation=*"vertical"*

tools:context=*".MainActivity"* >

<LinearLayout

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

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

android:paddingTop=*"8dp"*

android:weightSum=*"4"* >

<ImageView

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

android:layout\_width=*"0dp"*

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

android:layout\_weight=*"1"*

android:src=*"@drawable/eu\_flag"* />

<EditText

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

android:layout\_width=*"0dp"*

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

android:layout\_weight=*"3"*

android:ems=*"10"*

android:inputType=*"numberDecimal"* />

</LinearLayout>

**LinearLayout** - a layout that organizes its children into a single horizontal or vertical row. It creates a scrollbar if the length of the window exceeds the length of the screen.



Figure 1 LinearLayout

To change orientation in xml use **android:orientation=”horizontal/vertical”** attribute in LinearLayout tag.

**RelativeLayout** - a view group that displays child views in relative positions. The position of each view can be specified as relative to sibling elements (such as to the left-of or below another view) or in positions relative to the parent RelativeLayout area (such as aligned to the bottom, left of center).



Figure 2RelativeLayout

RelativeLayout

For more information go to the Android developer page:

<http://developer.android.com/guide/topics/ui/declaring-layout.html> <http://developer.android.com/guide/topics/resources/layout-resource.html>

### Views

This class represents the basic building block for user interface components. A View occupies a rectangular area on the screen and is responsible for drawing and event handling. View is the base class for widgets, which are used to create interactive UI components (buttons, text fields, etc.).

**EditText** - a text field allows the user to type text into your app. It can be either single line or multi-line. Touching a text field places the cursor and automatically displays the keyboard. In addition to typing, text fields allow for a variety of other activities, such as text selection (cut, copy, paste) and data look-up via auto-completion.

To change input text validation use **android:inputType=” ”**.

* “text” - normal text keybord
* “textEmailAddress” - normal text keybord with the '@' character
* “textUri” - normal text keybord with '/' character
* “number” - basic number keypad
* “phone” - phone-style keypad
* “numberDecimal” - basic number keypad with decimal input validation

<EditText

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

android:layout\_width=*"0dp"*

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

android:ems=*"10"*

android:inputType=*"numberDecimal"* />

**Button** - a button consists of text or an icon (or both text and an icon) that communicates what action occurs when the user touches it.

<Button

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

android:layout\_width=*"0dp"*

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

android:text=*"some text"* />

When the user clicks a button, the Button object receives an on-click event. To define the click event handler for a button, add the a**ndroid:onClick=”clickMethod”** attribute to the Button element in your XML layout. Called method should be similar to this:

public void clickMethod(View view) {

// Do something in response to button click

}

**ImageView** - Displays an arbitrary image, such as an icon. The ImageView class can load images from various sources (such as resources or content providers), takes care of computing its measurement from the image so that it can be used in any layout manager, and provides various display options such as scaling and tinting.

<ImageView

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

android:layout\_width=*"0dp"*

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

android:src=*"@drawable/eu\_flag"* />

To set image in this view use **android:src=” ”** in ImageView tag.

**TextView** - Displays text to the user and optionally allows them to edit it. A TextView is a complete text editor, however the basic class is configured to not allow editing.

<**TextView**

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

android:layout\_width=*"0dp"*

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

android:textStyle=*"bold"*

android:text="some text" />

To set the text to display use **android:text = “ ”** and to set text style as bold, italics or normal, use **android:testStyle= “ ”**.

For more information go to the Android developer page:

<http://developer.android.com/guide/topics/ui/overview.html>

<http://developer.android.com/training/basics/firstapp/building-ui.html>

### Map xml file to Activity class

To map views form xml file to java class you should use **findViewById(...)** method. Look at example code below.

**private** Button changeRateButton;

@Override

**protected** **void** onCreate(Bundle savedInstanceState) {

**super**.onCreate(savedInstanceState);

setContentView(R.layout.*activity\_main*);

changeRateButton = (Button) findViewById(R.id.*editCurrentRateButton*);

}

In **findViewById(...)** method you must give view's id argument as R.id.viewId, so you must declare id in the xml file.

<Button

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

android:layout\_width=*"0dp"*

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

To add reaction to click on the view you can add to view OnClickListener.

changeRateButton.setOnClickListener(**new** OnClickListener() {

@Override

**public** **void** onClick(View v) {

// do sth after click on button occure

}

});

### Shared Preferences

There are generally two kinds of persistent state than an activity will deal with: shared document-like data (typically stored in a SQLite database using a content provider) and internal state such as user preferences. Shared Preferences allows you to save and retrieve persistent key-value pairs of primitive data types. You can use SharedPreferences to save any primitive data: booleans, floats, ints, longs, and strings. This data will persist across user sessions (even if your application is killed).

