Module: Mobile Application development (Android)

Session 2: Overview of Android Component

1. Android components
   1. Intents
   2. Services
   3. ContentProvider
   4. BroadcastReceiver
   5. Home screen and lock screen widgets
   6. Live Wallpapers
2. Android Development Tools
   1. Android SDK
   2. Android Development Tools
   3. Dalvik Virtual Machine
   4. How to develop Android Applications
   5. Resource editors
3. Android Application Architecture
   1. AndroidManifest.xml
   2. Activities and Lifecycle
   3. Configuration Change
   4. Context
4. Resources
   1. Using resource files
   2. Defining IDs
5. Using Resources
   1. Reference to resources in code
   2. Reference to resources in XML files
   3. Activities and layouts
6. Assets
   1. Whats are assets?
   2. Accessing assets

## Android components

Android has several more components which can be used in your Android application.

### Intents

Intents are asynchronous messages which allow the application to request functionality from other Android components, e.g. from services or activities.

An application can call a component directly (explicit Intent) or ask the Android system to evaluate registered components based on the intent data (implicit intents). For example the application could implement sharing of data via an intent and all components which allow sharing of data would be available for the user to select. Applications register themselves to an intent via an intent filter.

Intents allow an Android application to start and to interact with components from other Android applications.

### Services

Services perform tasks without providing a user interface. They can communicate with other Android components and notify the user via the notification framework in Android.

### ContentProvider

A content provider provides a structured interface to application data. Via a content provider your application can share data with other applications. Android contains an SQLite database which is frequently used in conjunction with a content provider. The SQLite database would store the data, which would be accessed via the content provider.

### BroadcastReceiver

broadcast receivers can be registered to receive system messages and intents. A broadcast receiver gets notified by the Android system, if the specified event occurs.

For example you can register a broadcast receivers for the event that the Android system completed the boot processor or for the event that the state of the phone changes, e.g. someone is calling.

### Home screen and lock screen widgets

Widgets are interactive components which are primarily used on the Android homescreen. They typically display some kind of data and allow the user to perform actions via them. For example a widget could display a short summary of new emails and if the user selects an email, it could start the email application with the selected email.

### Live Wallpapers

Live wallpapers allow you to create animated backgrounds for the Android home screen.

## Android Development Tools

### Android SDK

The Android Software Development Kit (SDK) contains the necessary tools to create, compile and package Android application. Most of these tools are command line based.

The Android SDK also provides an Android device emulator, so that Android applications can be tested without a real Android phone. You can create Android virtual devices (AVD) via the Android SDK, which run in this emulator.

The Android SDK contains the Android debug bridge (adb) tool which allows to connect to an virtual or real Android device.

### Android Development Tools

Google provides the Android Development Tools (ADT) to develop Android applications with Eclipse. ADT is a set of components (plug-ins) which extend the Eclipse IDE with Android development capabilities.

ADT contains all required functionalities to create, compile, debug and deploy Android applications from the Eclipse IDE. ADT also allows to create and start AVDs.

The Android Development Tools (ADT) provides specialized editors for resources files, e.g. layout files. These editors allow to switch between the XML representation of the file and a richer user interface via tabs on the bottom of the editor.

### Dalvik Virtual Machine

The Android system uses a special virtual machine, i.e. the Dalvik Virtual Machine to run Java based applications. Dalvik uses an own bytecode format which is different from Java bytecode.

Therefore you cannot directly run Java class files on Android, they need to get converted in the Dalvik bytecode format.

### How to develop Android Applications

Android applications are primarily written in the Java programming language. The Java source files are converted to Java class files by the Java compiler.

The Android SDK contains a tool called dx which converts Java class files into a .dex (Dalvik Executable) file. All class files of one application are placed in one compressed .dex file. During this conversion process redundant information in the class files are optimized in the .dex file. For example if the same String is found in different class files, the .dex file contains only once reference of this String.

These dex files are therefore much smaller in size than the corresponding class files.

The .dex file and the resources of an Android project, e.g. the images and XML files, are packed into an .apk (Android Package) file. The program aapt (Android Asset Packaging Tool) performs this packaging.

The resulting .apk file contains all necessary data to run the Android application and can be deployed to an Android device via the adb tool.

