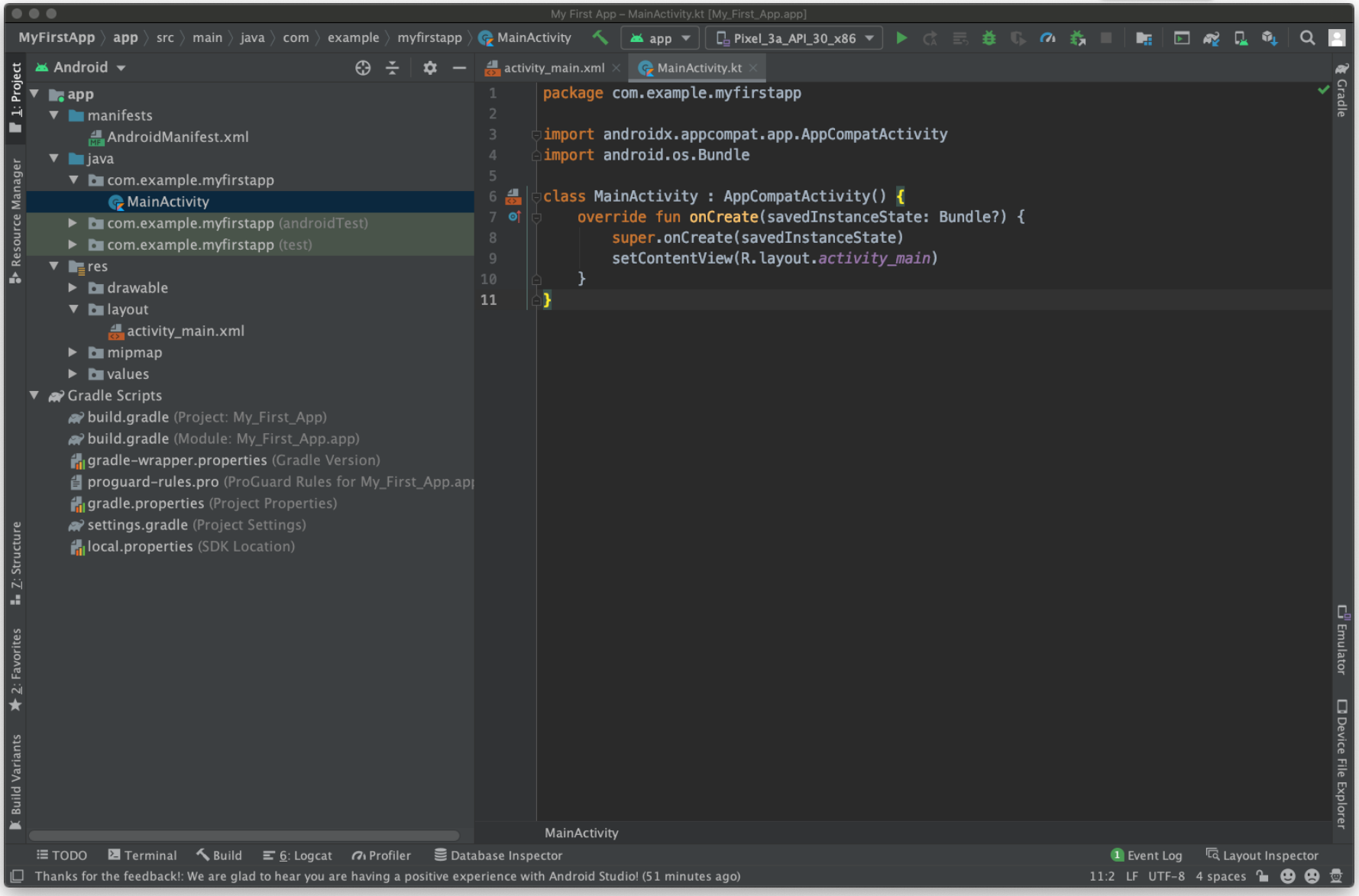
Mobile Computing Course

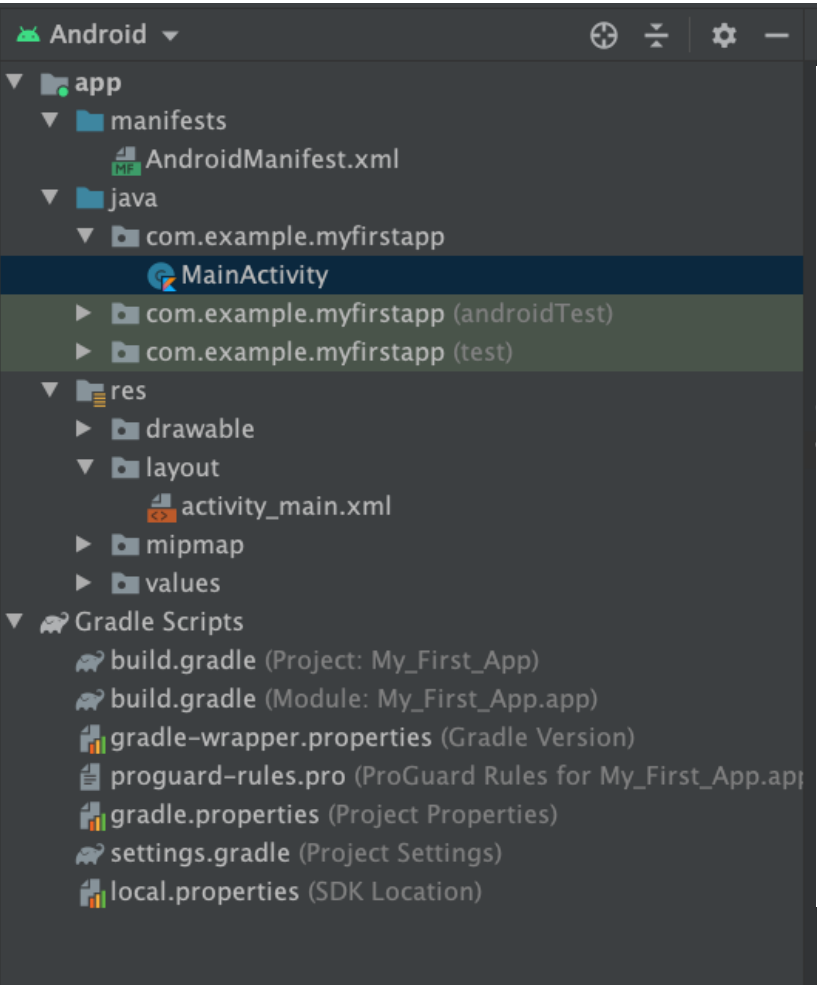
ANDROID APPLICATION

2ND SESSION

Last session, we created our first android application



Project Architecture



app > java > com.example.myfirstapp > MainActivity

This is the main activity. It's the entry point for your app. When you build and run your app, the system launches an instance of this `Activity` and loads its layout.

app > res > layout > activity_main.xml

This XML file defines the layout for the activity's user interface (UI). It contains a `TextView` element with the text "Hello, World!"

app > manifests > AndroidManifest.xml

The manifest file describes the fundamental characteristics of the app and defines each of its components.

Gradle Scripts > build.gradle

There are two files with this name: one for the project, "Project: My_First_App," and one for the app module, "Module: My_First_App.app." Each module has its own `build.gradle` file, but this project currently has just one module. Use each module's `build.gradle` file to control how the Gradle plugin builds your app. For more information about this file, see Configure your build.

Application Components

App components are the essential building blocks of an Android app. Each component is an entry point through which the system or a user can enter your app. Some components depend on others.

There are four different types of app components:

- Activities
- Services
- Broadcast receivers
- Content providers

Each type serves a distinct purpose and has a distinct lifecycle that defines how the component is created and destroyed.

