DA: Android Introduction: Core Components

Applications for mobile devices - Theory - Unit 2

Didac Florensa Cazorla

Any: 2021-2022
Curs: 102386
Institut: University of Lleida (Campus Igualada)
Titulació: Bachelor's degree in Digital Interaction and Computing Techniques (GTIDIC)

Campus Universitari Igualada - UdL

Universitat de Lleida

Agenda

Introduction

Warm-up

- Dr. Jordi Mateo Fornés
- **Office**:
  - Office A.12 (Campus Igualada)
  - Office 3.08 (EPS Lleida)
- **Email**: jordi.mateo@udl.cat
- **Twitter**: https://twitter.com/MatForJordi
- **Github**: https://github/JordiMateoUdL

- Ph.D Dídac Florensa Cazorla
- **Office**:
  - Office A.12 (Campus Igualada)
  - Office 3.08 (EPS Lleida)
- **Email**: didac.florensa@udl.cat

Activities

What is an Activity?

The mobile-app experience differs from its desktop counterpart in that a user's interaction with the app doesn't always begin in the same place. $\Rightarrow$ the interaction begins non-deterministic.

Exemple

If you open an email app from your home screen, you might see a list of emails. By contrast, if you are using a social media app that then launches your email app, you might go directly to the email app's screen for composing an email.

The Activity class is designed to facilitate this paradigm. When one app invokes another, the calling app invokes an activity in the other app, rather than the app as an atomic whole.

Definition

An **Activity** is an application component (JAVA class). It represents one window (App UI screen) and one hierarchy of views.

What does an Activity do?

- The **Activity** serves as the entry point for an app's interaction with the user. You implement an **Activity** as a subclass of the *Activity class*.
- Each **Activity** can then start another **Activity** to perform different actions.
- One **Activity** in an app represents the first screen when the user launches the App (MainActivity).
- Each **Activity** can then start another **Activity** to perform different actions.
- Has a life cycle: is created, started, runs, is paused, resumed, stopped, and destroyed.
- Handles user interactions, such as button clicks, text entry, or login verification.
- Each **Activity** is only loosely bound to the other activities.
- **Activities** can be organized in parent-child relationships in the Android manifest to aid navigation.

How to implement an Activity

New > Activity > EmptyActivity

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

1. *MainActivity* is a JAVA class that extends *AppCompatActivity*.
2. Define an XML file (screen layout). Check *res>layout>activity_mail.XML*.
3. Declare the Activity in the Android Manifest file.
4. Connect the activity with the layout. `setContentView(R.layout.activity_main);`.

Inheritance in Java

Definition

Inheritance is an essential pillar of OOP(Object-Oriented Programming). It is the mechanism in java by which one class can inherit the features(fields and methods) of another class.

- **Super Class**: The class whose inherited features (base class or a parent class).
- **Sub Class**: The class that inherits the other class is a subclass(or a derived class, extended class, or child class). The subclass can add its fields and methods to the superclass fields and methods.

```
public class User{}
public class Student extends Users{
// All from User
// Specifics from Students
}
public class Teacher extends Users{
// All from User
// Specifics from Teachers
}
```

⇒ By doing this, we are reusing the fields and methods of the existing class (**reusability**).

Override in Java

Definition

Override: Capacity of subclasses to modify parent methods.

```
class Users
{
      public void whoAmI()
      {
      System.out.println("User");
      }
}
```

```
class Student extends Users
{
      @Override
      public void whoAmI()
      {
      System.out.println("Student");
      }
}
```

Define the layout in XML

```xml
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
</android.support.constraint.ConstraintLayout>
```

This layout has 6 attributes. The first 3, `xmlns:android, xmlns:app & xmlns:tools` are declarations of **XML namespaces** that we will use in this file (this type of parameter only needs to be specified in the first element). The next two allow you to define the *width and height* of the **view** (screen). In the example, **view wants to be as big as its parent**. The last attribute indicates the Activity associated with this layout.

Declare main Activity in the manifest

```
<activity android:name=".MainActivity">
   <intent-filter>
       <action android:name="android.intent.action.MAIN" />
       <category android:name="android.intent.category.LAUNCHER" />
   </intent-filter>
</activity>
```

⇒ MainActivity needs to include an intent filter to start from the launcher.

Connecting the Activity with the Layout

R is an object created in execution time using the resource *activity_main.xml*.

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

Contents in **R file** can be seen as *identifiers* informing the resource manager what data needs to load. They are going to be created on **demand**.

