Embedded Systems Programming

Lecture 11

Verónica Gaspes
www2.hh.se/staff/vero

Center for Research on Embedded Systems
School of Information Science, Computer and Electrical Engineering

User interfaces

All android apps have user interfaces!

Views

A View object is a data structure whose properties store the layout parameters and content for a specific rectangular area of the screen.

Widgets

Views that are ready for user interaction. You can also program your own widgets (we will only look at the pre-defined ones)

Layouts

Views that put together a hierarchy of views. The top of the hierarchy is associated to the screen using `setContentView`.

Events

Capture interaction with the user. Event listeners can be associated to views.

User interfaces

All android apps have user interfaces!

Views

A View object is a data structure whose properties store the layout parameters and content for a specific rectangular area of the screen.

Widgets

Views that are ready for user interaction. You can also program your own widgets (we will only look at the pre-defined ones)

Layouts

Views that put together a hierarchy of views. The top of the hierarchy is associated to the screen using `setContentView`.

Events

Capture interaction with the user. Event listeners can be associated to views.

User interfaces

All android apps have user interfaces!

Views

A View object is a data structure whose properties store the layout parameters and content for a specific rectangular area of the screen.

Widgets

Views that are ready for user interaction. You can also program your own widgets (we will only look at the pre-defined ones)

Layouts

Views that put together a hierarchy of views. The top of the hierarchy is associated to the screen using `setContentView`.

Events

Capture interaction with the user. Event listeners can be associated to views.

User interfaces

All android apps have user interfaces!

Views

A View object is a data structure whose properties store the layout parameters and content for a specific rectangular area of the screen.

Layouts

Views that put together a hierarchy of views. The top of the hierarchy is associated to the screen using `setContentView`.

Widgets

Views that are ready for user interaction. You can also program your own widgets (we will only look at the pre-defined ones)

Events

Capture interaction with the user. Event listeners can be associated to views.

User interfaces

All android apps have user interfaces!

Views

A View object is a data structure whose properties store the layout parameters and content for a specific rectangular area of the screen.

Layouts

Views that put together a hierarchy of views. The top of the hierarchy is associated to the screen using `setContentView`.

Widgets

Views that are ready for user interaction. You can also program your own widgets (we will only look at the pre-defined ones)

Events

Capture interaction with the user. Event listeners can be associated to views.

Describing layouts in XML

A complete description of what is available can be found in
http://developer.android.com/guide/topics/resources/layout-resource.html

What we will show is what you place in the directory *res/layout* in the android project for your application in a file *filename.xml*.

In the Java code for the activity

The view described in *res/layout/filename.xml* can be attached to the view using

```
setContentView(R.layout.filename);
```

`R` is a class created by the compiler from the *res* directory.

Describing layouts in XML

A complete description of what is available can be found in
http://developer.android.com/guide/topics/resources/layout-resource.html

What we will show is what you place in the directory *res/layout* in the android project for your application in a file *filename.xml*.

In the Java code for the activity

The view described in *res/layout/filename.xml* can be attached to the view using

```
setContentView(R.layout.filename);
```

`R` is a class created by the compiler from the *res* directory.

Linear layouts

```
<?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"
>

    A list of views (widgets or other layouts)

</LinearLayout>
```

A linear layout with buttons and text views

```
<LinearLayout ... >
    <TextView
        android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a TextView"
    />
    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a Button"
    />
</LinearLayout>
```

Attaching the view to the activity

```
public class MyActivity extends Activity{

  public void onCreate(Bundle savedInstanceState){

    super.onCreate(savedInstanceState);

    setContentView.(R.layout.filename);

    TextView tv = (TextView)findViewById(R.id.text);
    Button b    = (Button)  findViewById(R.id.button);
  }

}
```

Toast messages

In order to explore a number of widgets we will make use of Toast notifications:

> *a message that pops up on the surface of the window. It only fills the amount of space required for the message and the user's current activity remains visible and interactive. The notification automatically fades in and out, and does not accept interaction events.*

Using a Toast notification

```
Toast.makeText(context, text, duration).show();
```

An editable text field

In the xml layout file

```
<EditText
     android:id="@+id/edittext"
     android:layout_width="fill_parent"
     android:layout_height="wrap_content"/>
```

An editable text field

In the onCreate() method of the FormStuff activity

```
final EditText et = (EditText)findViewById(R.id.edittext);
et.setOnKeyListener(new OnKeyListener() {
   public boolean onKey(View v,
                        int keyCode, KeyEvent event) {
      if ((event.getAction() == KeyEvent.ACTION_DOWN) &&
         (keyCode == KeyEvent.KEYCODE_ENTER)) {
            Toast.makeText(FormStuff.this,
                           et.getText(),
                           Toast.LENGTH_SHORT).show();
            return true;
      }
      return false;
   }
});
```

An editable text field

Remarks

- The `View.OnKeyListener` must implement the `onKey(View, int, KeyEvent)` method, which defines the action to be made when a key is pressed while the widget has focus.
- In this case, the method is defined to listen for the Enter key (when pressed down),
- The `onKey(View, int, KeyEvent)` method should always return true if the event has been handled, so that the event doesn't bubble-up (which would result in a carriage return in the text field).

