COMP-304
Winter 2023

Anatomy and Life Cycle of Android Applications

Objectives:

- Explain Android **activities**, **fragments**, **intents**.
- Explain application, activity, and fragment **life cycles**.
- Create and use activities.
- Apply intents to call built-in applications and pass information to other activities.
- Create and use fragments.

Activities

- **Activity:** is an application **component that provides a screen with which users can interact in order to do something**, such as dial the phone, take a photo, send an email, or view a map.
  - Each activity is given a window in which to draw its user interface.
- To create an activity, you must *create a subclass of Activity*:

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }
}
```

- **AppCompatActivity** is a subclass of Android's Activity class that provides compatibility support for older versions of Android.

Activities

- To assign a UI to an Activity, call setContentView from the onCreate method:

  setContentView(R.layout.activity_main)

- The argument of setContentView method is a resource ID for a layout defined in an external resource named activity_main.xml.
- To use an Activity in your application, you need to register it in the manifest.
  - Add a new **activity tag** within the application node of the manifest
    - the activity tag includes **attributes for metadata**, such as the label, icon, required permissions, and themes used by the Activity.

```
<activity
    android:name=". MainActivity">
    ……..
</activity>
```

Activities

- For an Activity to be available from the application launcher, it must include an Intent Filter listening for the MAIN action and the LAUNCHER category, as highlighted below:

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

Using the AppCompatActivity

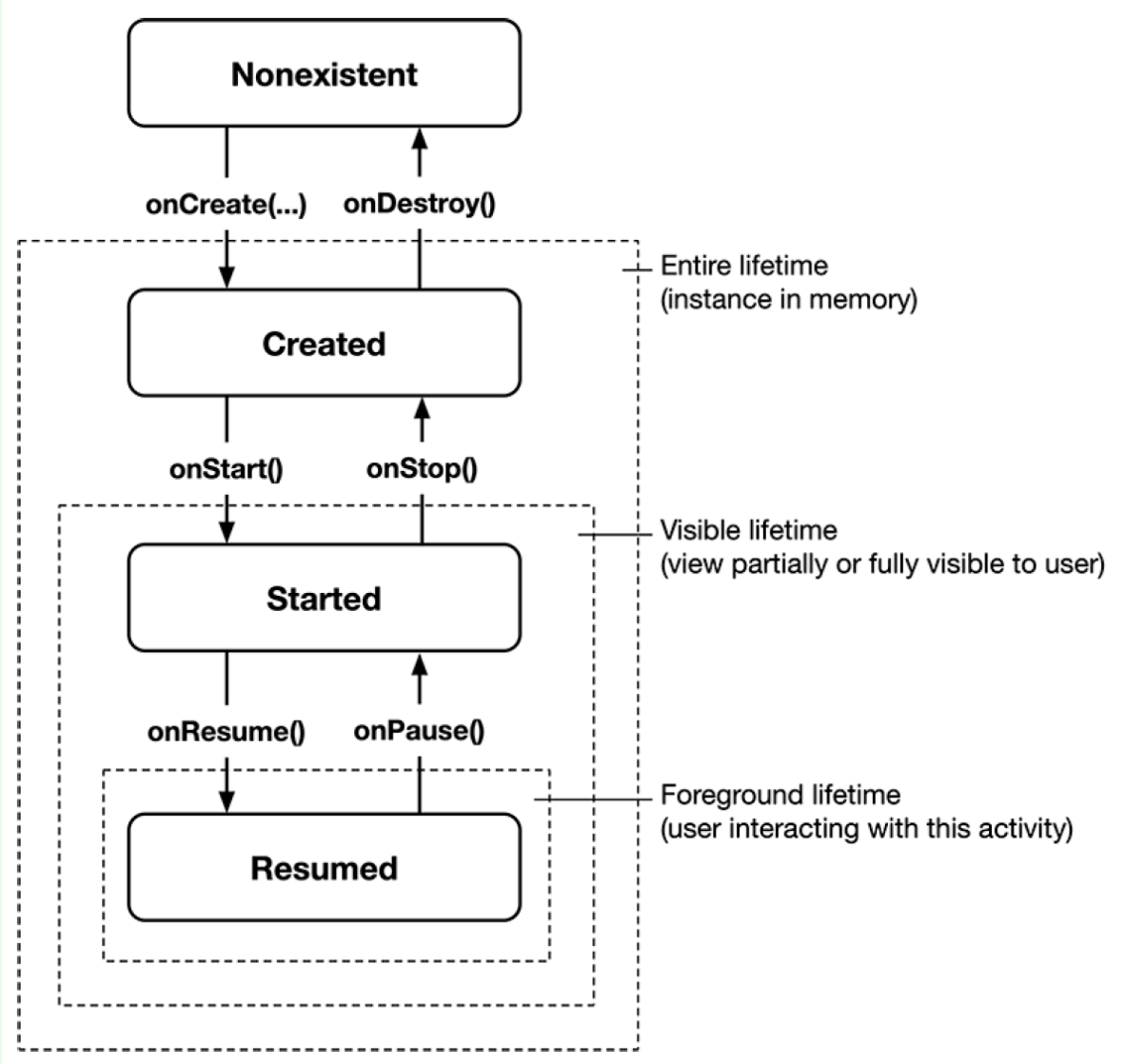
- The AppCompatActivity class is an Activity subclass available from the Android Support Library, and now from AndroidX.
- It provides ongoing, **backward compatibility** for features added to the Activity class in each new platform release.

