MODUL 2
Building User Screen Flows

THEME DESCRIPTION

This module covers the Android activity lifecycle and explains how the Android system interacts with your app.

WEEKLY LEARNING OUTCOME (SUB-LEARNING OUTCOME)

Students will have learned how to build user journeys through different screens. You'll also be able to use activity tasks and launch modes, save and restore the state of your activity, use logs to report on your application, and share data between screens.

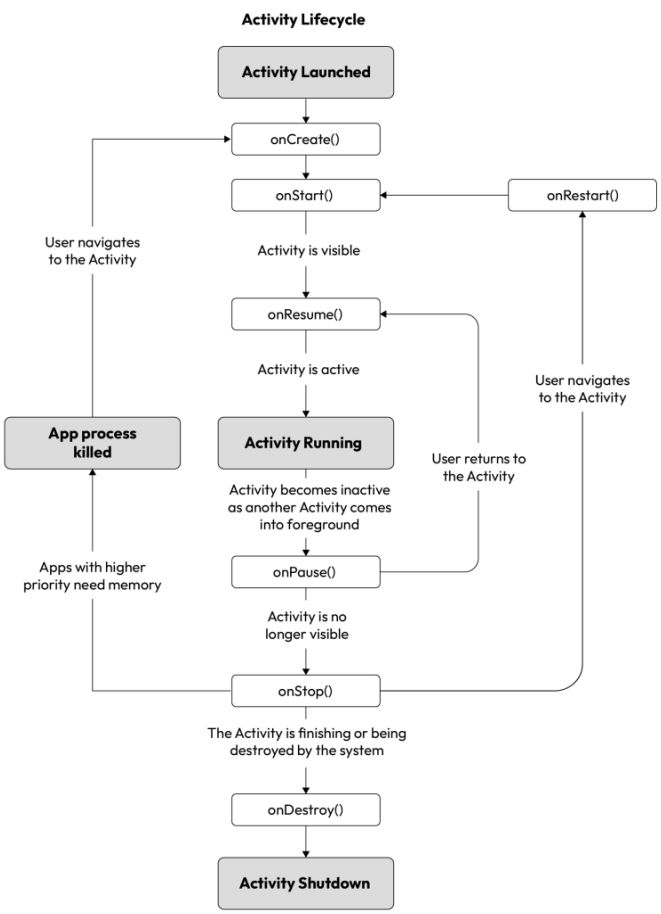
TOOLS/SOFTWARE USED

- Android Studio

CONCEPTS

Activity Lifecycle

Activity lifecycle is a **series of steps/processes** that correspond with your application being hidden, backgrounded, and then destroyed. For every one of these steps, there is a **callback** that your Activity can use to perform actions such as creating and changing the display and saving data when your app has been put into the background and then restoring that data after your app comes back into the foreground.



- **On Create**

```
override fun onCreate(savedInstanceState: Bundle?)
```

- **On Start**

```
override fun onStart()
```

- **On Restart**

```
override fun onRestart()
```

- **On Resume**

```
override fun onResume()
```

- **On Pause**

```
override fun onPause()
```

- **On Stop**

```
override fun onStop()
```

- **On Destroy**

```
override fun onDestroy()
```

Intent

An **intent** in Android is a communication mechanism between components. Within your own app, a lot of the time, you will want **another specific Activity** to start when some action happens in the current activity. There are 2 types of intent:

- **Explicit intent**
  Specifies which activity or component will start. Example: Other activity in the same app.
- **Implicit intent**
  Doesn't specify which component will start. Example: Camera.

Launch Modes

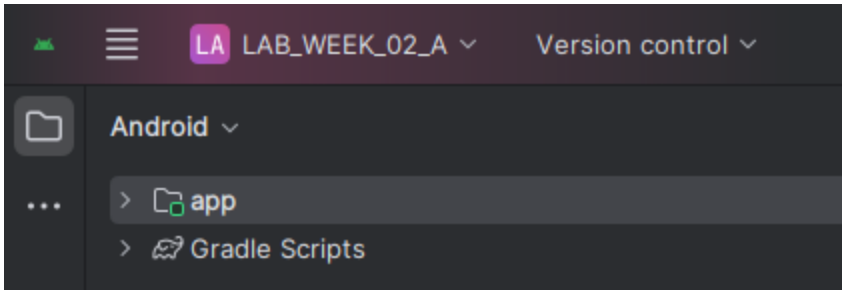
Launch mode controls the behavior of calling an activity into the screen. There are 5 modes currently available, but only 2 are often used:

- **Standard Mode**
  This is the **default** mode and doesn't need specifying in the Activity element of AndroidManifest.xml. When you open the app from the launcher with **standard mode**, it creates its **own Task**, and each Activity you create is added to a **back stack**, so when you open three Activities one after the other as part of your user's journey, pressing the back button three times will move the user back through the previous screens/Activities and then go back to the device's home screen, while still keeping the app open.
- **Single Top Mode**
  This is an **alternative** to the default behavior. If a **singleTop** Activity is the most recently added, when the same **singleTop** Activity is launched again, then instead of creating a **new Activity**, it uses the **same Activity** and runs the **onNewIntent** callback. In this callback, you receive an intent, and you can then process this intent as you have done previously in onCreate.

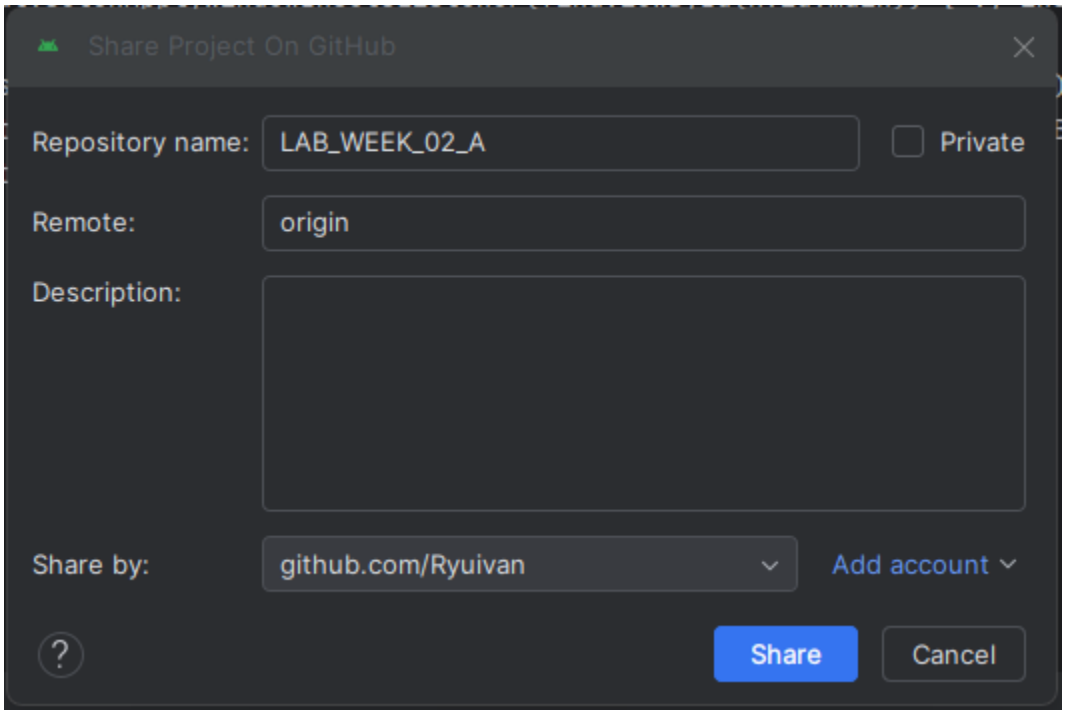
PRACTICAL STEPS

Part 1 - Activity & Life Cycle

1. Open Android Studio and click **New Project.**
2. Choose the **Empty Views Activity** to start with.
3. Name your project **"LAB_WEEK_02_A"**.
4. Set the minimum SDK to **"API 24: Android 7.0 (Nougat)"**.
5. Click **Finish**, and let your android application build itself.
6. Create GitHub repository by select **"Version Control > Share Project On > GitHub"** on the top left.



