Application development for mobile devices

- Usage context varies a lot
- Varying display sizes - from small phones to largish tablets
- Device security and usage fees - need to obtain permission from the user for some operations
- User personal data, such as contacts, are available
- Need to manage energy usage, be prepared for running out of battery
- Network behavior may vary quite a bit, often several alternatives available (cellular vs. WiFi)
- Devices have more features than typical PCs: acceleration, orientation etc. sensors, GPS, camera - how to take advantage of those

Application environment in Android

- Each application runs as separate Linux user
- Each application runs in its own JVM (Dalvik) and can by default access only its own files
  - (NDK - Native Development Toolkit - makes it possible to use C/C++ libraries)
- For an application with a UI the application integrates with the event processing loop in the framework - processes events the user triggers by his/her actions
- By default each application runs in its own process that has at minimum one thread - main thread where all UI events are processed

Application Components in Android

- In Android an application has four types of components that work with the platform in a specific ways:
  - **Activity -** a visible part of application that performs interaction with user. Simple applications might have just one or, more often, several activities. An activity is associated with a view that provides visible part that user can interact with.
  - **Service -** a long-running operation that doesn't necessarily need user interaction. For example, playing music in the background.
  - **Content provider -** interface provided by application for accessing (and changing) its data.
  - **Broadcast receiver -** receiving notifications generated by the system, or by applications
- To create an application, one subclasses one or more of these components, and creates zero or more custom classes (model part, for example)

Resources

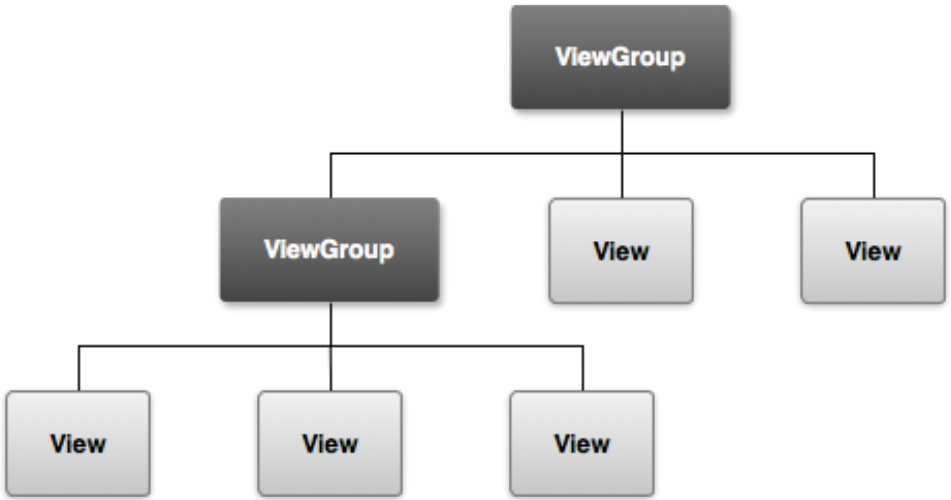
- Android resources can be used to define for example:
  - Layouts of the views in the application (res/layout folder)
  - Strings and other fixed values used in the application (res/values folder)
  - Menus (res/menu folder)
  - Graphics (res/drawables folder)
- Resource mechanism supports, for example, localization, layouts for different device orientations, graphics for different screen characteristics
- Resources are available in application Java code by referring to class R. R class is automatically compiled by the IDE.

Views and layouts

- Most straightforward way to define a layout is to write XML description for it either directly in XML (recommended way) or by using layout editor in Android studio. The description is automatically compiled by the IDE and can be loaded in an activity with `setContentView()` call.
- (View can also be created programmatically. Use that approach carefully, ie. with a good reason only)
- Create the layout for the view in `res/layout` directory (for example `main_layout.xml`, and refer to the compiled version in your activity with name `R.layout.main_layout`

View

- A view (on a screen) is a tree-like structure consisting of `View` and `ViewGroup` objects where `ViewGroups` are inner nodes and `Views` leaves:



- An example of a `ViewGroup` is `LinearLayout`, which lays out `View` or `ViewGroup` -objects in it either horizontally or vertically. Another, more flexible, example of `ViewGroup` is `RelativeLayout`.

