Android

Author: Chris Zou

Contents

1. Android Activity Class

An activity corresponds to a full-screen window which the user may interact with.(e.g. set up timer, read off sensor values, make a phone call)

App contain multiple activities. Android organizes activities into tasks, a task consists of a stack (LIFO).

2. onCreate()

is the most important activity method, it gets executed when activity starts.

onCreate() set up UI (1. Creating widgets, 2.setting up event listeners)

We must call **super.onCreate()**

3. Retrieving Widgets:

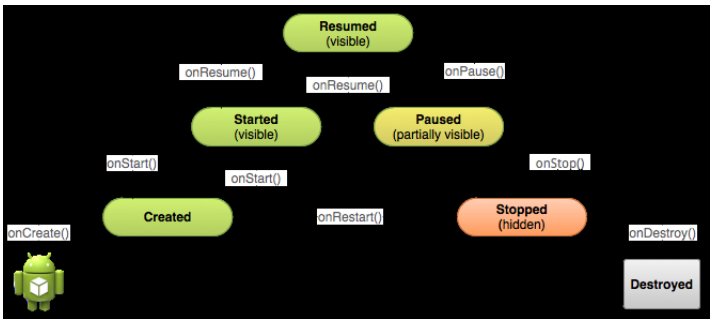
you can use the method "findViewById()" to get a hold of widgets which you declared them in the XML file.

1. You have to cast the return value: tv = (TextView) findViewById(R.id.t)
2. You must save the XML file to get the right ids on the R object.

4. Programmatically Adding Widgets:

1. Create the widget: tv = new TextView(getApplicationContext());
2. Add it to the Activity: addView(tv);

5. Android Life Cycle:



6. Programming Tips:

If you have code in a class, the code needs to access to the object that created it:

Add a filed to the EventListener with a reference to the MainActivity:

```
public class MainActivity extends Activity {
@Override
protected void onCreate (Bundle savedInstanceState ) {
/ / . . .
LightSensorEventListener el = new LightSensorEventListener ( this ) ;
}
}
```

```
clas s LightSensorEventListener implements SensorEventListener {
MainActivity m;
public LightSensorEventListener(MainActivity m) {
this.m = m;
}
/ / . . .

}
```

7. Intents:

Android uses intents to link all activities. Intent specifies a request or describes an event.

8. Saving and Restoring State:

Android kills your activity but brings it back later, you want the activity to have the same data when it comes back from the dead. (UI element is fine, but for an fields stored in activity like step counts need some code)

```
class MainActivity . . . {
String color ;
@Override
protected void onSaveInstanceState (Bundle b) {
super.onSaveInstanceState (b ) ;
b.putString ( " color " , color ) ;
}}
```

The Bundle b is another key/value map. You can then restore whatever data you saved in the onCreate method of the same Activity:

```
class MainActivity . . . {
@Override
protected void onCreate (Bundle savedInstanceState ) {
i f ( savedInstanceState != null )
color = b. getString ( " color " ) ;
}}
```

9. Inflate:

means taking an XML tree and creating a tree of View objects, based on the description in the XML.

Inflate keep on showing up in auto-generated Activity Code:

```
/ / Inf late the menu; thi s adds items to the action bar i f i t i s present .
getMenuInflater ( ) . inf lat e (R.menu. activity_main , menu) ;
```

10. Java vs XML:

Java: more flexibility

1. you can choose widgets based on user input or your computation
2. can use loops or other control flow to generate related items
3. get less error checking

XML: more safety.

1. You can select and place items on layouts at any point

2. No need to run to see how it looks
3. Get more error checking(compiler can tell you and help you fix it)

11. Android Graphics:

How to put graphics on the screen in Android:

1. Use a view (easier, suitable only for infrequent updates)
2. Paint to a Canvas (more complicated; but permits real-time updates)

Drawable Class:

- BitmapDrawable
- ShapeDrawable
- PictureDrawable

Drawing Bitmaps:

1. Drawing them in some other program and then include them
   a. Put a picture(PNG, JPG, GIF) in **res/drawables**
   b. Use ImageView to include it on the screen.

```
/ / Instantiate an ImageView and define its properties programmatically
                ImageView i = new ImageView( this ) ;
                i.setImageResource (R. drawable .my_image ) ;
        / / set the ImageView bounds to match the Drawable's dimensions
                i.setAdjustViewBounds ( true ) ;
                i.setLayoutParams (new Gallery.LayoutParams
        (LayoutParams .WRAP_CONTENT, LayoutParams .WRAP_CONTENT) ) ;
        / / Add the ImageView to the layout and
        / / set the layout as the content view
                mLinearLayout . addView( i ) ;
```

2. Do it through XML
   a. Put a picture(PNG, JPG, GIF) in **res/drawables**
   b. use XML to include it

```
<ImageView
        android: id ="@+id / imageView1"
        android: layout_height ="wrap_content "
        android: layout_width="wrap_content "
        android: src ="@drawable /myImage" / >
```