7. Don't forget to link your **GitHub** account with Android Studio.

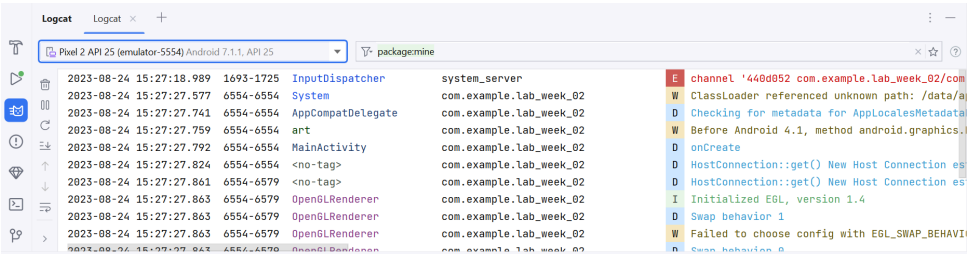


8. Select **Share** and type **"Initial commit"** as the first commit to your branch. Then select **Add**.
9. In this part, we will be focusing on how the **activity lifecycle** works in Android. Update your **MainActivity.kt** to the code below.

```
class MainActivity : AppCompatActivity() {
   companion object {
       private const val TAG = "MainActivity"
   }

   override fun onCreate(savedInstanceState: Bundle?) {
       super.onCreate(savedInstanceState)
       setContentView(R.layout.activity_main)
       Log.d(TAG, "onCreate")
```

```
    }
}
```

10. In order to know how they work, we need to know when and how they are called while our application is running. In this case, we use the help of **Log.d** which prints out debug logs into our **Logcat** window.
11. Run your application, and click on the **Logcat** button at the bottom left of your window.

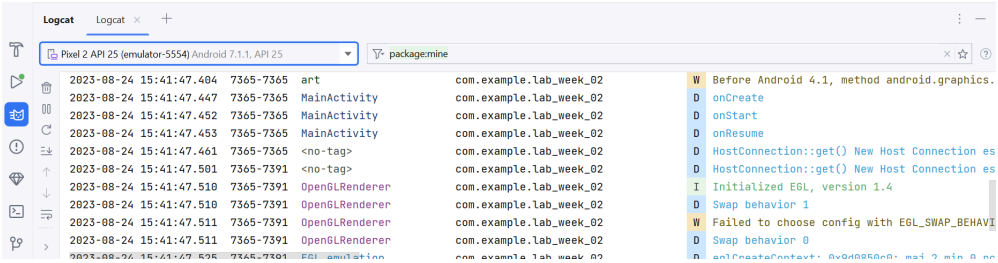


12. You can see our **Log.d** prints out **"onCreate"** into the **Logcat** window on the fifth line. This shows that our **onCreate** callback was successfully called.
13. Now try the above step for the other life cycles. Update your **MainActivity.kt** to the code below.

