Android Programming Basics

V. Paúl Pauca

Department of Computer Science
Wake Forest University

CSC 331-631
Fall, 2013

Integrated Development Environment

- ADT Bundle for Eclipse
  `http://developer.android.com`
- IntelliJ IDEA – JetBrains
  `http://jetbrains.com/idea`
- SDK tools from command line

Demo & Objectives

First Android App

- Android app creation process demo `http://developer.android.com/training/basics/firstapp/index.html`

Objectives

- Understand the Android app life cycle
- Understand user interface vs. backend code
- Understand class structure and object creation

Outcomes

For me

- Get you started with the process
- Use UML to glean code structure and behavior

For you

- Do the first Android app tutorial on your own
- Understand the big picture of where things go
- Start thinking about additional things you need, e.g. maps, accelerometer, etc.

Android Code Structure

AndroidManifest.xml

- Specifies app characteristics, defines each of its contents
- Specifies which `Activity` will be started

Java code (`src/`)

- `Activity.java` files, each handles a screen layout
- Contain all callbacks for views in the screen

XML code (`res/layout/`)

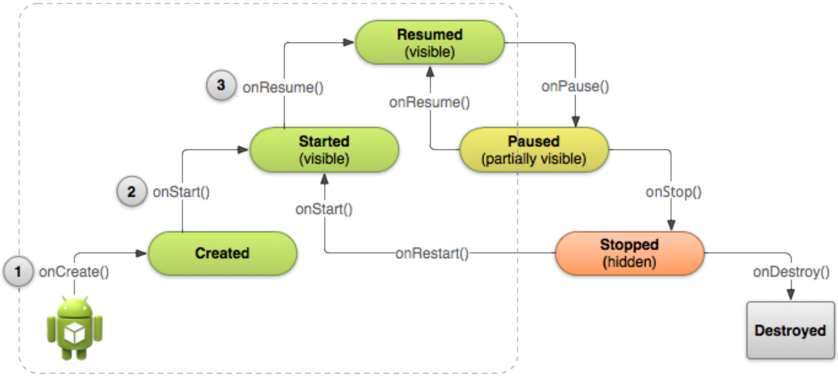
- Defines the layout for a particular screen
- Each layout is handled by an `Activity`

Activity Java files

- Located in `src/package`
- Each extends the `Activity` class
- Inherited Activity methods that handle the activity life cycle:
  - `onCreate()`
  - `onStart()`
  - `onResume()`
  - `onPause()`
  - `onStop()`
  - `onRestart()`
  - `onDestroy()`
- Your app must override at least `onCreate()`

(1) Life Cycle: Creating the Actvity

Creating an instance of an activity



(1) Life Cycle: Creating the Actvity

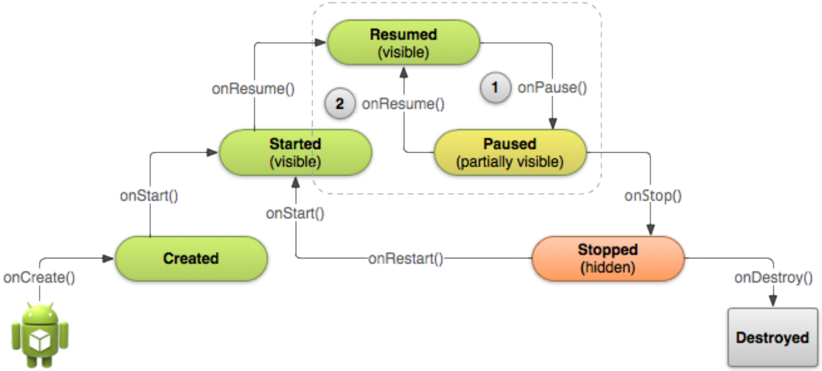
MainActivity.java

```
public class MainActivity extends Activity {

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

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        // Inflate the menu
        // this adds items to the action bar if it is present.
        getMenuInflater().inflate(R.menu.main, menu);
        return true;
    }
}
```

