Week 8: Android Lesson

[Refer to Lecture Note Files: Lesson0 and Lesson1]

Agenda

- Android Introduction and Project Setup
- Modify App Layout (XML)
- View and View Attributes
- Resources and R Class (Assets Identity)
- Referring View from Java Code (findViewById)
- Java: Nested Class
- Java: Anonymous Class
- Implement a callback

Android Introduction

- AndroidManifest.xml → Contains specifications of the app including components and permissions
- java folder → Contains source code for logic/controller and tests
- res folder → non-code resources: images, layout, app components, strings, icon, etc

Modify App Layout (XML)

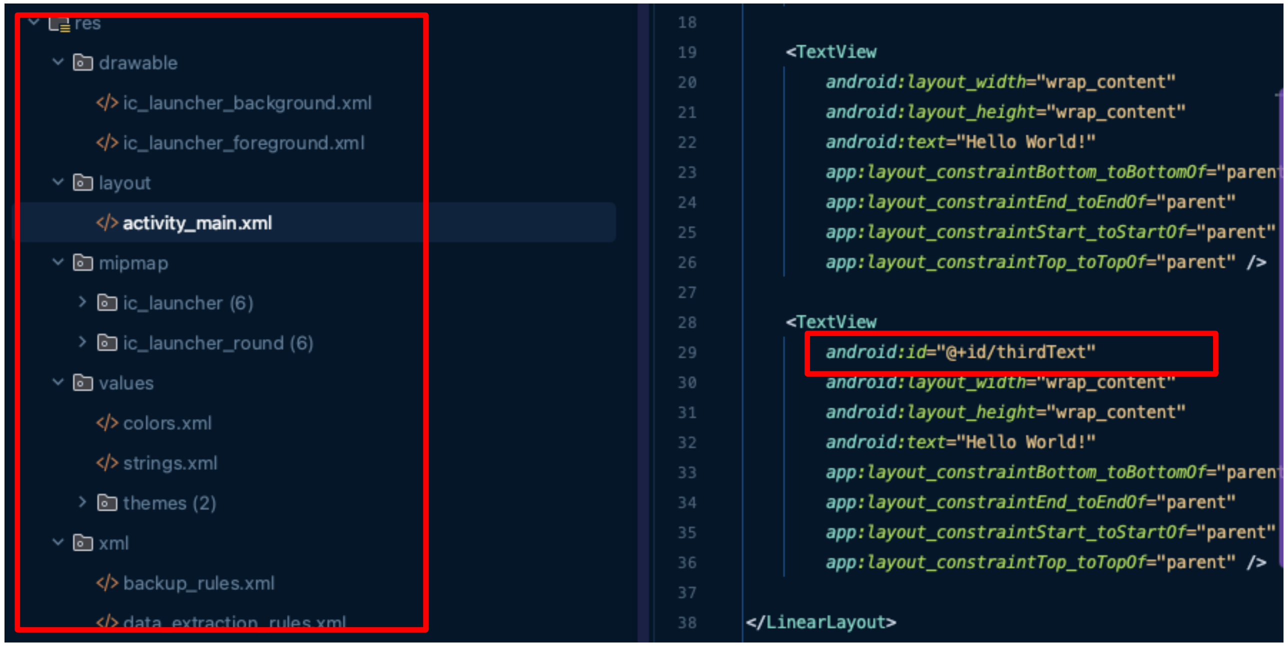
- Change ConstraintLayout to LinearLayout
- Set orientation attribute

View and View Attributes

- View is the building block of UI, e.g. buttons, text, input, image, etc
- Adjust the text, size, and alignment using the following attributes:
  - *android:layout_width*
  - *android:layout_height*
  - *android:text*
  - *android:layout_gravity*
  - *android:gravity*
- To set the ID of a View:
  - *android:id*
- Learn more here: https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/unit-1-get-started/lesson-1-build-your-first-app/1-2-c-layouts-and-resources-for-the-ui/1-2-c-layouts-and-resources-for-the-ui.html

R Class (Assets Identity)

- *R* class is generated when the app is compiled
- It contains all resource IDs (from */res* folder)



```
res
  drawable
    ic_launcher_background.xml
    ic_launcher_foreground.xml
  layout
    activity_main.xml
  mipmap
    ic_launcher (6)
    ic_launcher_round (6)
  values
    colors.xml
    strings.xml
    themes (2)
  xml
    backup_rules.xml
    data_extraction_rules.xml
```