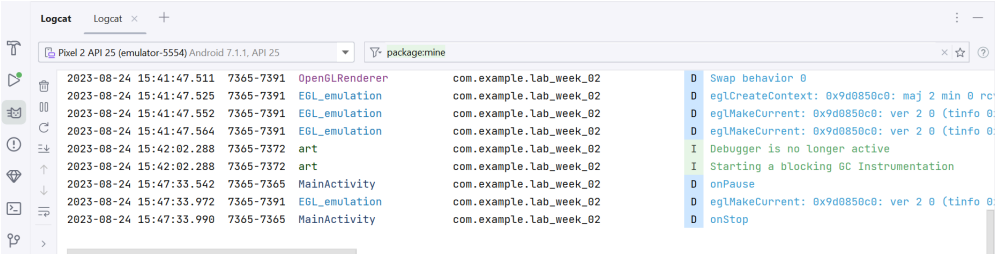
```
class MainActivity : AppCompatActivity() {
    companion object {
        private const val TAG = "MainActivity"
    }

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

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

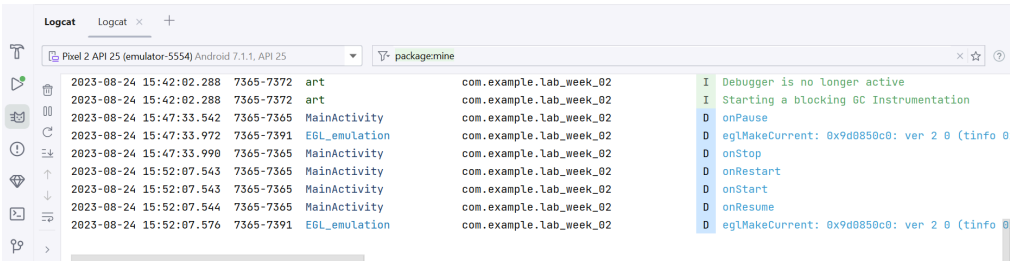
14. Now run your application again and your **Logcat** should print out **onCreate**, **onStart**, **onResume** in order.



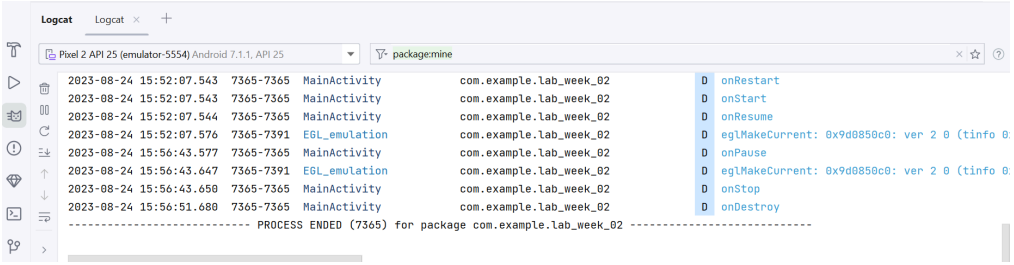
15. Now press the **Home** button on your device and your **Logcat** should print out **onPause**, **onStop** in order.



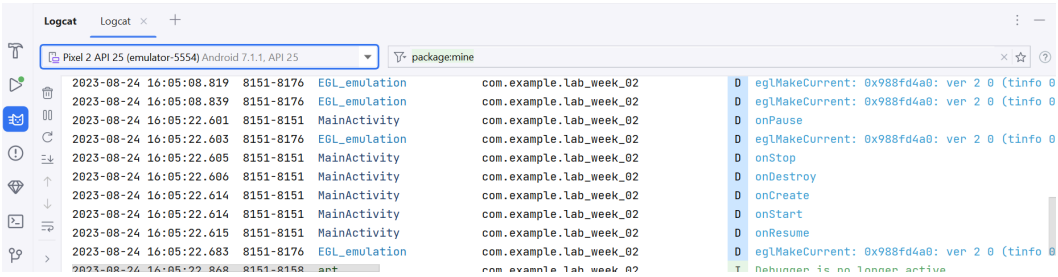
16. Now press the **Recent** square button on your device and bring the app back into the foreground. Your **Logcat** should print out **onRestart**, **onStart, onResume** in order.



17. Press the **Recent** square button again and swipe the app away to kill the activity. Your **Logcat** should print out **onPause**, **onStop**, **onDestroy** in order.



18. Lastly, you need to know that rotating the device also counts as recreating the activity. Try rotating your device and your **Logcat** should print out **onPause, onStop, onDestroy, onCreate, onStart, onResume** in order.



19. To prevent the activity recreation, update your **MainActivity** in your **AndroidManifest.xml** file to the code below.

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

20. Now try rotating your device again, and no lifecycle logs should appear in your **Logcat**.

COMMIT to **GITHUB** at this point. **Commit Message: "Commit No. 1 – Activity lifecycle"**.

Part 2 - Intent

1. Create a **New Project.**
2. Choose the **Empty Views Activity** to start with.
3. Name your project **"LAB_WEEK_02_B"**.
4. Set the minimum SDK to **"API 24: Android 7.0 (Nougat)"**.
5. Click **Finish**, and let your android application build itself.
6. Create new **GitHub** repository for this project and commit.
7. In this part, we will be focusing on how **intent** works in Android. We will be making a color code picker app. Update your **activity_main.xml** to the code below.