```
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
class MainActivity : AppCompatActivity() {
    /** Called when the activity is first created. */
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
    }
}
```

Activity States and Lifecycle Callbacks

- Every instance of Activity has a **lifecycle**.
- During this lifecycle, an activity transitions between **four states**: **resumed**, **started**, **created**, and **nonexistent**.
- For each transition, there is an Activity function that notifies the activity of the change in its state.
- The names of these functions are given in the next slide.

Activity States and Lifecycle Callbacks



Activity States and Lifecycle Callbacks

- Activity States:

State	In memory?	Visible to user?	In foreground?
nonexistent	no	no	no
created	yes	no	no
started	yes	yes/partially*	no
resumed	yes	yes	yes

Activity States and Lifecycle Callbacks

- **Nonexistent** - represents an activity that **has not been launched** yet or an activity that was destroyed (by the user completely killing the app, for example).
  - For that reason, this state is sometimes referred to as the “destroyed” state.
  - is no instance in memory, and there is no associated view for the user to see or interact with.
- **Created** - represents an activity that **has an instance in memory** but whose view is not visible on the screen.
  - This state occurs in passing when the activity is first spinning up and reoccurs any time the view is fully out of view (such as when the user launches another full-screen activity to the foreground, navigates to the Home screen, or uses the overview screen to switch tasks).

Activity States and Lifecycle Callbacks

- **Started** - represents an activity that **has lost focus** but whose view is visible or partially visible.
  - An activity would be **partially visible**, for example, if the user launched a new dialog-themed or transparent activity on top of it.
  - An activity could also be **fully visible** but not in the foreground if the user is viewing two activities in multi-window mode (also called “split-screen mode”).
- **Resumed** - represents an activity that **is in memory**, **fully visible**, and **in the fore**ground.
  - It is usually the state of the activity the user is currently interacting with.
- Functions named in the figure on slide 8, are often called **lifecycle callbacks**.

Monitoring State Changes

- When an activity transitions into and out of the different states, it is notified through various *callback or life cycle methods*.
- All the life cycle methods are hooks that you can override to do appropriate work when the state of your activity changes.
- The code on the next slide shows how to override the life cycle methods.

Lifecycle Callbacks Example

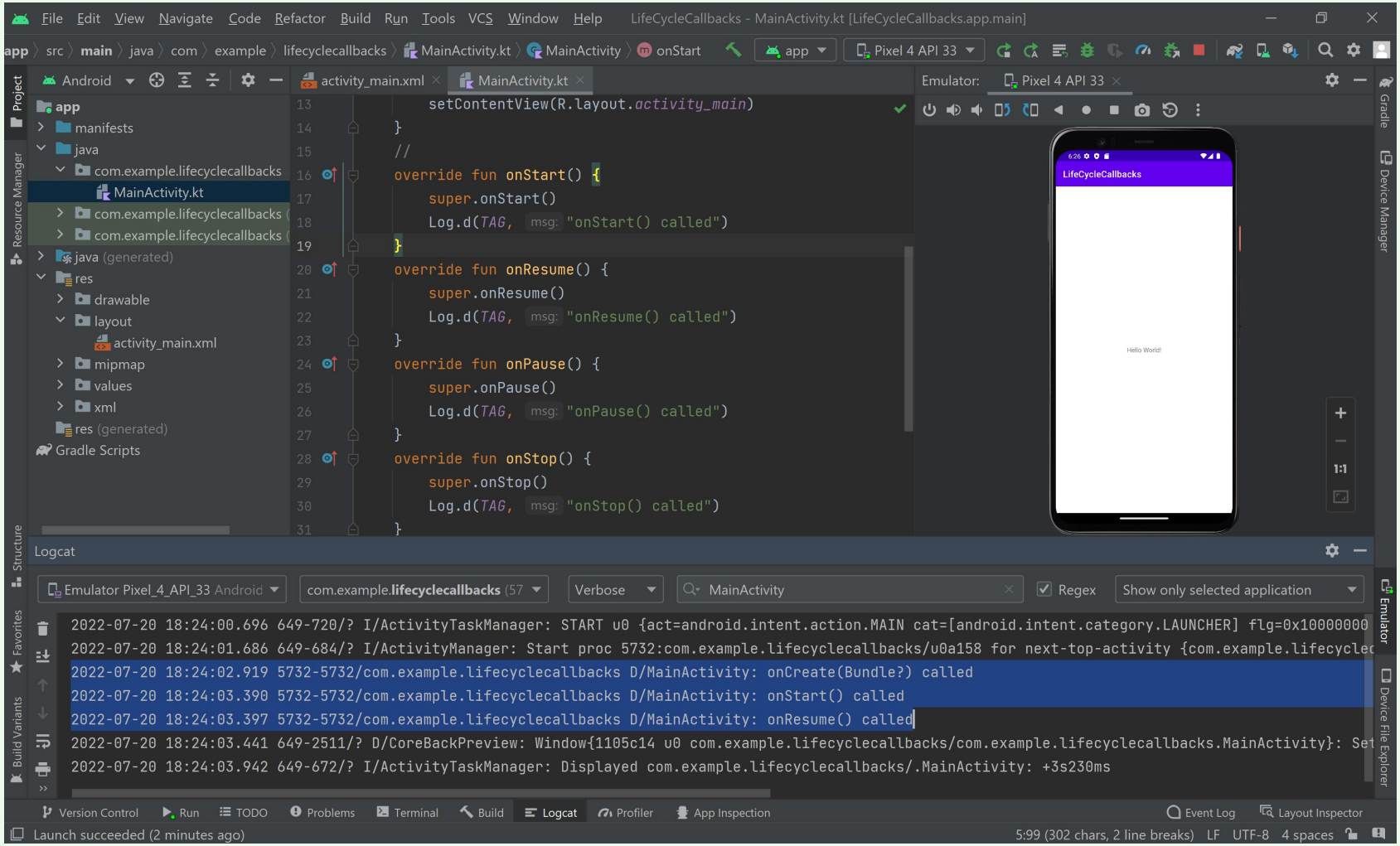
```
private const val TAG = "MainActivity"