The Android Development Tools (ADT) performs these steps transparently to the user.

If you use the ADT tooling you press a button the whole Android application (.apk file) will be created and deployed.

### Resource editors

The ADT allows the developer to define certain artifacts, e.g. Strings and layout files, in two ways: via a rich editor, and directly via XML. This is done via multi-page editors in Eclipse. In these editors you can switch between both representations by clicking on the tab on the lower part of the screen.

## Android Application Architecture

### AndroidManifest.xml

The components and settings of an Android application are described in the AndroidManifest.xml file. For example all activities and services of the application must be declared in this file.

It must also contain the required permissions for the application. For example if the application requires network access it must be specified here.

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <manifest xmlns:android="http://schemas.android.com/apk/res/android"  package="com.my.android.temperature"  android:versionCode="1"  android:versionName="1.0">  <application android:icon="@drawable/icon" android:label="@string/app\_name">  <activity android:name=".Convert"  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="9" />  </manifest> |

The package attribute defines the base package for the Java objects referred to in this file. If a Java object lies within a different package, it must be declared with the full qualified package name.

Google Play requires that every Android application uses its own unique package. Therefore it is a good habit to use your reverse domain name as package name. This will avoid collisions with other Android applications.

android:versionName and android:versionCode specify the version of your application. versionName is what the user sees and can be any String.

versionCode must be an integer. The Android Market determine based on the versionCode, if it should perform an update of the applications for the existing installations. You typically start with "1" and increase this value by one, if you roll-out a new version of your application.

The <activity> tag defines an activity, in this example pointing to the Convert class in the com.my.android.temperature package. An intent filter is registered for this class which defines that this activity is started once the application starts (action android:name="android.intent.action.MAIN" ). The category definition category android:name="android.intent.category.LAUNCHER" defines that this application is added to the application directory on the Android device.

The @string/app\_name value refers to resource files which contain the actual value of the application name. The usage of resource file makes it easy to provide different resources, e.g. strings, colors, icons, for different devices and makes it easy to translate applications.

The uses-sdk part of the AndroidManifest.xml file defines the minimal SDK version for which your application is valid. This will prevent your application being installed on unsupported devices.

### Activities and Lifecycle

The Android system controls the lifecycle of your application. At any time the Android system may stop or destroy your application, e.g. because of an incoming call. The Android system defines a lifecycle for activities via predefined methods. The most important methods are:

* onSaveInstanceState() - called after the Activity is stopped. Used to save data so that the Activity can restore its states if re-started
* onPause() - always called if the Activity ends, can be used to release resource or save data
* onResume() - called if the Activity is re-started, can be used to initialize fields

### Configuration Change

An Activity will also be restarted, if a so called "configuration change" happens. A configuration change happens if an event is triggered which may be relevant for the application. For example if the user changes the orientation of the device (vertically or horizontally). Android assumes that an Activity might want to use different resources for these orientations and restarts the Activity.

In the emulator you can simulate the change of the orientation via **Ctrl+F11**.

You can avoid a restart of your application for certain configuration changes via the configChanges attribute on your Activity definition in your AndroidManifest.xml. The following Activity will not be restarted in case of orientation changes or position of the physical keyboard (hidden / visible).

|  |
| --- |
| <activity android:name=".ProgressTestActivity"  android:label="@string/app\_name"  android:configChanges="orientation|keyboardHidden|keyboard">  </activity> |

### Context

The class android.content.Context provides the connection to the Android system and the resources of the project. It is the interface to global information about the application environment.

The Context also provides access to Android services, e.g. the Location Service.

Activities and services extend the Context class.

## Resources

### Using resource files

Android supports that resources, like images and certain XML configuration files, can be keep separate from the source code.

These resources must be defined in the res directory in a special folder dependent on their purpose. You can also append additional qualifiers to the folder name to indicate that the related resources should be used for special configurations, e.g. you can specify that a resource is only valid for a certain screen size.

The following table give an overview of the supported resources and their standard folder prefix.

