



## 7. Lesson 20/03/23

### Activities and Intents

#### Contents

- Activities
- Defining *Activities*
- Starting a new *Activity* with an *Intent*
- Passing data between activities with extras
- Navigating between activities

### Activities (high-level views)

#### What is an Activity?

An *Activity* is an application component. It represents one window (hierarchy of views).

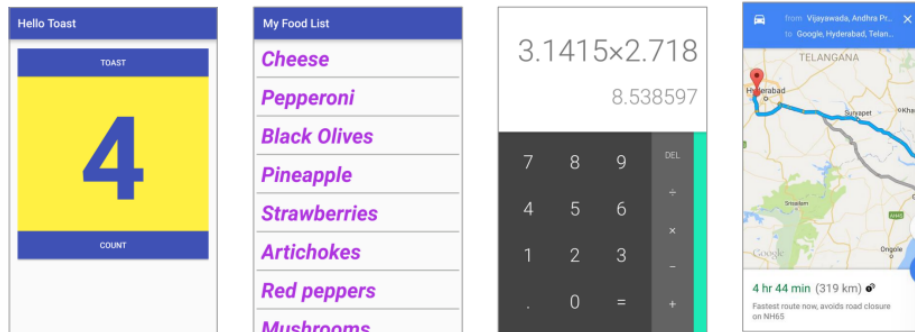
- Typically fills the screen
  - But can be embedded in other Activity
  - Or can appear as floating window

An activity represents a single screen in the app with an interface the user can interact with

#### What does an Activity?

- Apps are often a collection of activities
  - that you create yourself
  - that you reuse from other apps
- Handle user interactions, such as
  - button clicks
  - text entry
  - login verification

## Examples of activities



## Apps and activities

- First Activity user sees is typically called "main activity"
- Activities are loosely tied together to make up an app
- Activities can be organized in parent-child relationships in the Android manifest to aid navigation

## Life cycle of an Activity

- Has a life cycle: an activity is
  - Created
  - Started
  - Runs
  - Paused
  - Resumed
  - Stopped
  - Destroyed

## Layouts and Activities

- An Activity typically has a UI layout
- Layout is usually defined in one or more XML files
- Activity "inflates" layout as part of being created

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

# Implementing Activities

## Implement new Activities

You can add new activities:

- when creating a new project
- or choosing File > New > Activity

The wizard automatically performs the following steps: (done automatically by Android Studio)

1. Define layout in XML
2. Define Activity Java Class
3. Connect Activity with Layout
4. Declare Activity in the Android Manifest

## 1. Define Layout in XML

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Let's Shop for Food!" />
</RelativeLayout>
```

## 2. Define Activity Java class

When creating a new project the MainActivity is, by default, a subclass of the AppCompatActivity class.

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

This **allows to use up-to-date Android app** features such as the app bar and Material Design while still **enabling the app to be compatible** with devices running **older versions of Android**. 🗣️

The first task in the implementation of an Activity subclass is to implement the standard Activity lifecycle callback methods (such as onCreate()) to handle the state changes for your Activity.

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

These state changes include things such as when the Activity is **created**, **stopped**, **resumed**, or **destroyed**. 🗣️

The one required callback that an app must implement is the onCreate() method.

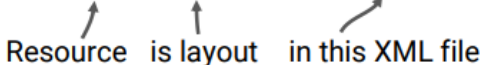
The system calls this method when it creates the Activity, and all the essential components of your Activity should be initialized here.

### 3. Connect Activity with Layout

The onCreate() method calls setContentView() with the path to a layout file.

The system creates all the initial views from the specified layout and adds them to your Activity. This is often referred to as inflating the layout.

**setContentView(R.layout.activity\_main);**



### 4. Declare Activity in Android manifest

Each Activity in an app must be declared in the AndroidManifest.xml file with the <activity> element, inside the <application> section.

The only required attribute is `android:name`, which specifies the class name for the Activity (such as `MainActivity`)

```
<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 `intent-filter` to start from launcher.

The `<action>` element specifies that this is the "main" entry point to the app.

## Add Activities to App

Each Activity of an app and its associated layout file is supplied by an Activity template in Android Studio such as Empty Activity or Basic Activity.

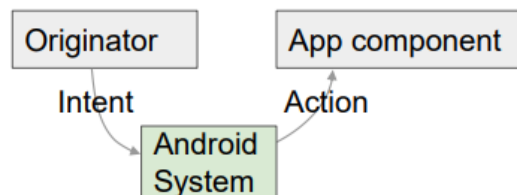
You can add a new Activity to your project by choosing: **File > New > Activity**.

## Intents

### What is an intent?

An **Intent** is a description of an operation to be performed.

An Intent is an object used to request an action from another app component via the Android system.