Saving to Shared Preferences:

@Override

**protected** **void** onStop() {

**super**.onStop();

**float** valueToStore = 5.0f;

SharedPreferences preferences = PreferenceManager.*getDefaultSharedPreferences*(**this**);

SharedPreferences.Editor editor = preferences.edit();

editor.putFloat("key", valueToStore);

editor.commit();

}

Loading form Shared Preferences:

@Override

**protected** **void** onStart() {

**super**.onStart();

SharedPreferences preferences = PreferenceManager.*getDefaultSharedPreferences*(**this**);

**float** loadedValue = preferences.getFloat("key", defaultValue);

}

For more information go to the Android developer page:

<http://developer.android.com/guide/topics/data/data-storage.html#pref>

### AsyncTask

AsyncTask enables proper and easy use of the UI thread. This class allows to perform background operations and publish results on the UI thread without having to manipulate threads and/or handlers. An asynchronous task is defined by a computation that runs on a background thread and whose result is published on the UI thread. An asynchronous task is defined by 3 generic types, called Params, Progress and Result, and 4 steps, called onPreExecute, doInBackground, onProgressUpdate and onPostExecute.

**onPreExecute()** - runs on the UI thread before [doInBackground(Params...)](http://developer.android.com/reference/android/os/AsyncTask.html#doInBackground%28Params...%29).

**doInBackground(Params... params)** - override this method to perform a computation on a background thread. The specified parameters are the parameters passed to execute(Params...) by the caller of this task. This method can call publishProgress(Progress...) to publish updates on the UI thread.

**onProgressUpdate(Progress... values)** - runs on the UI thread after publishProgress(Progress...) is invoked. The specified values are the values passed to publishProgress(Progress...).

**onPostExecute(Result result)** - runs on the UI thread after doInBackground(Params...). The specified result is the value returned by doInBackground(Params...). This method won't be invoked if the task was cancelled.

The three types used by an asynchronous task are the following:

1. Params, the type of the parameters sent to the task upon execution.
2. Progress, the type of the progress units published during the background computation.
3. Result, the type of the result of the background computation.

Not all types are always used by an asynchronous task. To mark a type as unused, simply use the type Void.

private class MyTask extends AsyncTask<Void, Void, Void> { ... }

Example of AsyncTask implementation:

private class DownloadFilesTask extends AsyncTask<URL, Integer, Long> {

protected void onPreExecute() {

// do sth before starting main task (everything here will be done in UI Thread)

}

protected Long doInBackground(URL... urls) {

// do sth in this task like downloading data form the internet (everything here will be // done in new Thread)

}

protected void onProgressUpdate(Integer... progress) {

// do sth while main task is progressing (everything here will be done in UI Thread)

}

protected void onPostExecute(Long result) {

// do sth after starting main task (everything here will be done in UI Thread)

}

}

Once created, a task is executed very simply:

new DownloadFilesTask().execute(url1, url2, url3);

For more information go to the Android developer page:

<http://developer.android.com/reference/android/os/AsyncTask.html>

### Dialogs

A dialog is a small window that prompts the user to make a decision or enter additional information. A dialog does not fill the screen and is normally used for modal events that require users to take an action before they can proceed.

Example of embeded dialog is AlertDialog. This is a dialog that can show a title, up to three buttons, a list of selectable items, or a custom layout. To build AlertDialog use AlertDialog.Builder class.

How to build AlertDialog you can see below.

**AlertDialog exampleAlertDialog =**  **new** AlertDialog.Builder(activity) //

.setIcon(R.drawable.*ic\_launcher*) //

.setTitle(R.string.*dialog\_title*) //

.setMessage(R.string.*dialog\_message*) //

.setPositiveButton(R.string.*ok*, **new** PositiveOnClickListener()) //

.setNegativeButton(R.string.*cancel*, **new** NullOnClickListener()) //

.setView(inputEditText) //

.create();

To display dialog on the screen use show() method.

**exampleAlertDialog.show()**

For more information go to the Android developer page:

<http://developer.android.com/guide/topics/ui/dialogs.html>

<http://developer.android.com/design/building-blocks/dialogs.html>

### Exercises

1. Create new Android application

* App Name: Exchange Rate Calculator
* Package Name: com.best.exchangeratecalculator

1. Look into main Activity(auto generated code)

You can see how default Android Activity looks like.

1. Make layout in xml file

Use your mocks from UX workshops

1. Map xml layout into Java Activity class

You must map views like EditText, Button to Activity using findViewByIt(...) method.

1. Add functionality to Activity

Functionality:

* Calculating PLN → Euro
* Calculating Euro → PLN
* Displaying current exchange rate 1 Euro = x PLN

1. Add saving and loading exchange rate in phone memory

Use Shared Preferences to save and load exchanged rate form phone memory. You can do it in onStart() and onStop() methods.

1. Load exchange rate from the internet

Implement AsyncTask to download exchange rate form the internet not in UI thread. You can use url: <http://www.ecb.europa.eu/stats/eurofxref/eurofxref-daily.xml>. Response is in XML format, so you must parse it to get and use require information. You must add permission for internet to AndroidManifest.

PS. Type url in web browser and look how the XML look like.

1. Add AlertDialog to modify exchange rate manually

Use AlertDialog to modify exchange rate, which is used in app, manually.