Activity

- An ***activity*** is the entry point for interacting with the user. It represents a single screen with a user interface.
- For example, an email app might have one activity that shows a list of new emails, another activity to compose an email, and another activity for reading emails
- Although the activities work together to form a cohesive user experience in the email app, each one is independent of the others.
- You implement an activity as a subclass of the Activity class.

```
public class MainActivity extends Activity {
}
```

The Activity Lifecycle

Over the course of its lifetime, an activity goes through a number of states. You use a series of callbacks to handle transitions between states

onCreate()

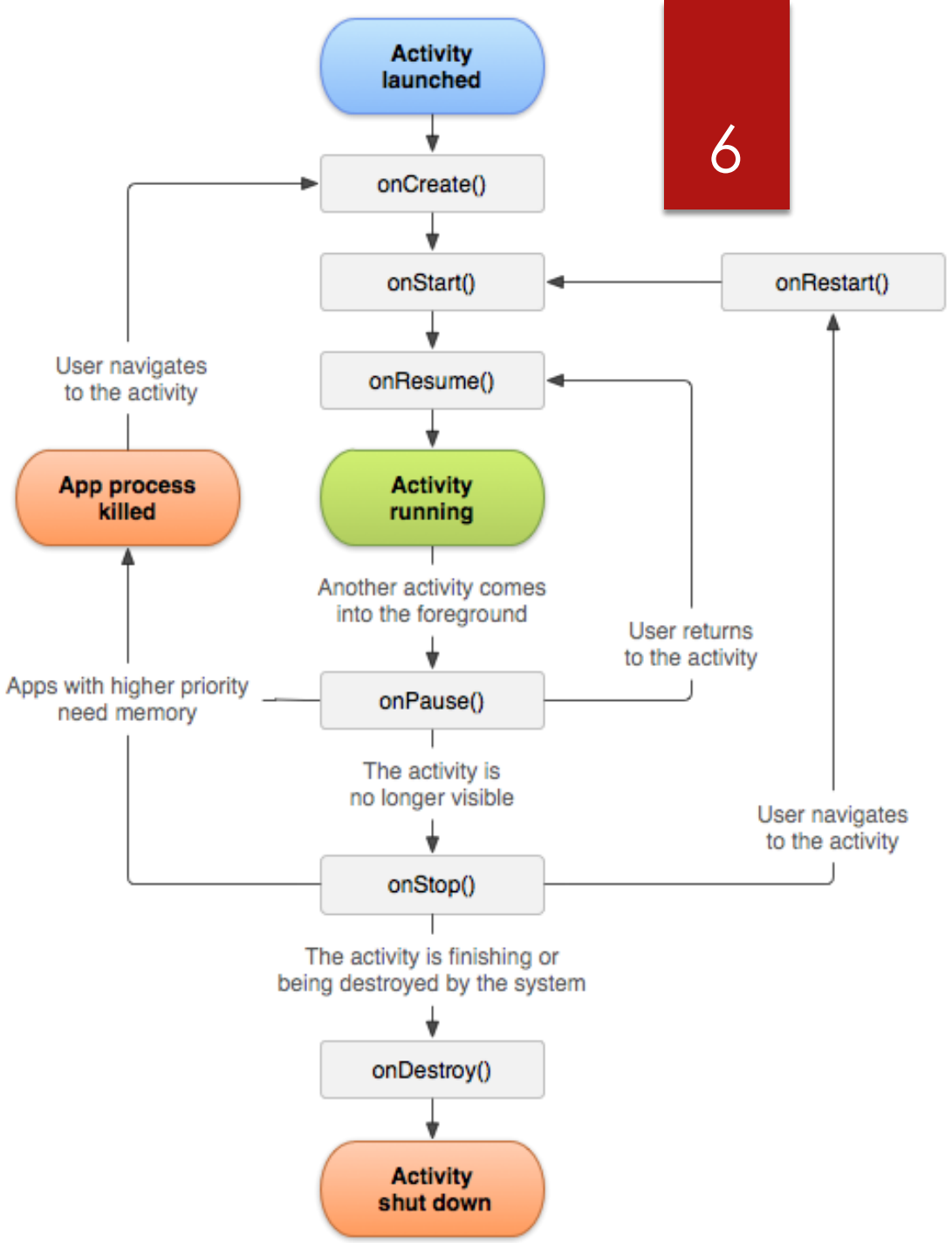
- You must implement this callback, which fires when the system creates your activity.
- Your implementation should initialize the essential components of your activity: For example, your app should create views and bind data to lists here.
- When onCreate() finishes, the next callback is always onStart().

onStart()

- As onCreate() exits, the activity enters the Started state, and the activity becomes visible to the user.
- This callback contains what amounts to the activity's final preparations for coming to the foreground and becoming interactive.

onResume()

- The system invokes this callback just before the activity starts interacting with the user.
- At this point, the activity is at the top of the activity stack, and captures all user input.
- Most of an app's core functionality is implemented in the onResume() method.
- The onPause() callback always follows onResume().