### Starting the main activity

1. When the app is first started from the device home screen.
2. The Android runtime sends an Intent to the app to start the app's main activity.

```

<activity android:name=".MainActivity" >

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

</activity>

```

## What can intents do?

- Start an *Activity*
- Start a *Service*
- Deliver *Broadcast*

## Explicit and implicit intents

### Explicit Intent

- Starts a specific *Activity*

### Implicit Intent

- Asks system to find an *Activity* that can handle this request

## Starting Activities

### Start an Activity with an explicit intent

To start a **specific Activity** use an explicit *Intent*

1. Create an *Intent*

```
Intent intent = new Intent(this, ActivityName.class);
```

2. Use the *Intent* to start the *Activity*

```
startActivity(intent);
```

Example:

```
Intent messageIntent = new Intent(this, ShowMessageActivity.class);
startActivity(messageIntent);
```

## Start an Activity with implicit intent

To ask Android to find an Activity to handle your request, use an **implicit Intent**:

1. Create an *Intent*

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

2. Use the *Intent* to start the Activity

```
startActivity(intent);
```

### Example

#### Show a web page

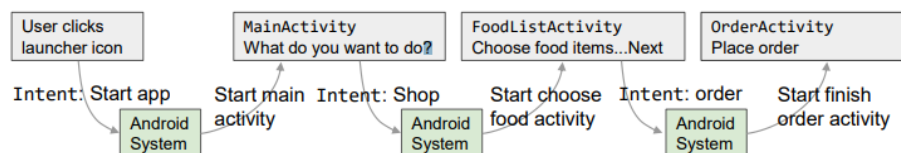
```
Uri uri = Uri.parse("http://www.google.com");  
Intent it = new Intent(Intent.ACTION_VIEW, uri);  
startActivity(it);
```

#### Dial a phone number

```
Uri uri = Uri.parse("tel:8005551234");  
Intent it = new Intent(Intent.ACTION_DIAL, uri);  
startActivity(it);
```

## How Activities Run

- All *Activity* instances are managed by the Android runtime
- Started by an "Intent", a message to the Android runtime to run an activity



## Sending and Receiving Data

### Sending data with intents

In addition to open a new activity, with an intent it is **possible to pass data from one Activity to another**.

It is possible to use **Intent data** or **Intent extras**:

- **Data**: one piece of information whose data location can be represented by an URI.
- **Extras**: one or more pieces of informations as a collection of key-values pairs.

There are several key differences between data and extras that determine which you should use.

## Intent data

The **Intent data can hold only one piece of information**: a URI representing the location of the data you want to operate on.

The URI could be:

- a web page URL
- a telephone number
- a geographic location
- any other custom URI you define

## When use Intent data

Use the Intent data field when:

- You **only have one piece of information that you need to send** to the started Activity
- That **information is a data location that can be represented by a URI**

## Extras data

Intent extras are for any other arbitrary data you want to pass to the started Activity.

Intent extras are stored in a Bundle **object as key and value pairs**.

A **Bundle is a map**, optimized for Android, in which

- A **key** is a **string**
- A **value** can be **any primitive or object type**

To put data into the Intent extras you can

- **Use** any of the Intent class **putExtra()** methods
- **Create your own Bundle** + put the Bundle into the Intent with **putExtras()**.



## When use Extras data

Use the **Intent extras**:

- If you want to pass more than one piece of information to the started Activity.
- If any of the information you want to pass is not expressible by a URI.

**NOTE: Intent data and extras are not exclusive!**

You can use data for a URI and extras for any additional information the started Activity needs to process the data in that URI.

## Passing data from one Activity to another

**For sending data to an Activity:**

1. Create the Intent Object
2. Put data or extras into that Intent
3. Start the new Activity

**For receiving data from an Activity:**

1. Get the Intent object, the Activity was started with
2. Retrieve the data or extras from the Intent object

## Sending data - Create the Intent Object

**Step 1:** Create the Intent object

```
Intent messageIntent = new Intent(this, ShowMessageActivity.class);
```

## Sending data - Put data into that Intent

**Step 2:** Put data into the Intent

Use the setData() method with a URI object to add it to the Intent.

```
// 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 Data - Start the new Activity

**Step 3:** Start the new Activity

Keep in mind that:

- The data field can only contain a single URI
- If you call setData() multiple times only the last value is used
- You should use Intent extras to include additional information

After you've added the data, you can start the new Activity with the Intent:

```
startActivity(messageIntent);
```

## Sending Extras (with multiple putExtra)

**Step 1:** Create the Intent object

**Step 2:** Put extras into that Intent

Use a putExtra() method with a key to put data into the Intent extras. The Intent class defines many putExtra() methods for different kinds of data.

```
putExtra(String name, int value)
⇒ intent.putExtra("level", 406);

putExtra(String name, String[] value)
⇒ String[] foodList = {"Rice", "Beans", "Fruit"};
intent.putExtra("food", foodList);
```

**Step 3:**

```
startActivity(messageIntent);
```

## Example

```
public static final String EXTRA_MESSAGE_KEY =
    "com.example.android.twoactivities.extra.MESSAGE";
```

Conventionally you define Intent extra keys as static variables with names that begin with EXTRA\_. To guarantee that the key is unique, the string value for the key itself should be prefixed with your app's fully qualified class name.

```
Intent intent = new Intent(this, SecondActivity.class);
String message = "Hello Activity!";
intent.putExtra(EXTRA_MESSAGE_KEY, message);
startActivity(intent);
```

## Sending Extras (with a Bundle)

If lots of data:

- first create a bundle and
- pass the bundle

Bundle defines many "put" methods for different kinds of primitive data as well as objects that implement Android's Parcelable interface or Java's Serializable.

## Get data from intents

When you start an Activity with an Intent, the started Activity has access to the Intent and the data it contains.

To retrieve the Intent the Activity (or other component) was started with, use the getIntent() method:

```
Intent intent = getIntent();
```

**Data:** Use getData() to get the URI from that Intent:

```
getData();
⇒ Uri locationUri = intent.getData();
```

**Extras:** Use one of the getExtra() methods to extract extra data out of the Intent object:

```
int getIntExtra (String name, int defaultValue)
⇒ int level = intent.getIntExtra("level", 0);

Bundle bundle = intent.getExtras();
⇒ Get all the data at once as a bundle.
```