ShapeDrawable:

Primitive shapes: PathShape(lines) RectShape(rectangles) OvalShape(ovals & rings)

1. **XML way:**
   a. Draw shapes in XML and put them into an ImageView. Again, you need the drawable in the res/drawables directory. Add this snippet, which creates the ImagesView backed by the drawable to your layout XML.

```
<ImageView android : id ="@+id / imageView2"
  android : src ="@drawable / cyan_shape " . . . / >
```

   b. Create a separate XML file for the drawable itself. This provides instructions for Android to put dots on the screen.
2. **Code way:**

a. Specify the shape in Java

```
private class MyDrawableView extends ImageView {
        private ShapeDrawable mDrawable ;
        public MyDrawableView( Context context , int color ) {
        . . .
        mDrawable = new ShapeDrawable (new OvalShape ( ) ) ;
        mDrawable . getPaint ( ) . setColor ( color ) ;
        mDrawable . setBounds (0 , 0 , size , s ize ) ;
        mDrawable . setAlpha ( alpha ) ;
                }
        protected void onDraw( Canvas canvas ) {
        mDrawable .draw( canvas ) ;
                }
            }
```

b. Instantiate this MyDrawableView and add it to the parent view:

```
MyDrawableView magentaView =   new MyDrawableView( this , Color .MAGENTA) ;
magentaView. setLayoutParams (new LinearLayout . LayoutParams (160 , 160 ) ) ;
addView(magentaView) ;
```

12. Android Data Storage:

- **shared preferences**
- **file storage**
- **SQLite** (a software library which implements a database engine)
- **the Internet** (load data remotely)

12.1 Shared Preferences:

it's a set of key-value pairs. (unlike key-value pairs in the Bundle, these pairs persist across the app)

Get hands on shared preferences:

```
SharedPreferences settings = getPreferences ( 0 ) ;
```

Two important points:

- **putting data into the shared preferences (writing)**

```
SharedPreferences settings = getPreferences ( 0 ) ;
SharedPreferences . Editor editor = settings.edit ( ) ;
editor.putString ( " textFieldValue " , newFieldValue ) ;
editor.commit ( ) ;
```

- **getting it from the shared preferences. (reading)**

```
SharedPreferences set t ings = getPreferences ( 0 ) ;

 v = settings.getString ( " textFieldValue " , " " ) ;
```

This retrieves the value of the String preference textFieldValue; if no value is present, the getString call returns the default value "".

12.2 File Storage (internal & external)

All Android devices have internal storage.

External Storage: SD card, enclosed in the device

• Determine whether or not external storage is accessible and writable:

```
String state = Environment.getExternalStorageState();
if (Environment.MEDIA_MOUNTED.equals(state)){
        / / We can read and write the media
    } else if (Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)){
        / / We can only read the media
        } else{
        / / Something else is wrong .
        / / It may be one of many other states , but all we need
        / / to know is we can neither read nor write
            }
```

• External Storage Example: MapLoader

```
NavigationalMap map = MapLoader.loadMap(getExternalFilesDir(null) , "Lab-room-
peninsula.svg" ) ;
 mapView.setMap(map);
```

Our MapLoader contains this line:

```
File map = new File(dir,filename);
```

and then it builds a parser using the library call:

```
doc = docBuilder.parse (map) ;
```

which we call to get specific information about the map. It is actually the docBuilder that does all the I/O.

• Shared External Storage Directories

The getExternalFilesDir() call takes a parameter. null means that we are asking for the app's external storage directory. Android provides some shared Direcotry (e.g. DIRECTORY_MUSIC…)

• Reading from internal storage Files

```
Try {
// internal storage:
        FileInputStream os = openFileInput("internal.txt")
        BufferedReader br = new BufferedReader( new InputStreamReader(os));
        String I = br.readLine();
// - > I contains the line you just read.
        os.close();
        }  catch (IOException e) {}
```

• Reading from external storage Files

```
Try {
// external storage:

        FileOutputStream os = new FileOutputStream(new File(getExternalFilesDir(null), "external.txt"));
        BufferedReader br = new BufferedReader( new InputStreamReader(os));
        String I = br.readLine();
// - > I contains the line you just read.
        os.close();
        }  catch (IOException e) {}
```

Writing to Internal storage Files

```
try {
        // internal storage:
        FileOutputStream os = openFileOutput ( " internal.txt " , Context.MODE_PRIVATE);
        PrintWriter osw = new PrintWriter (new OutputStreamWriter (os ) ) ;
        / / --> write out the contents of string i.
        osw.println( i ) ;
        osw.close( ) ;
                } catch( IOException e) { }
```

Writing to External storage Files

```
try {
        // external storage:
        FileOutputStream os = new FileOutputStream(new File (getExternalFilesDir(nll), “external.txt”));
        PrintWriter osw = new PrintWriter (new OutputStreamWriter (os ) ) ;
        / / --> write out the contents of string i.
        osw.println( i ) ;
        osw.close( ) ;
                } catch( IOException e) { }
```

13. Toast

Toast is used to display a short message to the user.