The Activity Lifecycle

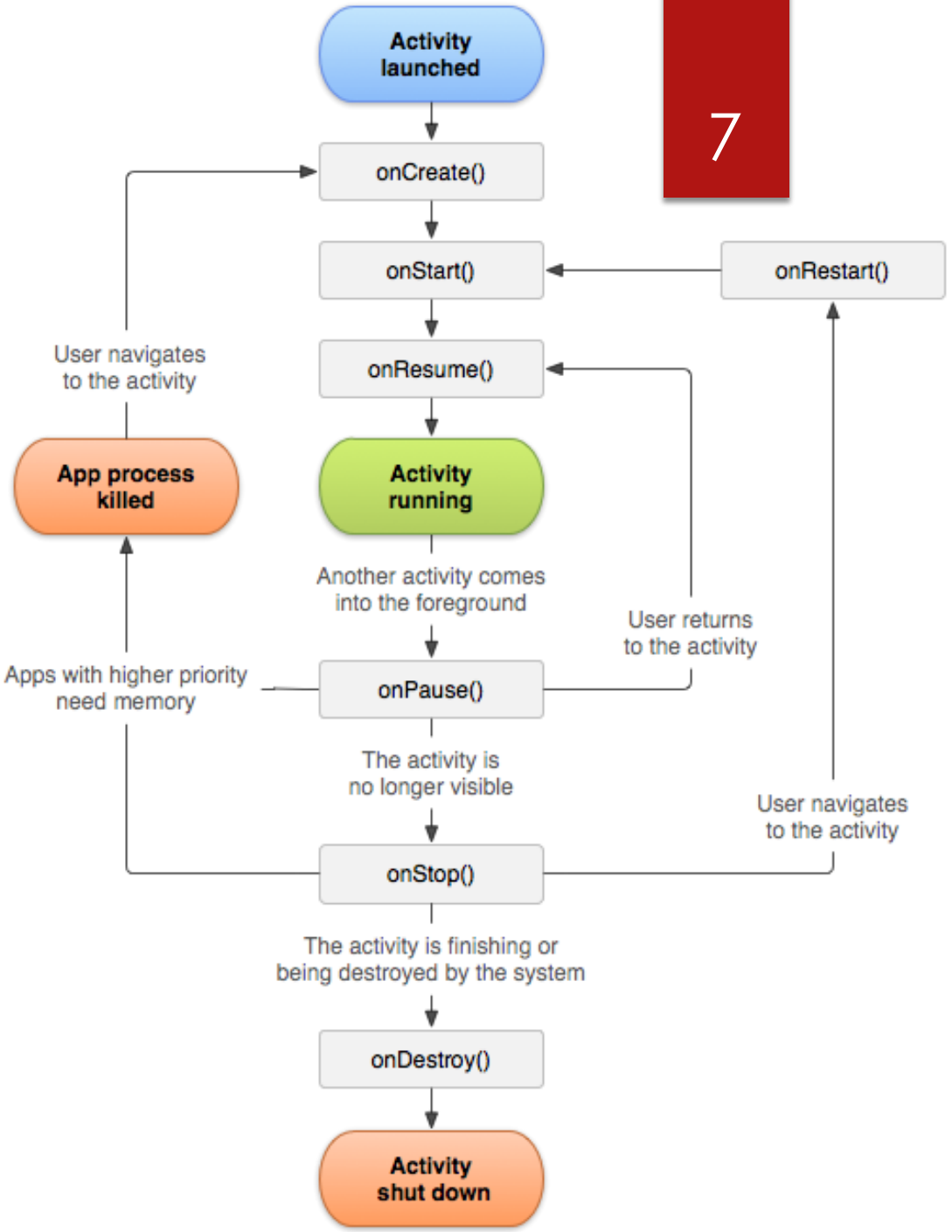
Over the course of its lifetime, an activity goes through a number of states. You use a series of callbacks to handle transitions between states

onPause()