You can see this file:

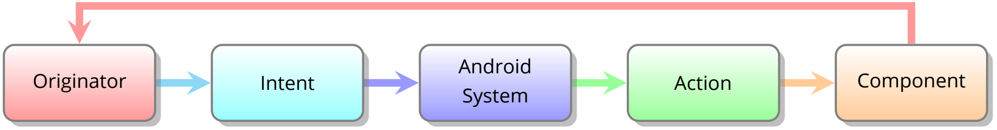
```
app\build\generated\not_namespaced_r_class_sources\
debug\processDebugResources
\r\<package_name_tree>\R.java
```

Intents

Intents

Definition

An **Intent** is a description of an operation to be performed. It is an object used to request an action from another Android *component*.



Start an Activity

- Click a button an navigate to another activity.

Start a Service

- Initiate downloading a file in background.

Deliver a broadcast

- The system informs everybody that the phone is charging.

How to start an Activity with an explicit intent

Create an intent and use it to start an activity

- **action**:The general action to be performed, such as *ACTION_VIEW, ACTION_EDIT, ACTION_MAIN*...
- **data**: The data to operate on, such as a person record in the contacts database, is expressed as a Uri.

```
Intent intent = new Intent(action,uri)
startActivity(intent)
```

```
//MainActivity
Intent intent = new Intent(
    this, Activity2.class);
startActivity(intent);
```

```
Uri uri = Uri.parse(
    "http://www.google.com");
Intent it = new Intent(
    Intent.ACTION_VIEW,uri);
startActivity(it);
```

Sending and retrieving data

```java
// A web page URL
intent.setData(
    Uri.parse("http://www.google.com"));
// a Sample file URI
intent.setData(
     Uri.fromFile(new File("/sdcard/sample.jpg")));
```

Sending and retrieving data using extras

Activity (sender):

- Create the **Intent** object
- Put data or extras into that **Intent**
- Start the new Activity with *startActivity()"' Activity (receiver):* Get the **Intent** object, the Activity was started with
- Retrieve the data or extras from the **Intent** object

```java
// Sender
Intent intent = new Intent(
    this, Activity2.class);
intent.putExtra(
    "argumentKey", "argumentValue");
startActivity(intent);
// Receiver
Bundle extras = getIntent().getExtras();
String s = extras.getString(
    "stringArgumentKey");
int i = extras.getInt(
    "integerArgumentKey");
```

⇒ Extras is a collection of key-value pairs

Returning data to the starting activity (First Activity)

```java
public static final int SECOND_REQUEST = 1;
Intent intent = new Intent(this, SecondActivity.class);
startActivityForResult(intent, SECOND_REQUEST);

public void onActivityResult(int requestCode,
                             int resultCode, Intent data) {
  super.onActivityResult(requestCode, resultCode, data);
  if (requestCode == SECOND_REQUEST) { // Identify activity
    if (resultCode == RESULT_OK) { // Activity succeeded
      String reply = data.getStringExtra(SecondActivity.EXTRA_REPLY);
      // … do something with the data
 }}}
```

⇒ startActivityForResult(intent, requestCode): start **SecondActivity**, assigns it identifier(**requestCode**), return data via *extras* and when **SecondActivity** is done, return to **FirstActivity (this)** and executes *onActivityResult()* callback. The **requestCode** is to identify which activity is returning.

Returning data to the starting activity (Second Activity)

```
 // Create an intent
Intent intent = new Intent();
// Put the data to return into the extra
intent.putExtra(EXTRA_REPLY, reply);
// Set the activity's result to RESULT_OK
setResult(RESULT_OK, intent);
// Finish the current activity
finish();
```

Getting a result from an activity: New Way (Android X)

To get a result from an activity using the new way, we need to migrate our project to AndroidX. **refactor > migrate to androidx**. Then it would help if you edited the dependencies:

```
    implementation 'androidx.appcompat:appcompat:1.4.1'
    implementation 'androidx.activity:activity:1.4.0'
```

Why old way is deprecated in androidx?

⇒ Starting another activity, whether one within your app or from another app, doesn't need to be a one-way operation.

⇒ New way helps reduce the complexity we face when we call Activity from a fragment or another activity.

⇒ Easily ask for any permission and get a callback.

What is AndroidX?

AndroidX is a significant improvement to the original Android Support Library, which is no longer maintained. androidx packages fully replace the Support Library by providing feature parity and new libraries.

Definition

The androidx namespace comprises the Android Jetpack libraries. Like the Support Library, libraries in the androidx namespace ship separately from the Android platform and provide backward compatibility across Android releases.

- All packages in AndroidX live in a consistent namespace, starting with the string androidx. The Support Library packages have been mapped into the corresponding androidx.
- The androidx packages use strict Semantic Versioning, starting with version 1.0.0. You can update AndroidX libraries in your project independently.
- Version 28.0.0 is the last release of the Support Library. There will be no more *android.support* library releases. All new feature development will be in the **androidx namespace**.

