PART III
Hello World, and Bye!

Table of Contents

Exercise 1 – Synchronously call Activities....2
Task 1....3
Task 2....4
Task 3....5
Task 4....6
Exercise 2 – Intent parameter and asynchronously call activities....6
Task 1....6
Task 2....6
Exercise 3 – Standard Intents (Optional)....8
Exercise 4 – Misc Questions (Optional)....8
Exercise 5 - Activity Lifecycle (optional)....8
Exercise 6 – Views (optional)....8

Illustration Index

Illustration 1: Synchronously call Activities....2
Illustration 2: Asynchronously call activities....7

- The goal of this part is to implement a first simple Android application which includes three activities. To ease this task the application will be demonstrated first. Make sure you have the application Activities imported into your workspace.

- It is important to use the Android Developer web site[1] especially the tutorials that can be found there

- Teamwork is important

- Please do not hesitate to ask a lot of questions

Exercise 1 – Synchronously call Activities

The Activity concept is elementary when programming Android applications. As shown in the presentation Activities are components that communicate with each other via intents. Some tutorials start, for example, with user interface (UI) related concepts. We, however, found it valuable to start with the Activity concept because of its importance and, for example, UI related concepts do not differ from UI programming like Java Swing or even more modern Web technologies like Java Server Faces. We expect you to be familiar with the basics of UI programming.

The goal is to write an application involving three activities whereas **ActivityHello** instantiates and sends a new **Intent** (message) to **ActivityWorld,** which in turn instantiates a new **Intent** and sends it to **ActivityBye** *(see Illustration 1).*

The project Activities already contains the Activity ActivityHello, so you can use this part as orientation for the following steps.

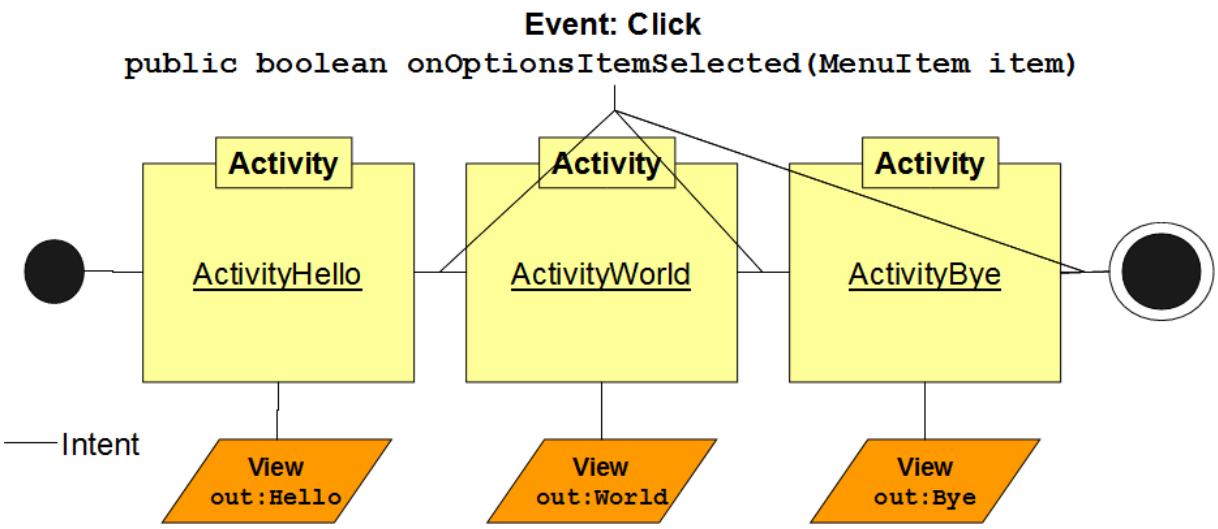

Illustration 1: Synchronously call Activities

1 http://developer.android.com/index.html (2010-11-13)

Task 1

Within the project's ***src*** folder create the three activities:
`ActivityHello.java` (does exist already)
`ActivityWorld.java`
`ActivityBye.java`

To create an Activity create a new Java class and set the superclass to **android.app.Activity**. An easy way to do so is to **right click on the source folder and select New → Java Class**.

After you have entered the name for the Activity, e.g., `ActivityWorld`, **click on the Browse button left to the Superclass text-input box**. Just **type Activity, hit enter** and **select the android.app.Activity** entry.

The Java editor should pop up and you should be able to modify the class. Before you can use the Activity it needs to be registered for this application. At the bottom of your project tree you will find a file called **AndroidManifest.xml**. Open the file. If you cannot see any XML code, this is because you are in the wrong tab, presumably. You should see five tabs at the bottom of the window. **Select the AndroidManifest.xml tab** and you should see something like the following XML snippet. If you do not see these 5 tabs you should activate the Android Views as described in the last part!