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        Log.d(TAG, "onCreate(Bundle?) called")
        setContentView(R.layout.activity_main)
    }
    override fun onStart() {
        super.onStart()
        Log.d(TAG, "onStart() called")
    }
    override fun onResume() {
        super.onResume()
        Log.d(TAG, "onResume() called")
    }
```

Lifecycle Callbacks Example

```
    override fun onPause() {
        super.onPause()
        Log.d(TAG, "onPause() called")
    }
    override fun onStop() {
        super.onStop()
        Log.d(TAG, "onStop() called")
    }
    override fun onDestroy() {
        super.onDestroy()
        Log.d(TAG, "onDestroy() called")
    }
}
```

Lifecycle Callbacks Example



Managing Activity Transitions with Intents

- Android applications can have **multiple entry points**.
- There is **no main() function**, such as you find in iPhone development.
- A specific Activity can be designated as the main Activity to launch by default within the **AndroidManifest.xml** file
- Launching a New **Activity by Class Name**
  - You can start activities in several ways:
    - The simplest method is to use the Application **Context** object to call the **startActivity**() method, which takes a single parameter, an **Intent** object

Managing Activity Transitions with Intents

- **Intents** allow sending or receiving data from and to other activities or services.
- Intents are objects of type "android.content.**Intent**" and are used to send asynchronous messages within your application or between applications.
- The following code uses Intent to **launch an Activity** named ActivityB by its class:

```
val intent = Intent(this, ActivityB::class.java)
startActivity(intent)
```

- You can use the **Intent** structure to **pass data between Activities**

Creating Intents with Action and Data

- Intent objects are composed of two main parts:
  - the ***action*** *to be performed*
  - the ***data*** *to be acted upon*
- *You can also specify action/data pairs using* ***Intent*** **Action** types and **Uri** objects
- The most common action types are defined in the Intent class, including ACTION_MAIN (describes the main entry point of an Activity) and ACTION_EDIT (used in conjunction with a Uri to the data edited)

Launching an Activity Belonging to Another Application

- Here is an example of how to create a simple Intent with a predefined Action (ACTION_DIAL) to **launch the Phone Dialer** with a specific phone number to dial in the form of a simple Uri object:

```
Uri number = Uri.parse(tel:5555551212)
val dial = Intent(Intent.ACTION_DIAL, number)
startActivity(dial)
```

- This example uses action type of ACTION_VIEW followed by the URL of a web page:

```
val intent = Intent(Intent.ACTION_VIEW,
        Uri.parse("http://www.google.com"))
startActivity(intent)
```

Passing Additional Information Using Intents

- You can also include additional data in an Intent.
  - The **Extras** property of an Intent is stored in a **Bundle** object.
- The Intent class also has a number of helper methods for **getting and setting name/value pairs** for many common data types.
- For example, the following Intent includes two extra pieces of information - a string value and a boolean:

```
val intent = Intent(this, ActivityB::class.java)
intent.putExtra(“SomeStringData”,”Foo”)
intent.putExtra(“SomeBooleanData”,false);\
```

Intents Example

```
const val EXTRA_MESSAGE = "com.example.myfirstapp.MESSAGE"

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }
    /** Called when the user taps the Send button */
    fun sendMessage(view: View) {
        val editText = findViewById<EditText>(R.id.editText)
        val message = editText.text.toString()
        val intent = Intent(this, DisplayMessageActivity::class.java).apply {
            putExtra(EXTRA_MESSAGE, message)
        }
        startActivity(intent)
    }
}
```

Intents Example

```
class DisplayMessageActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_display_message)
        //
        // Get the Intent that started this activity and extract the string
        val message = intent.getStringExtra(EXTRA_MESSAGE)

        // Capture the layout's TextView and set the string as its text
        val textView = findViewById<TextView>(R.id.textView).apply {
            text = message
        }
    }

}
```

Intents Example

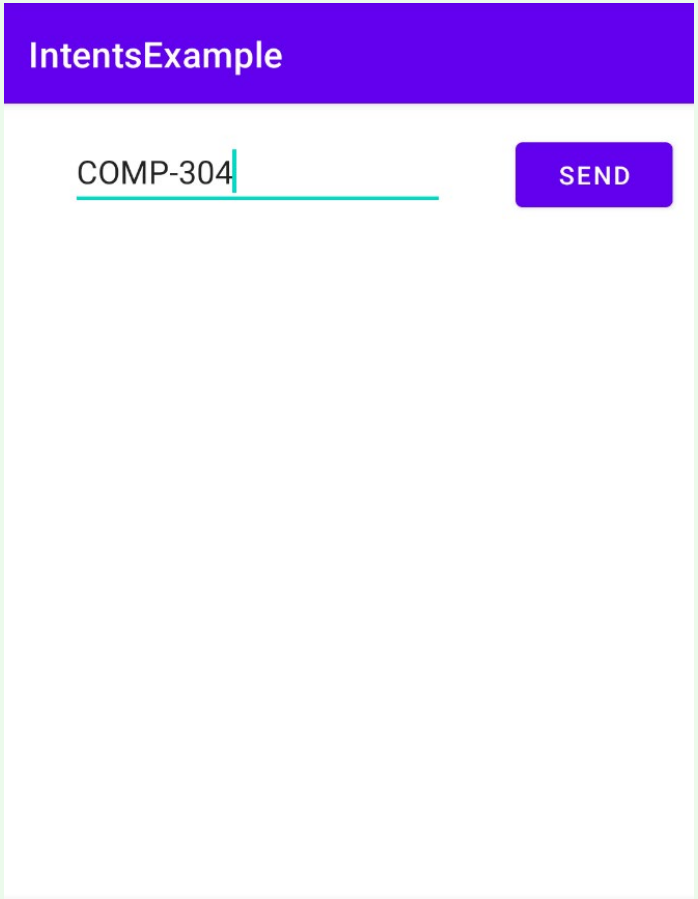
- Note the following Kotlin syntactic notations:
  - **::class.java** – used to get an instance of Class

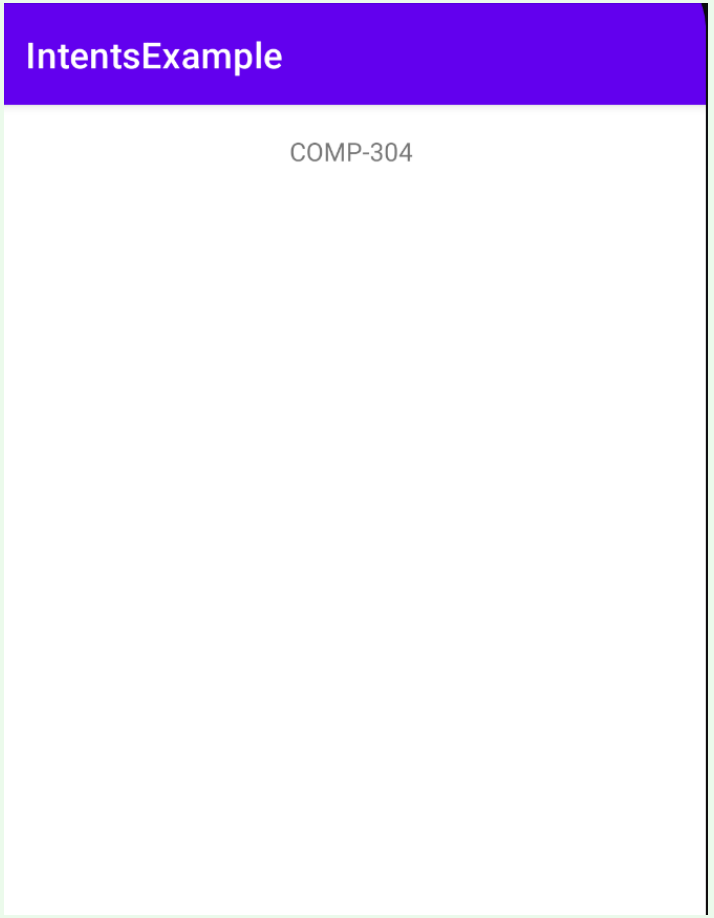
```
val intent = Intent(this, ActivityB::class.java)
```

  - **apply** – used to call the specified function block with this value as its receiver and returns this value.