A check box

In the xml layout file

```
<CheckBox
    android:id="@+id/checkbox"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="check it out" />
```

A check box

In the onCreate() method of the FormStuff activity

```
final CheckBox cb = (CheckBox) findViewById(R.id.checkbox);
cb.setOnClickListener(new OnClickListener() {
   public void onClick(View v) {
      if (((CheckBox) v).isChecked())
         Toast.makeText(FormStuff.this,
                        "Selected",
                        Toast.LENGTH_SHORT).show();
      else
         Toast.makeText(FormStuff.this,
                        "Not selected",
                        Toast.LENGTH_SHORT).show();
  }
});
```

A check box

Remark

If you need to change the state yourself (such as when loading a saved CheckBoxPreference), use the `setChecked(boolean)` or `toggle()` method.

Radio buttons

In the xml layout file

```
<RadioGroup
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:orientation="vertical">
    <RadioButton android:id="@+id/radio_red"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:text="Red" />
    <RadioButton android:id="@+id/radio_blue"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:text="Blue" />
</RadioGroup>
```

Radio buttons

In the onCreate() method of the FormStuff activity

```
final RadioButton radio_red =
    (RadioButton) findViewById(R.id.radio_red);
final RadioButton radio_blue =
   (RadioButton) findViewById(R.id.radio_blue);
radio_red.setOnClickListener(radio_listener);
radio_blue.setOnClickListener(radio_listener);
```

The listener that is used twice

```
private OnClickListener radio_listener=new OnClickListener(){
  public void onClick(View v) {
      RadioButton rb = (RadioButton) v;
      Toast.makeText(FormStuff.this,
                     rb.getText(), Toast.LENGTH_SHORT).show();
  }
};
```

Custom views

A custom view

You might want to define a view to visualize something. Then you need to define your own view class and include it in the xml file.

In the xml layout file

```
<se.hh.examples.forms.BallView
      android:id="@+id/bv"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_marginBottom = "350dip"/>
```

Remarks

Still do not know if this is the recommended way to use *the rest of the space*. The other view has to have a `android:layout_marginTop = "-350dip"`

The BallView class

```
public class BallView extends View {

  public BallView(Context c, AttributeSet a){
     super(c,a);
  }

  public void onDraw(Canvas canvas) {
    Paint p = new Paint();
    p.setColor(Color.RED);
    canvas.drawColor(Color.WHITE);
    canvas.drawCircle(getWidth()/2,getHeight()/2,10,p);
  }
}
```

Remark

We don't need to do anything in the activity class.

What happens when an application starts?

If the application does not have any other components running, the Android system starts a new Linux process for the application with a single thread of execution.

By default, all components of the same application run in the same process and thread (called the "main" thread).

If an application component starts and there already exists a process for that application (because another component from the application exists), then the component is started within that process and uses the same thread of execution.

Keeping the UI reactive

Because of the single thread model described above, it's vital to the responsiveness of your application's UI that you do not block the UI thread. If you have operations to perform that are not instantaneous, you should make sure to do them in separate threads ("background" or "worker" threads).

But also: do not access the Android UI toolkit from outside the UI thread (beacuse the methods that use the UI are not thread safe and we want to avoid race conditions).

What is the main thread doing?

It will *run* the chunks of code in the event listeners. Hopefully it is at rest otherwise!

Posting *runnables*

Views have a couple of methods that allow you to ask the UI thread to run some code:

```
public boolean post (Runnable action)
public boolean postDelayed (Runnable action,
                            long delayMillis)
```

What is a runnable?

The interface `java.lang.Runnable` represents a command that can be executed. A class that implements `Runnable` has to provide a method

```
public void run()
```

Threads in Java

Runnables are also what is used to introduce new threads in a Java program. This is how:

- Create an instance of class `Thread` passing a `Runnable` in the constructor.
- When you want to ask the system to start executing the thread you use the method `start()` that willl do some setups and then call the `run()` method in the runnable.

Two examples

We look at the ManyThreads program to illustrate the constructs.

We look at the prime calculator to illustrate how to use threads and runnables to keep the UI reactive.