**Table 1. Resources**

| Resource | Folder | Description |
| --- | --- | --- |
| Simple Values | /res/values | Used to define strings, colors, dimensions, styles and static arrays of strings or integers. By convention each type is stored in a separate file, e.g. strings are defined in the res/values/strings.xml file. |
| Layouts | /res/layout | XML file with layout description files used to define the user interface for activities and Fragments. |
| Styles and Themes | /res/values | Files which define the appearance of your Android application. |
| Animations | /res/animator | Define animations in XML for the property animation API which allows to animate arbitrary properties of objects over time. |
| Menus | /res/menu | Define the properties of entries for a menu. |

The gen directory in an Android project contains generated values. R.java is a generated class which contains references to certain resources of the project.

If you create a new resource, the corresponding reference is automatically created in R.java via the Eclipse ADT tools. These references are static integer values and define IDs for the resources.

The Android system provides methods to access the corresponding resource via these IDs.

For example to access a String with the R.string.yourString ID, you would use the getString(R.string.yourString)) method.

R.java is automatically created by the Eclipse development environment, manual changes are not necessary and will be overridden by the tooling.

### Defining IDs

Android allows that you define ID of user interface components dynamically in the layout files, via the @+id/your\_id notation.

To control your IDs you can also create a file called ids.xml in your /res/values folder and define all IDs in this file.

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <resources>  <item name="button1" type="id"/>  </resources> |

This allow you to use the ID directly in your layout file.

|  |
| --- |
| <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  xmlns:tools="http://schemas.android.com/tools"  android:layout\_width="match\_parent"  android:layout\_height="match\_parent"  tools:context=".MainActivity" >  <Button  android:id="@id/button1"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:layout\_centerHorizontal="true"  android:layout\_centerVertical="true"  android:layout\_marginRight="27dp"  android:text="Button" />  </RelativeLayout> |

## Using Resources

### Reference to resources in code

The Resources class allows to access individual resources. An instance of Resources can get access via the getResources() method of the Context class.

The Resources class is also used by other Android classes, for example the following code shows how to create a Bitmap file from a reference ID.

|  |
| --- |
| BitmapFactory.decodeResource(getResources(), R.drawable.ic\_action\_search); |

### Reference to resources in XML files

In your XML files, for example your layout files, you can refer to other resources via the @ sign.

For example, if you want to refer to a color which is defined in a XML resource, you can refer to it via @color/your\_id. Or if you defined a "hello" string in an XML resource, you could access it via @string/hello.

### Activities and layouts

The user interface for activities is defined via layouts. The layout defines the included Views (widgets) and their properties.

A layout can be defined via Java code or via XML. In most cases the layout is defined as an XML file.

XML based layouts are defined via a resource file in the /res/layout folder. This file specifies the ViewGroups, Views, their relationship and their attributes for this specific layout.

If a View needs to be accessed via Java code, you have to give the View a unique ID via the android:id attribute. To assign a new ID to a View use . The following shows an example in which a @+id/yourvalue Button gets the button1 ID assigned.

|  |
| --- |
| <Button  android:id="@+id/button1"  android:layout\_width="wrap\_content"  android:layout\_height="wrap\_content"  android:text="Show Preferences" >  </Button> |

By conversion this will create and assign a new yourvalue ID to the corresponding View. In your Java code you can later access a View via the method findViewById(R.id.yourvalue).

Defining layouts via XML is usually the preferred way as this separates the programming logic from the layout definition. It also allows the definition of different layouts for different devices. You can also mix both approaches.

## Assets

### Whats are assets?

While the res directory contains structured values which are known to the Android platform, the assets directory can be used to store any kind of data.

### Accessing assets

You access this data via the AssetsManager which you can access the getAssets() method.

The AssetsManager class allows to read a file in the assets folder as InputStream with the open() method. The following code shows an example for this.

|  |
| --- |
| // Get the AssetManager  AssetManager manager = getAssets();  // Read a Bitmap from Assets  InputStream open = null;  try {  open = manager.open("logo.png");  Bitmap bitmap = BitmapFactory.decodeStream(open);  // Assign the bitmap to an ImageView in this layout  ImageView view = (ImageView) findViewById(R.id.imageView1);  view.setImageBitmap(bitmap);  } catch (IOException e) {  e.printStackTrace();  } finally {  if (open != null) {  try {  open.close();  } catch (IOException e) {  e.printStackTrace();  }  }  } |