```
val intent = Intent(this, DisplayMessageActivity::class.java).apply {
        putExtra(EXTRA_MESSAGE, message)
    }
```

Intents Example





Fragments

- Fragments are **mini activities** which are added to one or more activities.
  - Fragments can either contain a UI or not,
- Dividing your activity into Fragments, doing so can drastically **improve the flexibility of your UI** and make it easier for you to adapt your user experience for new device configurations.
- Extend the Fragment class to create a new Fragment, (optionally) defining the UI and implementing the functionality it encapsulates.
- If your Fragment does require a UI, override the onCreateView handler **to inflate and return** the required View hierarchy.

Fragments

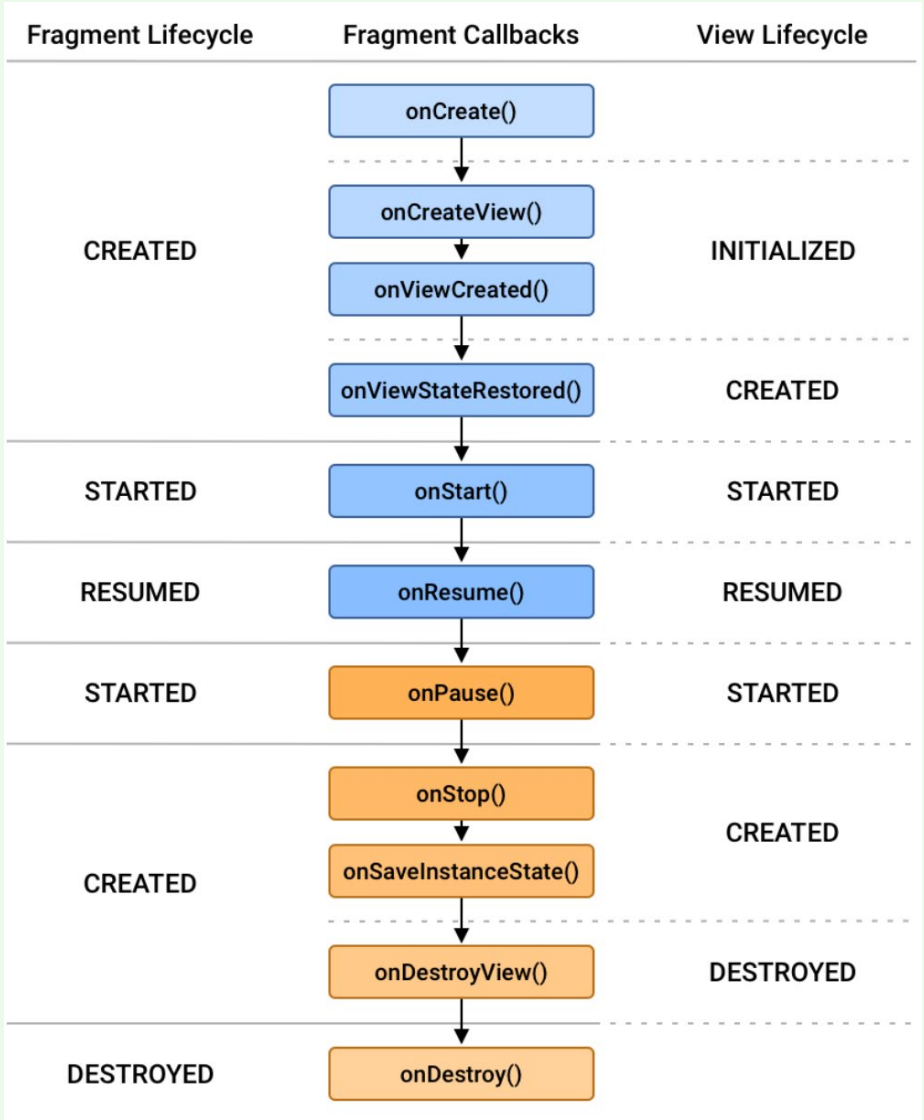
```
class AboutFragment : Fragment() {
    override fun onCreateView(inflater: LayoutInflater, container:
ViewGroup?, savedInstanceState: Bundle?): View? {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.fragment_about, container, false)
    }
}
```

- Fragments don't need to be registered in your manifest.
  - This is because Fragments can exist only when embedded into an Activity.

The Fragment Life Cycle

- Each Fragment instance has its own lifecycle.