Either:

```
Toast.makeText ( getApplicationContext ( ) , "A Toast!", Toast .LENGTH_LONG) . show( );
```

Or:

```
Toast.makeText ( MainActivity.this , "A Toast!", Toast .LENGTH_LONG) . show( );
```

You can store all of these parameters, as well as the toast object itself, in local vriables.

14. Broadcast Receivers

Android uses Intents to broadcast info about what’s happening on the system between apps.

e.g. Apps want to know about events like screen rotation, phone connected to PC

Rotation Listener: (register programmatically all java)

```
public class MainActivity extends Acivity {
    BroadcastReceiver broadcastReceiver = new BroadcastReceiver ( ) {
            @Override
        public void onReceive (Context c , Intent i) {
            int orientation = c.getResources ( ).getConfiguration ( ).orientation ;
          if ( orientation == Configuration .ORIENTATION_PORTRAIT)
                    {Toast .makeText ( c , " Portrait " , Toast .LENGTH_SHORT).show( ) ;}
           else if (orientation == Configuration .ORIENTATION_LANDSCAPE)
                    {Toast .makeText ( c , "Landscape " , Toast .LENGTH_SHORT) . show( ) ;}
            else
                    { Toast .makeText ( c , " ? ? ? " , Toast .LENGTH_SHORT) . show( ) ;}
        }
            } ;
 IntentFilter int entFilter = new IntentFilter (Intent .ACTION_CONFIGURATION_CHANGED) ;
```

```
@Override
    protected void onCreate (Bundle savedInstanceState ) {
    super.onCreate ( savedInstanceState ) ;
    setContentView(R. layout . activity_main ) ;
    registerReceiver ( broadcastReceiver , intentFilter ) ;
        } }
```

Unfortunately, by default Android destroys your app on a screen rotate event, so that doesn't work unless you ask Android to not destroy your app on rotation.

Phone Ring Listener: (register statically through XML, and Java)

1. **Modify the manifest.xml to permit the app to listen for phone calls:**

```
<uses-permission android :name="android . permission .READ_PHONE_STATE">
</uses-ppermission>
```

2. **Register the listener statically in the manifest:**

```
<receiver android:name=" ca .xxx.xxx.MyPhoneReceiver " >
    <intent-filter >
        <action android:name="android.intent.action.PHONE_STATE" / >
    </ intent-filter >
</ receiver >
```

3. **Create a class MyPhoneReceiver:**

```
public class MyPhoneReceiver extends BroadcastReceiver {
 @Override
  public void onReceive ( Context context , Intent intent ) {
  Bundle extras = intent.getExtras ( ) ;
   if ( extras != null ) {
        String state = extras.getString (TelephonyManager .EXTRA_STATE ) ;
    Log.w( "PHONE" , state ) ;
   if ( state . equals (TelephonyManager .EXTRA_STATE_RINGING) ) {
     String phoneNumber = extras . getString (TelephonyManager.EXTRA_INCOMING_NUMBER) ;
Log.w( "PHONE" , phoneNumber ) ;
}}}}
```

15. Lists (ListView)

ListView : an UI element which shows a list of items to the user.

15.1 Creating and populating a ListView

15.1.1 Create a ListView objet by dragging it onto the Activity's XML file.

(manually edit the android::id attribute so that its value is @android:id/list)

15.1.2 Change your activity to be a ListActivity & populate the list with entries

This allows us to set the items of the ListView. Also, you need to add a filed listAdapter of type ArrayAdapter<String> to your Activity. (we currently store the ArrayAdapter as a field. That means it'll go away whenever the Activity is destroyed (e.g. rotation)).

```
List <String > data = new ArrayList <String > ( ) ;

data . add( " ECE155 " ) ;
```

```
data . add( " ECE106 " ) ;

data . add( " ECE124 " ) ;

listAdapter = new ArrayAdapter<String >( this , android .R. layout . simple_list_item_1 ,
data ) ; setListAdapter ( listAdapter) ;
```

15.1.3 Create a new method in the ListActivity. (When we click on a list item, we want something to happen.)

```
@Override
protected void onListItemClick ( ListView l , View v , int position , long id ) {
super . onListItemClick ( l , v , position , id ) ;
String s = ( String ) getListAdapter ( ) . getItem( posi t ion ) ;
Toast .makeText ( this , "Aha: "+s , Toast .LENGTH_SHORT) . show( ) ;
}
```

15.2 Dynamically updating the ListView(add tiems to the ListView programmatically)

Note that adding elements to the ArrayAdapter also adds them to the ListActivity.

In a click listener, you can write:

```
String now = String.valueOf ( System. currentTimeMillis ( ) ) ;
listAdapter.add(now) ;
```