Getting a result from an activity: New Way (Android X)

```
ActivityResultLauncher<Intent> someActivityResultLauncher = registerForActivityResult(
    new ActivityResultContracts.StartActivityForResult(),
    new ActivityResultCallback<ActivityResult>() {
    @Override
    public void onActivityResult(ActivityResult result) {
        if (result.getResultCode() == Activity.RESULT_OK) {
            Intent data = result.getData();
            //doSomeOperations();
            }
        }});
public void openSomeActivityForResult() {
    Intent intent = new Intent(this, SecondActivity.class);
    someActivityResultLauncher.launch(intent);
}
```

Getting a result from an activity: New Way (Android X)

⇒ While the underlying **startActivityForResult()** and **onActivityResult()** APIs are available on the Activity class on all API levels, it is strongly recommended to use the *Activity Result APIs* introduced in **AndroidX**.

⇒ While it is safe to call **registerForActivityResult()** before your fragment or Activity is created, you cannot launch the **ActivityResultLauncher** until the fragment or Activity's Lifecycle has reached **CREATED**.

⇒ Since your process and Activity can be destroyed between when you call *launch()* and when the **onActivityResult()** callback is triggered, any additional state needed to handle the result must be saved and restored separately from these APIs.

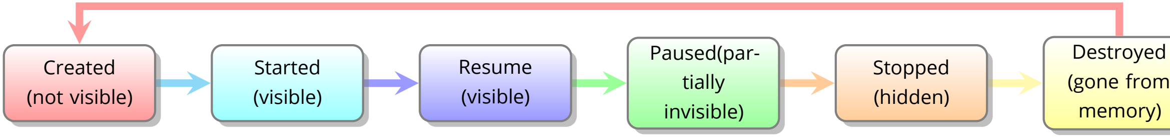
Activity lifecycle

What is the purpose

The main purpose is to define how your Activity behaves when the user leaves and re-enters the Activity.

Exemple

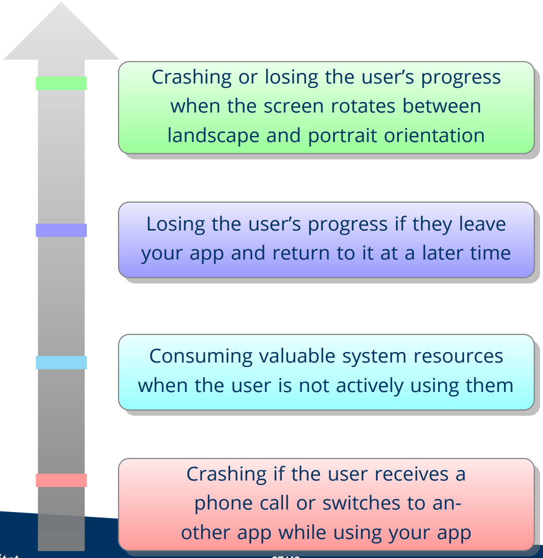
Streamlining a song, you might pause and terminate the network connection when the user switches to another app. When the user returns, you can reconnect to the network and allow the user to resume the song from the same place they left.



⇒ State changes are triggered by user action, configuration changes such as device rotation, or system action.

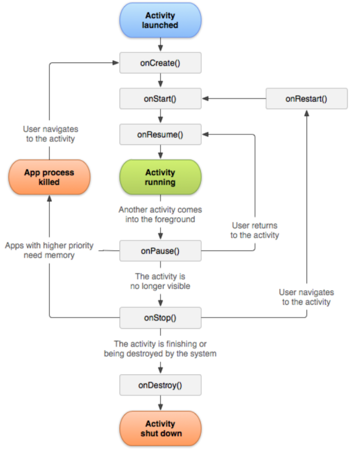
When to use

You must ensure that your app avoids:


Crashing or losing the user's progress when the screen rotates between landscape and portrait orientation
Losing the user's progress if they leave your app and return to it at a later time
Consuming valuable system resources when the user is not actively using them
Crashing if the user receives a phone call or switches to another app while using your app

Activity Life Cycle

- **onCreate**(Bundle savedInstanceState): static initialization
- **onStart()**: when Activity (screen) is becoming visible
- **onRestart()**: called if Activity was stopped (calls onStart())
- **onResume()**: start to interact with user
- **onPause()**: about to resume PREVIOUS Activity
- **onStop()**: no longer visible, but still exists and all state info preserved
- **onDestroy()** final call before Android system destroys Activity


Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

OnCreate

```java
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // The activity is being created.
}
```

- Basic applications startup logic.
- Bind data to lists.
- Associate the Activity with ViewModel.
- Fundamental setup for the Activity (XML layout file).