- When a user navigates and interacts with your app, your fragments transition through various states in their lifecycle as they are added, removed, and enter or exit the screen.
- When a fragment is instantiated, it begins in the INITIALIZED state.
- For a fragment to transition through the rest of its lifecycle, it must be added to a **FragmentManager**.
- The FragmentManager is responsible for determining what state its fragment should be in and then moving them into that state.

The Fragment Life Cycle

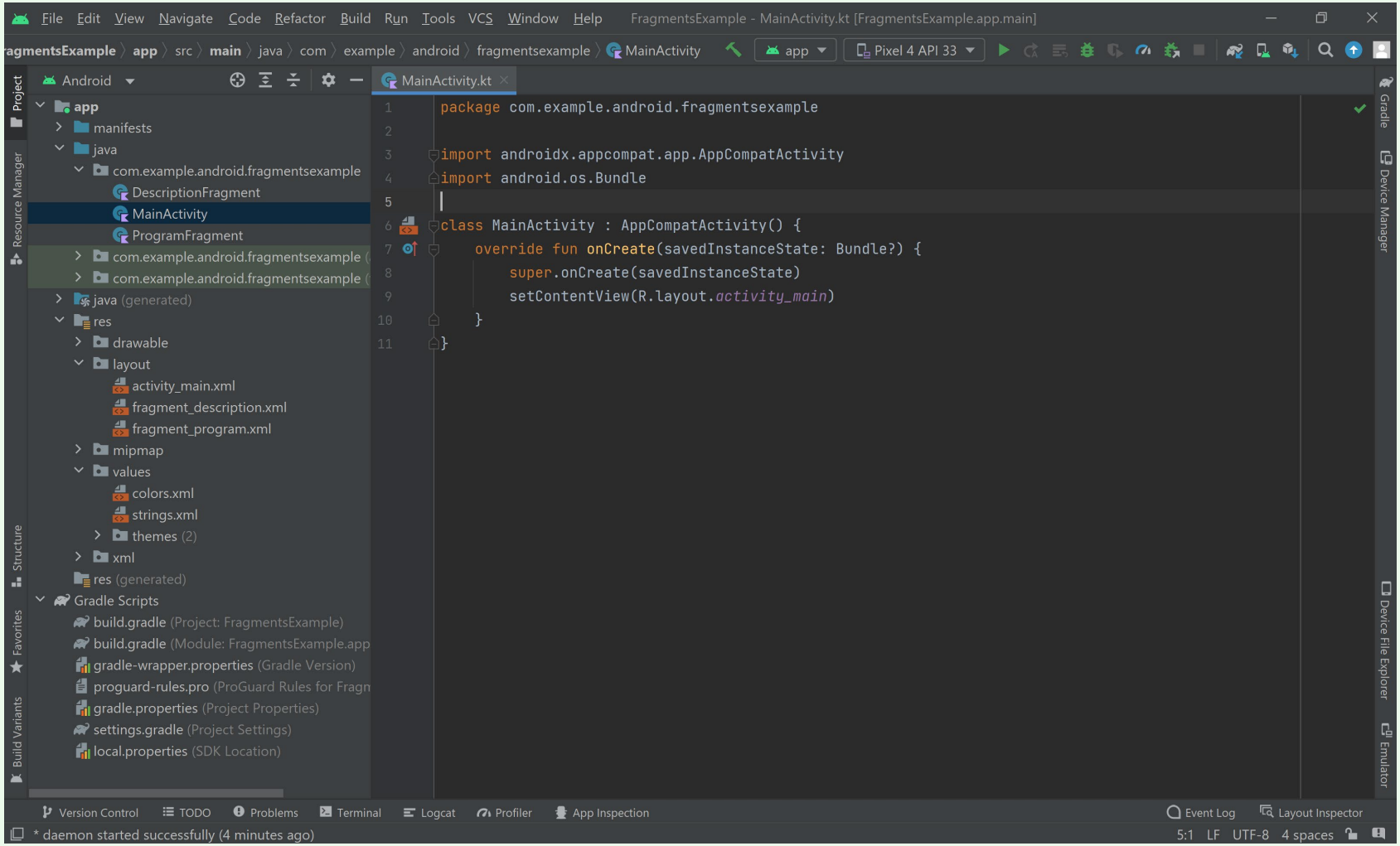


Adding Fragments to Activities

- The simplest way to add a Fragment to an Activity is by including it within the Activity's layout using the **FragmentContainerView** tag:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout  xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:weightSum="1.0">
    <androidx.fragment.app.FragmentContainerView
        android:id="@+id/fragment_container_view_program"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="0.5"
        android:name="com.example.android.fragmentsexample.ProgramFragment" />
    <androidx.fragment.app.FragmentContainerView
        android:id="@+id/fragment_container_view_description"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="0.5"
        android:name="com.example.android.fragmentsexample.DescriptionFragment" />
</LinearLayout >
```