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
        package="exercise3.activities" android:versionCode="1" android:versionName="1.0">
        <application android:icon="@drawable/icon" android:label="@string/app_name">
                <activity android:name=".ActivityHello" 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>
        <uses-sdk android:minSdkVersion="3" />
   <uses-permission android:name="android.permission.INTERNET" />
</manifest>
```

The **AndroidManifest.xml** configuration file defines everything for your application, Activities, Intent-filter, or security settings for instance. The file starts with a couple of standard things, like `android:icon`, an `android:label` an so on. At this stage we are interested in defining our newly created Activity.

You will find one or a couple of `<activity>` tags within the `<application>` tag. One of them should define an Intent-filter that defines an action `android.intent.action.MAIN` and `android.intent.category.LAUNCHER`.

So far, we have not treated Intent-filter and at this stage it is sufficient to know that Android is able to filter for specific pre-defined, or even user definable, Intents and a filtered Intent is forwarded to some Activity registered for this filter or one of its specific child classes, e.g., a Service. This is also called as implicit Intent and the receiver of an Intent is not specifically specified by the developer in terms of a class name. For example,

```
Intent i = new Intent();
i.setAction(Intent.ACTION_CALL);
i.setData(Uri.parse("tel:5556"));
```

```
startActivity(i);
```

will start a phone call as the Intent `Intent.ACTION_CALL` has an associated filter that forwards the Intent with the intention to call 5556 to the phone Activity.

Have a look into the slides or the Android Help[2] for more information.

Coming back to the AndroidManifest.xml. The Intent-filter specified for **ActivityHello** ensures that as soon as the application is loaded **ActivityHello** is loaded. Precisely, it is put on top of the stack.

Each Activity you create must be defined in the same way, however, at this stage **do not define any Intent-filter.**

The message simply is:
Each new Activity you want to use in your application, notice that an Activity is a Java class that extends android.app.Activity, **must be specified in the AndroidManifest.xml. You will get an according exception, though, if you forgot to define it.**

Before you can do something with the Activity you must at least override the method `onCreate(Bundle savedInstanceState)`.

```
/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.hello);
}
```

Task 2

In the last part you should have made yourself familiar with resources. We also mentioned that the file `R.java` is something like an index to access resources of your application. The following code exploits now this functionality

```
setContentView(R.layout.hello)
```

and accesses via the index the layout, a `TextView` resource here, that is defined in ***res/layout/hello.xml***.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent">
<TextView android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="@string/hello"/>
</LinearLayout>
```

2 http://developer.android.com/guide/topics/intents/intents-filters.html (10.11.2010)

In the `<TextView>` tag you can find the attribute `android:text="@string/hello"`.
Again, the resources are accessed via the index, however, a string value as defined in ***/res/values/strings.xml***.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello</string>

    <string name="app_name">Activities</string>
</resources>
```

Define all your messages, namely "Hello", "World", and "Bye" in this file. Resources are briefly treated in the presentation of Part IV. Just have a look into the slides or the Android developer documentation[3].

Task 3

Now you will have to build the menus, instantiate and send Intents. There are several ways to fire Intents, however, in this exercise you should create a menu with the entries specified in the following table and overwrite the method `onOptionsItemSelected(MenuItem item)`. See **ActivityWorld** to get an impression.

Activity	Entry 1	Entry 2	Destination of the Intent
ActivityHello	World	Quit	ActivityWorld
ActivityWorld	Bye	Quit	ActivityBye
ActivityBye	Quit		Home

Advice:
The methods

`public boolean onCreateOptionsMenu(Menu menu)` and
`public boolean onOptionsItemSelected(MenuItem item)`

must be overwritten by each Activity to build the menu and listen to events fired by a menu. Please consider the UI section of the Android documentation[4].

To eventually fire the event have a look at the following code:

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
      switch (item.getItemId()) {
            case WORLD:
                  Intent i = new Intent(this, ActivityWorld.class);
                  startActivity(i);
                  return true;
            case QUIT:
                  quit();
                  return true;
            }
```

3 http://developer.android.com/guide/topics/resources/index.html (10.11.2010)
4 http://developer.android.com/guide/topics/ui/index.html (10.11.2010)

```
        return super.onOptionsItemSelected(item);
    }
```

Do the same for each Activity and construct the sequence of activities.

Task 4

Each Activity has to print text, for example, ActivityHello prints *"Hello"*. All text elements required for this application should be a resource. So, not only the text elements *"Hello"*, *"World"* and *"Bye!",* also the text elements for the menu should be specified as a resource. Details about resources are provided by the chapter Resources and Assets of the Android documentation[5].

Further, each Activity has an own View and you should use either a `TextView` or an `EditText` object to show the text.

In the folder */res/layout/* you will find the file hello.xml, see the XML snippet above. Now, define for each Activity a similar UI element within the layout folder. So for ActivityWorld define a world.xml, ActivityBye respectively.

Exercise 2 – Intent parameter and asynchronously call activities

As shown in Illustration 2, the scenario as defined so far is a sequence of intents. Some scenarios, however, may require a more asynchronous communication between Activities. This basically means, it is possible to send the Intent from **ActivtityHello** to **ActivityWorld,** and as soon as **ActivtiyWorld** finishes its work it sends a response back to **ActivityHello**. Notice further that **ActivtiyHello** is not interrupted and can continue with its work; it is asynchronous.