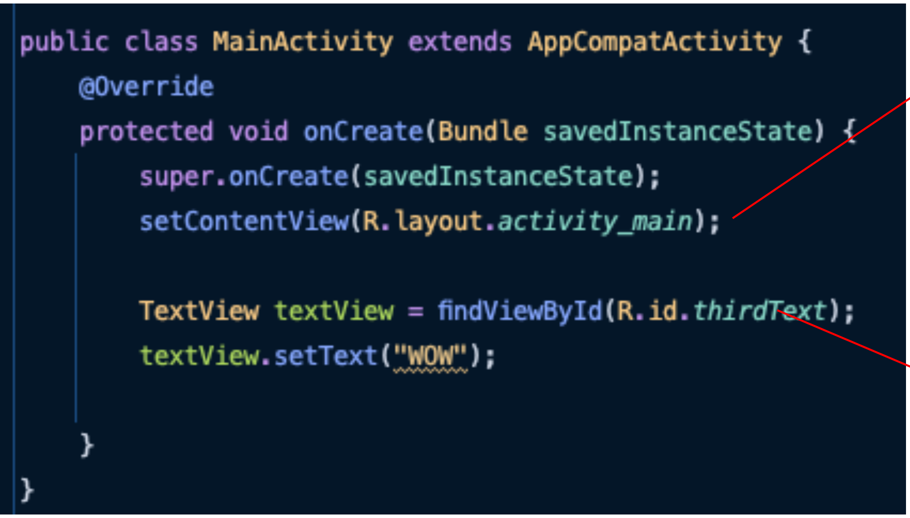
```xml

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="paren
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/thirdText"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="paren
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</LinearLayout>
```

https://developer.android.com/guide/topics/resources/providing-resources#Accessing

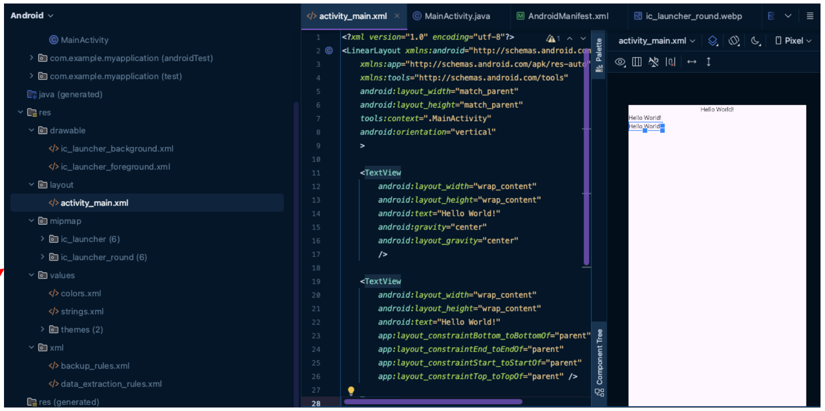
Referring View from Java Code (findViewById)

- *setContentView(R.layout.activity_main)* = Inflating all view components from activity_main.xml to fill your screen with the defined views.
- Android reads the XML code in the layout file and instantiates objects in the memory that represent each of the widgets on the Activity.
- Use *findViewById* to refer to a specific View (which has an ID when it was defined in xml file)

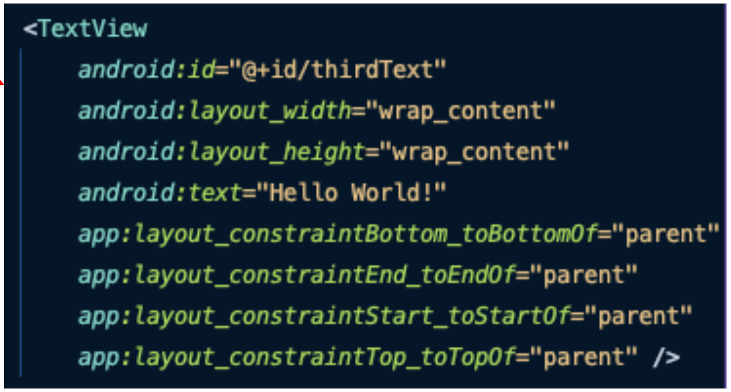
Accessing Resources



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

        TextView textView = findViewById(R.id.thirdText);
        textView.setText("WOW");