OnStart

```java
@Override
protected void onStart() {
    super.onStart();
    // The activity is about to become visible.
}
```

- Makes the Activity visible to the user.
- Init the code that maintains the UX.
- Method completes very quickly, and, as with the Created state, the Activity does not stay resident in the Started state.

onResume

```
@Override
protected void onResume() {
    super.onResume();
    // The activity has become visible
    // it is now "resumed"
}
```

- App interacts with the user.
- Stays in this state until something happens to take focus away from the app.
- If the Activity returns to the Resumed state from the Paused state, the system again calls onResume() method. For this reason, you should implement onResume() to initialize components that you release during onPause() and perform any other initializations that must occur each time the Activity enters the Resumed state.

onPause

```java
@Override
protected void onPause() {
    super.onPause();
    // Another activity is taking focus
    // this activity is about to be "paused"
}
```

- Indicator that the user is leaving your Activity.
- Release system resources handles sensors (like GPS) or any resources that may affect battery life while your Activity is paused and the user does not need them.
- Do not save data or make network calls or execute database transactions.

onStop

```java
@Override
protected void onStop() {
    super.onStop();
    // The activity is no longer visible
    // it is now "stopped"
}
```

- Activity is no longer visible to the user.
- Save data, make network calls, execute database transactions.
- Heavy-load shut down operations.

onDestroy

```java
@Override
protected void onDestroy() {
    super.onDestroy();
    // The activity is about to be destroyed.
}
```

- Activity is finishing (due to the user completely dismissing the Activity or due to finish() being called on the Activity)
- The system is temporarily destroying the Activity due to a configuration change (such as device rotation or multi-window mode)

Challenge 1: TracerActivity

Goal. The objective of this Activity is to make an activity called Tracer Activity aimed to write into the logs all the activity status changes. This class must be reusable to trace any Activity in the project.

TIPS

1. To write in the log: `Log.d(TAG, "onStart() state.")`;
2. *TAG* must be the activity name.
3. To show a toast:

```
Toast.makeText(getApplicationContext(),
    TAG + " -> onStart() state.", Toast.LENGTH_LONG).show();
```

4. Check how the app moves between the different states.

Work in pairs—5 minutes of coding. The first to complete the challenge will receive a positive.

Activity instance and state

What happens if the configuration is changed?

The configuration can be changed while the user is interacting with the app. Different events, users or systems triggered can cause the transition from one state to another.

Exemple

The user changes the orientation (portrait, landscape).

When this change occurs, the activity is **destroyed** (onPause(),onStop(),onDestroy()) and then a new instance is **created** (onCreate(),onStart(),onResume()). The `onSaveInstanceState()` method allow the persistent local storage to preserve an activity's UI state across configuration changes.

onSaveInstanceState()

The system only saves the state of views with unique ID *(android:id)* such as text entered into EditText and Intent (data and extras). You are responsible for saving other Activity and user progress data.

```
@Override
public void onSaveInstanceState(Bundle outState) {
    super.onSaveInstanceState(outState);
    outState.putParcelable("parcelable", model);
}
```

⇒ Called by Android runtime when there is a possibility the Activity may be destroyed.

⇒ Saves data only for this instance of the Activity during the current session

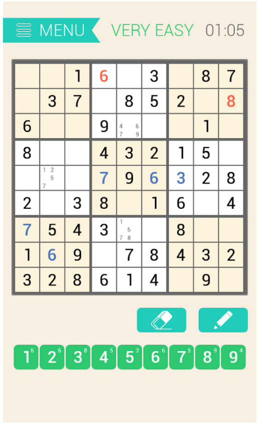
onRestoreInstanceState(Bundle state)

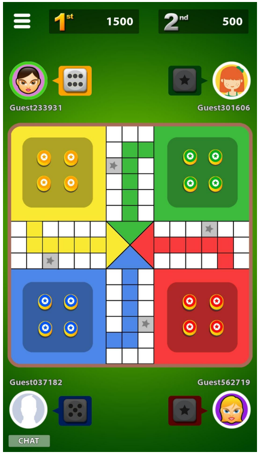
```
@Override
public void onRestoreInstanceState (Bundle mySavedState) {
   super.onRestoreInstanceState(mySavedState);

   if (mySavedState != null) {
       Model model = savedInstanceState.getParcelable("parcelable");
       if (model != null)
           //do sth
   }
}
```

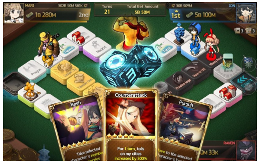
When you stop and restart a new app session, the Activity instance states are lost, and your activities will revert to their default appearance. If you need to save user data between app sessions, use shared preferences or a database.

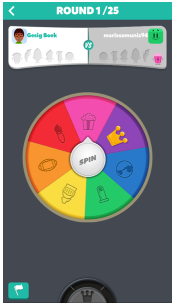
Homework

Task: Project (Sprint I)









That's all

QUESTIONS?

About me

www — jordimateofornes.com
github — github.com/JordiMateo
twitter — @MatForJordi
gdc — Distributed computation group