Define Layout in XML

```
<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" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:text="@string/hello_world" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/textView1"
        android:id="@+id/button1"
        android:text="button1"/>

</RelativeLayout>
```

- `@+id` associates an identifier with an object. In activity you can refer to the object with it (for example, `R.id.textView1`).
- In XML one can refer to a named object, for example `@string/hello_world` refers to a value defined in `res/values/strings.xml`

Handling UI events

- An *event listener* is an interface in View class that contains a single callback method. This method will be called by the Android framework when the the user interacts with a view, causing an event to be created. Events depend on the view, for button widget it can be for example click or long click.
- Implementing event listener:
  - Implement anonymous event listener and register it with the View that produces the event
  - Implement event listener as a part of your Activity
  - Implement event listener in a separate object (which can, of course, be also an inner class in activity)
  - Or: refer to event handler callback method in the layout spec (simplest, if this is enough) with `android:onClick` attribute

Anonymous listener

```
public class MainActivity extends Activity {
    private final String MYNAME = "EventHandlingExample";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Log.d(MYNAME, "after setContentView()");

        Button b1 = (Button)findViewById(R.id.button1);
        b1.setOnClickListener(new View.OnClickListener() {
              @Override
              public void onClick(View v) {
                  Button b = (Button)v;
                  Log.d(MYNAME, "anonymous listener: " + b.getText() + " clicked");
              }
           }
        );
    }
…
```

Listener in activity

```
public class MainActivity extends Activity implements View.OnClickListener {
    private final String MYNAME = "EventHandlingExample";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Log.d(MYNAME, "after setContentView()");

        Button b2 = (Button)findViewById(R.id.button2);
        b2.setOnClickListener(this);
   }

    @Override
    public void onClick(View v) {
        Button b = (Button)v;
        Log.d(MYNAME, "listener in activity: " + b.getText() + " clicked");
    }
```

Separate listener

```
public class MainActivity extends Activity {
    private final String MYNAME = "EventHandlingExample";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Log.d(MYNAME, "after setContentView()");

        Button b3 = (Button)findViewById(R.id.button3);
        b3.setOnClickListener(new MyClickListener());
   }
…
```

```
public class MyClickListener implements OnClickListener {
    private final String MYNAME = "EventHandlingExample";

    @Override
    public void onClick(View v) {
      Button b = (Button)v;
      Log.d(MYNAME, "separate listener (MyClickListener): " + b.getText() + " clicked");
    }
}
```

Listener specified in layout

```
public class MainActivity extends Activity {
    private final String MYNAME = "EventHandlingExample";

…

    public void onMyClick(View v) {
      Button b = (Button)v;
      Log.d(MYNAME, “listener in layout (onMyClick): " + b.getText() + " clicked");
    }
}
```

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@id/button3"
    android:layout_alignRight="@id/button3"
    android:id="@+id/button4"
    android:text="button4"
    android:onClick="onMyClick"/>
```

AndroidManifest.xml

- Defines general (and very useful) information about the application, for example
  - Java package name for the application
  - Components in the application, in particular the component where application execution is started
  - Permissions needed by the application (for example networks access or access to contacts data)
  - Device features needed by the application (for example acceleration sensor)

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.peterh.courselab01">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Reading list

- session01.pdf
- http://developer.android.com/guide/topics/resources/overview.html
- http://developer.android.com/training/basics/firstapp/index.html
- http://developer.android.com/guide/topics/ui/declaring-layout.html (you can skip material starting from Building Layouts with an Adapter)
- http://developer.android.com/guide/topics/ui/ui-events.html
- http://developer.android.com/guide/components/activities.html