    }
}
```



```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com
    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:orientation="vertical"
    >

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        android:gravity="center"
        android:layout_gravity="center"
        />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
```



```
<TextView
    android:id="@+id/thirdText"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello World!"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```

Java: Nested Class

- class inside another class. You can have interface inside a class too.
- Nested classes enable you to logically group classes that are only used in **one place**, increase the use of encapsulation, and create more readable and maintainable code.
- Refer to lecture note for the detailed examples

https://docs.oracle.com/javase/tutorial/java/javaOO/whentouse.html

Java: Anonymous Class

- To avoid declaring too many classes
- Usually for declaring a class that is only used once
- For example, in Android, you need to pass an object when you define what a button should do. Imagine you have 10 buttons that perform different things. Instead of defining 10 different classes, you can use anonymous class

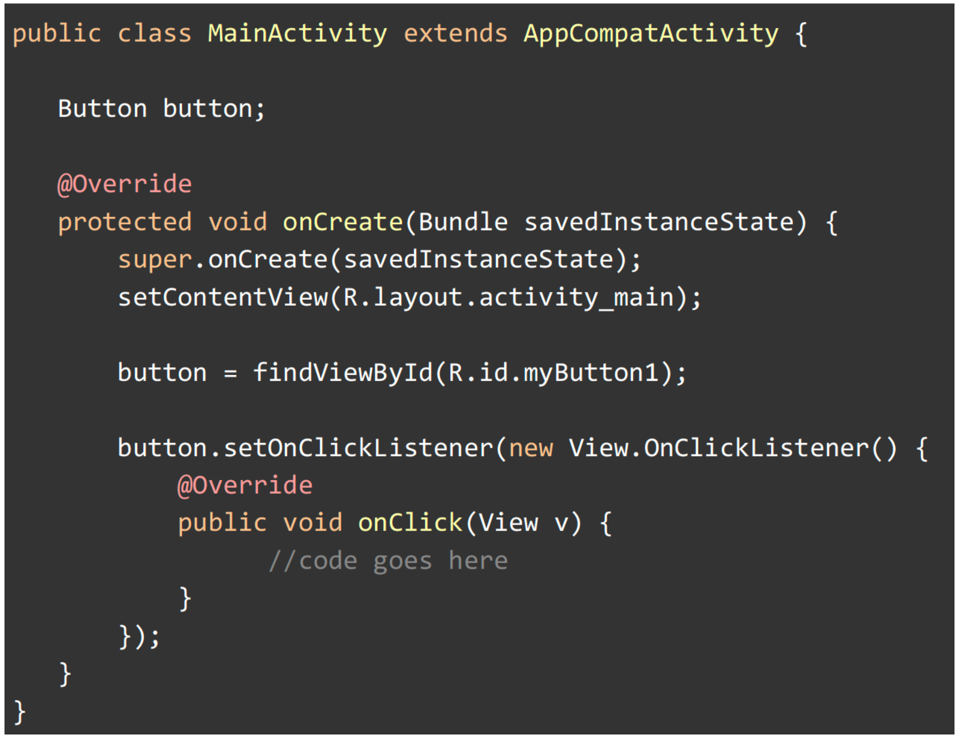
Implement Callback

- What is callback? It is a **function** passed as an **argument** to another function: It's not executed immediately but "called back" later when a specific event or condition occurs.
- Useful for:
  - Asynchronous Operations: Handling events (e.g., button clicks, network requests), timers, animations, I/O tasks.
  - Event-Driven Programming: Reacting to user interactions, system events, sensor readings.

https://stackoverflow.com/questions/824234/what-is-a-callback-function

Implement Callback

- Example: using anonymous class for handling button-click event. The onClick callback will be invoked when the button is clicked.



```
public class MainActivity extends AppCompatActivity {

   Button button;

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

       button = findViewById(R.id.myButton1);

       button.setOnClickListener(new View.OnClickListener() {
           @Override
           public void onClick(View v) {
                 //code goes here
           }
       });
   }
}
```

You can implement using inner class also. But anonymous class is preferred in most cases