- The system calls onPause() when the activity loses focus and enters a Paused state. This state occurs when, for example, the user taps the Back or Recents button.
- When the system calls onPause() for your activity, it technically means your activity is still partially visible, but most often is an indication that the user is leaving the activity, and the activity will soon enter the Stopped or Resumed state.
- An activity in the Paused state may continue to update the UI if the user is expecting the UI to update. Examples of such an activity include one showing a navigation map screen or a media player playing. Even if such activities lose focus, the user expects their UI to continue updating.
- Once onPause() finishes executing, the next callback is either onStop() or onResume(), depending on what happens after the activity enters the Paused state.

onStop()

- The system calls onStop() when the activity is no longer visible to the user. This may happen because the activity is being destroyed, a new activity is starting, or an existing activity is entering a Resumed state and is covering the stopped activity. In all of these cases, the stopped activity is no longer visible at all.
- The next callback that the system calls is either onRestart(), if the activity is coming back to interact with the user, or by onDestroy() if this activity is completely terminating.



The Activity Lifecycle

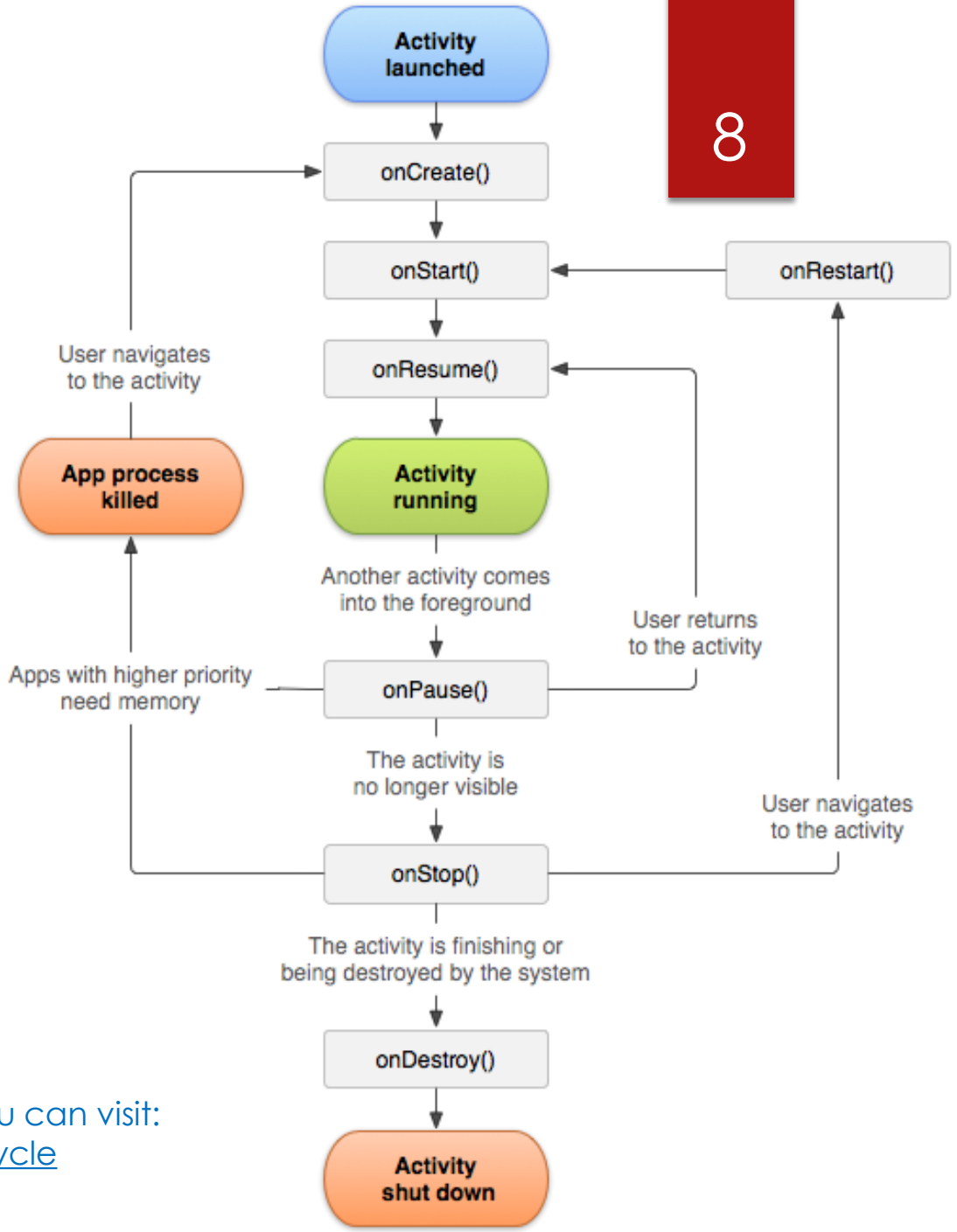
Over the course of its lifetime, an activity goes through a number of states. You use a series of callbacks to handle transitions between states