onStart() and **onResume()** are inherited

(2) Life Cycle: Normal Operation

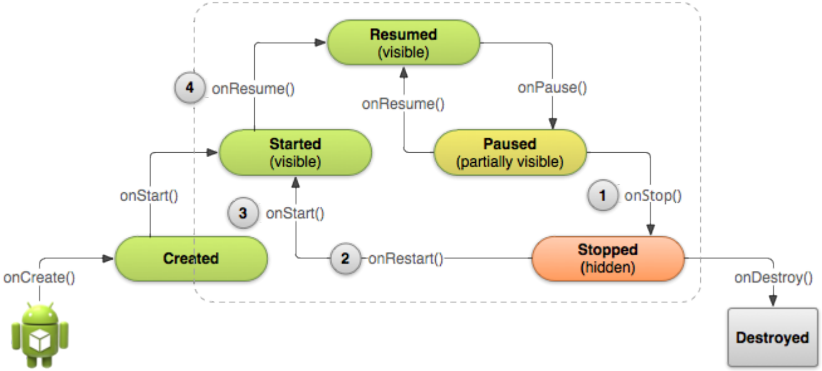
App is waiting for user interaction



onResume() and **onPause()** are inherited

(3) Life Cycle: App is Stopped

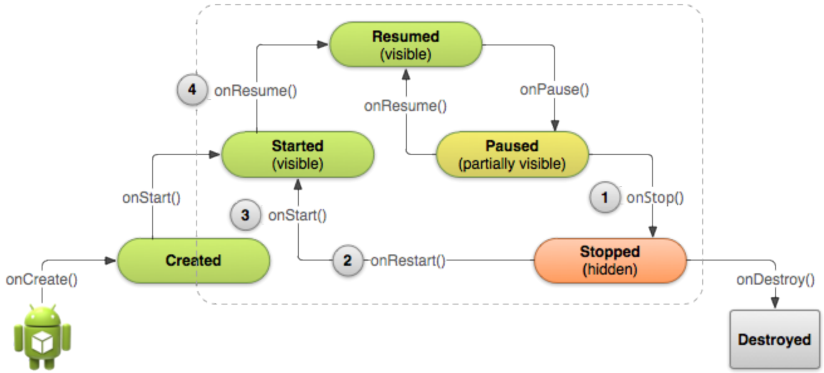
App went to the background, e.g. user pressed Home



Use `onStop()` to save app data

(4) Life Cycle: App is Restarted

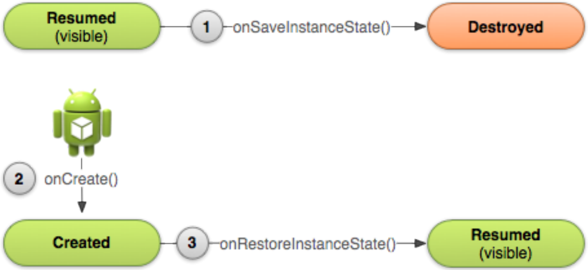
App is restarted



`onRestart()` called to make app visible again

(5) Life Cycle: App is Destroyed

App is destroyed, e.g. Back is pressed



Override `onSaveInstanceState()` to save app data not in the views into the `Bundle`

Override `onRestoreInstanceState()` to retrieve app data from the `Bundle`

Managing the Activity Life Cycle

Complete description about the Activity Life Cycle found in:

`http://developer.android.com/training/basics/activity-lifecycle/index.html`

Android Java Project

File Content

- `src/` Java `Activity` files
- `gen/` auto-generated code, `R.java`
- `res/`
  - `drawable/` image, icon bitmaps, etc.
  - `layout/` screen layout
  - `menu/` menu layout
  - `values/` string and color definitions (facilitate internationalization)

`R.java` contains addresses of all layout views

`findViewById()` used in `Activity` to obtain reference to a layout view

Designing Screen Layouts

- Default layout: `activity_main.xml`
- Edit: drag & drop in the graphical layout, or write XML code directly in `activity_main.xml`

```
<LinearLayout 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"
    android:orientation="horizontal"
    tools:context=".MainActivity" >

        <EditText android:id="@+id/edit_message"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:hint="@string/edit_message"
            android:layout_weight="1"
            />

        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/button_send"
            android:onClick="sendMessage"
            />

</LinearLayout>
```

Designing Screen Layout

- Don't use `RelativeLayout`, it's hard to modify
- Instead, use a hierarchy of `LinearLayout`s

- Drag & drop image, icons, etc. in a `res/drawable/` folder
- Image names must contain only lowercase, underscore, numbers (preferred type is .png)

- Caution! Errors in resource files turn off auto-generation of `R.java`

Designing Screen Layout

Showing images over views

- Set `background` property of the desired view
- Example:

```
        <Button
            android:layout_width="100dp"
            android:layout_height="50dp"
            android:onClick="sendMessage"
            android:background="@drawable/send_button"
            />
```

- Can adjust the view size directly (unit is dp)
- `text` property removed since image background is used
- `send_button.jpg` image must be located in a `drawable` folder

- `onClick` property specifies the Activity method responding to this button

Activity Code

```
public class MainActivity extends Activity {
   private EditText editText;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      editText = (EditText)findViewById(R.id.edit_message);
   }

   @Override
   public boolean onCreateOptionsMenu(Menu menu) {
      getMenuInflater().inflate(R.menu.main, menu);
      return true;
   }

   public void sendMessage(View view) {
      Intent intent = new Intent(this, DisplayMessageActivity.class);
      String message = editText.getText().toString();
      intent.putExtra(EXTRA_MESSAGE, message);
      startActivity(intent);
   }
```