FragmentsExample



FragmentsExample –fragment_program.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
   >

    <!-- TODO: Update blank fragment layout -->


    <RadioGroup xmlns:android="http://schemas.android.com/apk/res/android"
        android:id="@+id/radio_group"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical">
        <RadioButton android:id="@+id/radio_sety"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/sety"
            />
```

FragmentsExample –fragment_program.xml

```
<RadioButton android:id="@+id/radio_hit"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/hit“  />
    <RadioButton android:id="@+id/radio_gp"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/gp“  />
    <RadioButton android:id="@+id/radio_ai"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/ai“  />

    <RadioButton android:id="@+id/radio_mapd"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/mapd"
        />
</RadioGroup>
</LinearLayout>
```

FragmentsExample –fragment_description.xml

```
<?xml version="1.0" encoding="utf-8"?>
<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:background="#5ba4e5"
    android:orientation="horizontal"
    >
    <!-- TODO: Update blank fragment layout -->

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="40sp"
        android:textColor="#ffff00"
        android:text="Program Description"
        android:id="@+id/program_view"/>

</LinearLayout>
```

FragmentsExample – ProgramFragment class

```
package com.example.android.fragmentsexample

import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.*
//
class ProgramFragment : Fragment() {
    val onCheckedChangeListener =
        RadioGroup.OnCheckedChangeListener { radioGroup, checkedId ->
            when (checkedId) {
                R.id.radio_sety -> {Toast.makeText(this@ProgramFragment.requireActivity(),
                    "Software Engineering Technology", Toast.LENGTH_SHORT).show()}
                R.id.radio_setn -> {}
                R.id.radio_hit -> {}
                R.id.radio_gp -> {}
                R.id.radio_ai -> {}
                R.id.radio_mapd -> {}
                else -> {}
            }
        }
```

FragmentsExample – ProgramFragment class

```
override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        // Inflate the layout for this fragment
        val fragmentView = inflater.inflate(R.layout.fragment_program, container, false)