onRestart()

- The system invokes this callback when an activity in the Stopped state is about to restart. onRestart() restores the state of the activity from the time that it was stopped.
- This callback is always followed by onStart().

onDestroy()

- The system invokes this callback before an activity is destroyed.
- This callback is the final one that the activity receives.
- onDestroy() is usually implemented to ensure that all of an activity's resources are released when the activity, or the process containing it, is destroyed.



For a more detailed treatment of the activity lifecycle and its callbacks, you can visit: https://developer.android.com/guide/components/activities/activity-lifecycle

Services

- A ***service*** is a general-purpose entry point for keeping an app running in the background for all kinds of reasons. It is a component that runs in the background to perform long-running operations or to perform work for remote processes.
- A service does not provide a user interface.
- For example, a service might play music in the background while the user is in a different app, or it might fetch data over the network without blocking user interaction with an activity.
- A service is implemented as a subclass of Service

```
public class MyService extends Service {
}
```

Broadcast Receivers

- A ***broadcast*** *receiver* is a component that enables the system to deliver events to the app outside of a regular user flow, allowing the app to respond to system-wide broadcast announcements.
- Many broadcasts originate from the system—for example, a broadcast announcing that the screen has turned off, the battery is low, or a picture was captured. Apps can also initiate broadcasts—for example, to let other apps know that some data has been downloaded to the device and is available for them to use.
- Although broadcast receivers don't display a user interface, they may create a status bar notification to alert the user when a broadcast event occurs.
- A broadcast receiver is implemented as a subclass of BroadcastReceiver and each broadcast is delivered as an Intent object.

```
public class MyReceiver  extends  BroadcastReceiver {
   public void onReceive(context,intent){}
}
```

Content Providers

- A ***content provider*** manages a shared set of app data that you can store in the file system, in a SQLite database, on the web, or on any other persistent storage location that your app can access. Through the content provider, other apps can query or modify the data if the content provider allows it.
- For example, the Android system provides a content provider that manages the user's contact information.
- A content provider is implemented as a subclass of ContentProvider.

```
public class MyContentProvider extends  ContentProvider {
   public void onCreate(){}
}
```

Three of the four component types—activities, services, and broadcast receivers—are activated by an asynchronous message called an **intent**.

Intent

- Intents bind individual components to each other at runtime. You can think of them as the messengers that request an action from other components, whether the component belongs to your app or another.
- For activities and services, an intent defines the action to perform (for example, to view or send something) and may specify the URI of the data to act on. For example, an intent might convey a request for an activity to show an image or to open a web page.
- For broadcast receivers, the intent simply defines the announcement being broadcast. For example, a broadcast to indicate the device battery is low includes only a known action string that indicates battery is low.