Hence, the control must be given back to the calling Activity, **ActivityHello** here, after the called Activity **ActivtityWorld** has finished its work. Technically, the Intent fired from **ActivtiyHello** must define and bind a **`requestCode`** to the Intent itself, and after **ActivityWorld** has finished, it sends this **`responseCode`** back to **ActivityHello**.

Task 1

ActivityBye must send a **`requestCode`** back to **ActivityWorld** as soon as its work is done. To pass this exercise you will have to show, at least with the Debugger, a correct **`requestCode`** and **`responseCode`**.

Task 2

It is also possible to put complete objects into the Intent. Please modify the scenario and put the text element (String object) required by the successive Activity into the

5 http://developer.android.com/guide/topics/resources/index.html (10.11.2010)

Intent. The successive Activity must now read the String object form the Intent to print it on the screen.

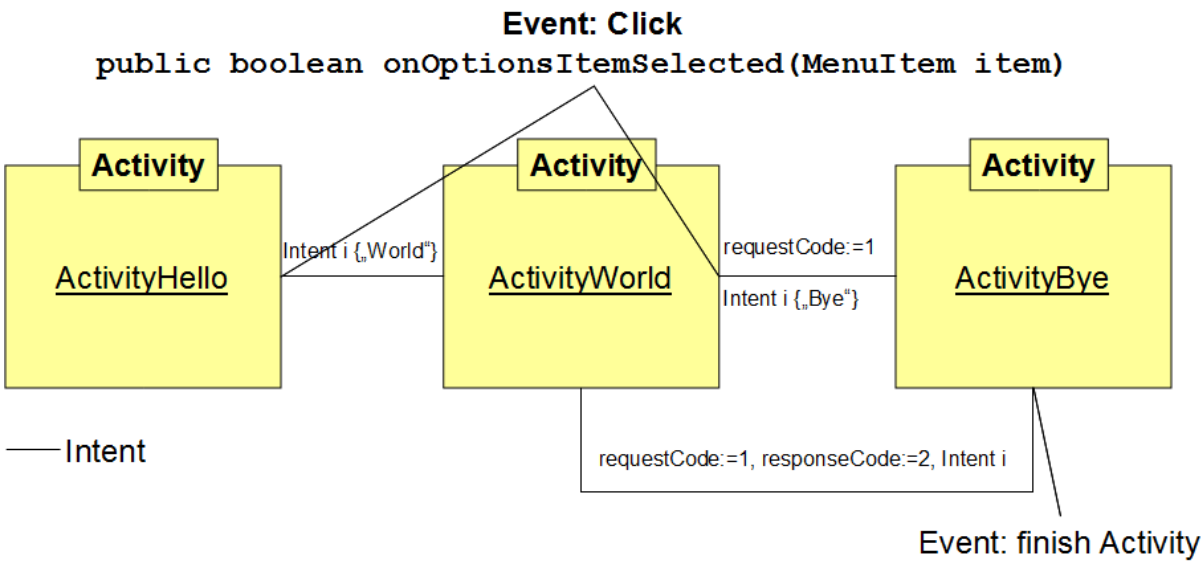

Illustration 2: Asynchronously call activities

Advice:
http://developer.android.com/guide/topics/intents/intents-filters.html
Methods (class `Activity`):
`startActivityForResult(intent, requestCode)`
`getIntent()`
`setResult()`

Exercise 3 – Standard Intents (Optional)

Please extend the menu of **ActivityHello** and provide menu items to display a picture, a web page, and simulate a call between two devices (you will have to create a second device and make yourself familiar with inter device communication).

Exercise 4 – Misc Questions (Optional)

a) How could you handle recurring menu entries, e.g. `private static final int BYE = Menu.FIRST`?

b) What is the purpose of the *AndroidManifest.xml* and what is defined in this file?

c) How could you create a new ID for a new View object so that the method `findViewById(int id)` finds the new View? Why is a cast necessary?

d) What is the purpose of the class `R.java`!

e) The constructor of Intent takes `this` as parameter. In which situation is it indispensable to pass `this` as parameter?

Exercise 5 - Activity Lifecycle (optional)

Demonstrate the *Activity Lifecycle* with at least two Activities.

Exercise 6 – Views (optional)

Advice:
Study the HelloViews example provided by the "Tutorials and Sample Code" section of the Android documentation[6].

Now, you will have to extend **ActivityHello** with an `AutoCompleteTextView`. Populate the list with words of your choice and pass the selection to the **ActivityWorld** within the Intent. Please use the menu to fire the event.

Save the selected word in an additional String object, e.g. `strWorld`, instead of printing the word in **ActivityWorld**. Change **ActivityWorld** to a **ListView** with content of your choice. After an item has been selected in the **ListView** concatenate the content of the selected item with `strWorld` and pass the result to **ActivityBye** to print it on the screen.

6 http://developer.android.com/guide/tutorials/views/index.html (10.11.2010)