        val radioGroup = fragmentView.findViewById(R.id.radio_group) as RadioGroup
        radioGroup.setOnCheckedChangeListener(onCheckedChangeListener);
        //
        return fragmentView
    }
}
```

FragmentsExample – DescriptionFragment class

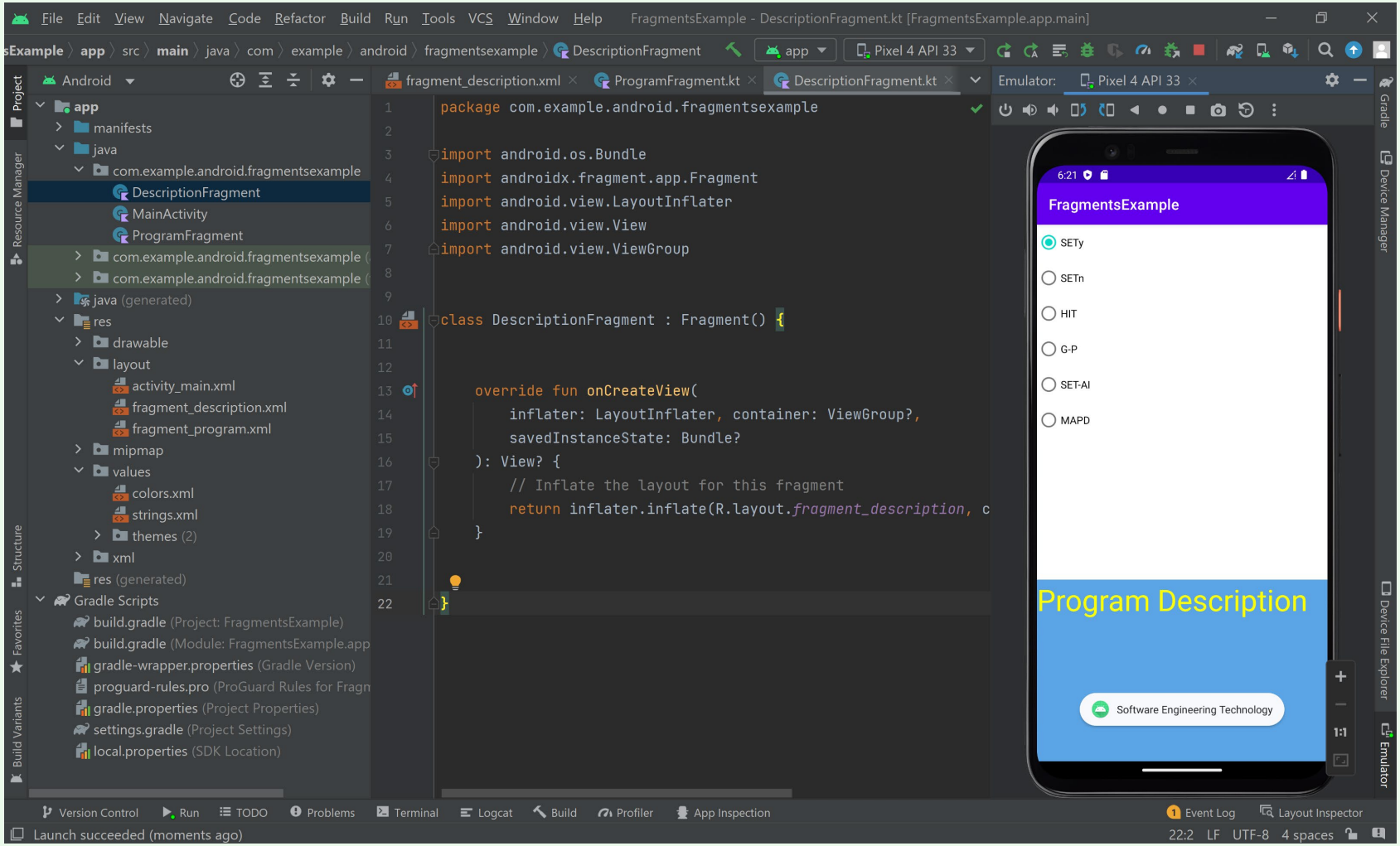
```
package com.example.android.fragmentsexample

import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup


class DescriptionFragment : Fragment() {

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.fragment_description, container, false)
    }
}
```

FragmentsExample



Using Fragment Transactions

- Fragment Transactions are used to **add**, **remove**, and **replace** Fragments within an Activity **at run time**.
  - A new Fragment Transaction is created using the beginTransaction method from the Fragment Manager.
  - Modify the layout using the add, remove, and replace methods.

```
val fragmentManager = ...
val fragmentTransaction = fragmentManager.beginTransaction()
val fragmentManager = ...
// The fragment-ktx module provides a commit block that automatically
// calls beginTransaction and commit for you.
fragmentManager.commit {
    // Add operations here
}
```

Using Fragment Transactions

- To find Fragments within your Activity, use the Fragment Manager's findFragmentById method:

```
val fragment: ExampleFragment =
supportFragmentManager.findFragmentById(R.id.frag
ment_container) as ExampleFragment
```

Communicating Between Fragments and Activities

- The Fragment library provides two options for communication:
  - a shared **ViewModel**
  - Fragment Result API.
- The recommended option depends on the use case.
  - To share persistent data with any custom APIs, you should use a **ViewModel**.
  - For a one-time result with data that can be placed in a Bundle, you should use the **Fragment Result API**.

References

- Textbook
- Reference book
- Android Documentation: https://developer.android.com/guide/components/activities.html
- https://developer.android.com/guide/components/fragments.html
- https://developer.android.com/codelabs/kotlin-android-training-create-and-add-fragment#1