Building The First Android Application

- XML
- Strings
- MainActivity

ple > lapshop > session2calculator > MainActivity

vity_main.xml × | strings.xml × | Translations Editor × | MainActivity.java ×

MainActivity | adding()

```java
package com.example.lapshop.session2calculator;

import ...

public class MainActivity extends AppCompatActivity {

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

    public void adding (View view){
        EditText firstET = (EditText) findViewById(R.id.fstNumEditText);
        EditText secondET = (EditText) findViewById(R.id.secNumEditText);
        TextView resultTV = (TextView) findViewById(R.id.resultTextView);

        //convert value into int
        int fstNum = Integer.parseInt(firstET.getText().toString());
        int secNum = Integer.parseInt(secondET.getText().toString());

        //sum these two numbers
        int result = fstNum + secNum;

        //display this text to TextView
        resultTV.setText(result+ "");
    }
}
```

OR

ple > lapshop > session2calculator > MainActivity

vity_main.xml × | strings.xml × | Translations Editor × | MainActivity.java ×

```java
package com.example.lapshop.session2calculator;

import ...

public class MainActivity extends AppCompatActivity {

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

        Button addbtn = (Button) findViewById(R.id.addBtn);

        addbtn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                EditText firstET = (EditText) findViewById(R.id.fstNumEditText);
                EditText secondET = (EditText) findViewById(R.id.secNumEditText);
                TextView resultTV = (TextView) findViewById(R.id.resultTextView);

                int fstNum = Integer.parseInt(firstET.getText().toString());
                int secNum = Integer.parseInt(secondET.getText().toString());
                int result = fstNum + secNum;
                resultTV.setText(result+ "");
            }
        });
    }
}
```

Building The Second Android Application

Building an intent for starting another activity

Intent Implementation

- The **Intent** constructor takes two parameters, a Context and a Class.
- The Context parameter is used first because the Activity class is a subclass of Context.
- The Class parameter of the app component, to which the system delivers the Intent, is, in this case, the activity to start.
- The **putExtra()** method passes data as key-value pairs called *extras*.
- The **startActivity()** method starts an instance of the **Activity** that's specified by the Intent.
- The **getIntent()** method gets the intent that was passed to an Activity.
- The **getStringExtra():** retrieved the String that passed to the intent object.

lapshop > session2 > MainActivity

tivity_main.xml × | strings.xml × | Translations Editor × | MainActivity.java × | activity_displa

```java
package com.example.lapshop.session2;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;

public class MainActivity extends AppCompatActivity {
    public static final String EXTRA_MESSAGE = "com.example.myfirstapp.MESSAGE";

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

    /** Called when the user taps the Send button */
    public void sendMessage(View view) {
        // Do something in response to button
        Intent intent = new Intent(this, DisplayMessageActivity.class);
        EditText editText = (EditText) findViewById(R.id.editText);
        String message = editText.getText().toString();
        intent.putExtra(EXTRA_MESSAGE, message);
        startActivity(intent);
    }
}
```

ivity_main.xml × | DisplayMessageActivity.java × | MainActivity.java × | AndroidManifest.xml

```java
package com.example.lapshop.session2;

import ...

public class DisplayMessageActivity extends AppCompatActivity {

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

        // Get the Intent that started this activity and extract the string
        Intent intent = getIntent();
        String message = intent.getStringExtra(MainActivity.EXTRA_MESSAGE);

        // Capture the layout's TextView and set the string as its text
        TextView textView = (TextView) findViewById(R.id.textView);
        textView.setText(message);

    }
}
```

Add upward navigation

Each screen in your app that's not the main entry point, which are all the screens that aren't the home screen, must provide navigation that directs the user to the logical parent screen in the app's hierarchy. To do this, add an **Up** button in the app bar.

To add an **Up** button, you need to declare which activity is the logical parent in the `AndroidManifest.xml` file. Open the file at **app > manifests > AndroidManifest.xml**, locate the `<activity>` tag for `DisplayMessageActivity`, and replace it with the following:

```
<activity android:name=".DisplayMessageActivity"
          android:parentActivityName=".MainActivity">
    <!-- The meta-data tag is required if you support API level 15 and lower -->
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value=".MainActivity" />
</activity>
```

The Android system now automatically adds the **Up** button to the app bar.

Task

Create an android application that accept two numbers and when the user press on **calculate** button it will display the result of addition, subtraction, Multiplication and division for the two numbers and store the result in text file.

Use minimum API level 4.3 (jelly beans).

To send your task, please submit an e-mail to enghebafathy18@gmail.com

Which include a compressed folder of your project and APK file with subject:
StudentName _Task no.1

Or you hand your project over to me personally.

See you in next session