```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.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"
   style="@style/screen_margin">

   <LinearLayout
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       android:layout_marginTop="16dp"
       android:orientation="vertical"
       app:layout_constraintTop_toTopOf="parent">

       <TextView
           android:id="@+id/welcome_message"
           android:layout_width="match_parent"
           android:layout_height="match_parent"
```

```
            style="@style/text_display"
            android:text="@string/welcome_message" />
        <com.google.android.material.textfield.TextInputLayout
            android:id="@+id/color_code_input_wraper"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            style="@style/text_input"
            android:hint="@string/color_code_input_message">
            <com.google.android.material.textfield.TextInputEditText
                android:id="@+id/color_code_input_field"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:inputType="text"
                android:maxLength="6" />
        </com.google.android.material.textfield.TextInputLayout>
        <Button
            android:id="@+id/submit_button"
            android:layout_width="200dp"
            android:layout_height="wrap_content"
            style="@style/button"
            android:text="@string/submit_button"/>
    </LinearLayout>

</androidx.constraintlayout.widget.ConstraintLayout>
```

8. Now update your **strings.xml** to the code below.

```
<resources>
    <string name="app_name">LAB_WEEK_02</string>
    <string name="welcome_message">Welcome! Enter your preferred color
code!</string>
    <string name="color_code_input_message">Enter color code</string>
    <string name="color_code_input_empty">Color code field is empty!</string>
    <string name="color_code_input_wrong_length">Color code has to be 6
digits!</string>
    <string name="color_code_result_message">Color code #%s is applied!</string>
    <string name="color_code_input_invalid">Color code is invalid!</string>
    <string name="submit_button">SUBMIT</string>
</resources>
```

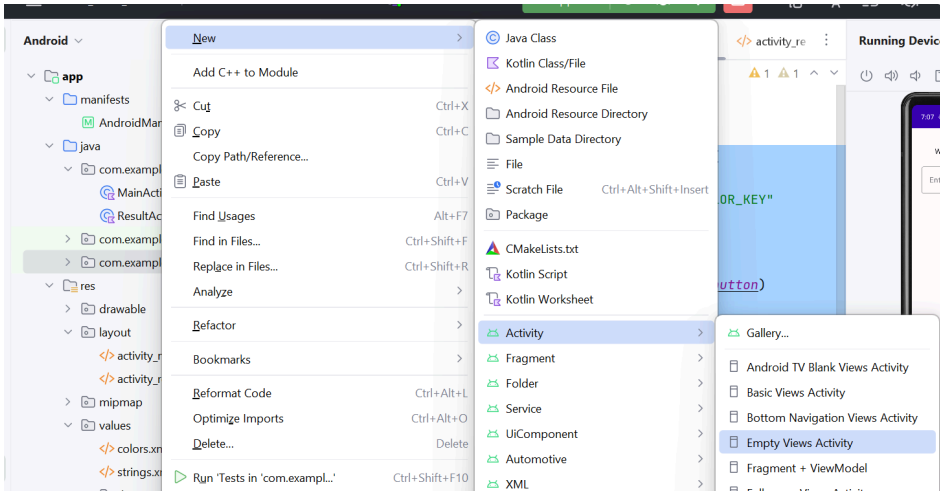
9. Next update your **themes.xml** to the code below.

```
<resources xmlns:tools="http://schemas.android.com/tools">
    <!-- Base application theme. -->
    <style name="Base.Theme.LAB_WEEK_02"
```

```
parent="Theme.Material3.DayNight.NoActionBar">
       ...
   </style>

   <style name="Theme.LAB_WEEK_02" parent="Base.Theme.LAB_WEEK_02" />

   <!-- Your custom styles -->
   <style name="text_input"
parent="Widget.MaterialComponents.TextInputLayout.OutlinedBox">
       <item name="android:layout_margin">8dp</item>
   </style>
   <style name="button">
       <item name="android:layout_margin">8dp</item>
       <item name="android:layout_gravity">center</item>
   </style>
   <style name="text_display">
       <item name="android:layout_margin">8dp</item>
       <item name="android:gravity">center</item>
       <item name="android:layout_height">40dp</item>
   </style>
   <style name="screen_margin">
       <item name="android:layout_margin">12dp</item>
   </style>
</resources>
```

10. Your main activity layout should now look like this.

11. Our main layout is done, now let's make another activity. **Right click** anywhere in the **files menu** and then choose **New > Activity > Empty Views Activity.** Name your new activity **Result Activity**.



12. Now edit the **activity_result.xml** file to the code below.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.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=".ResultActivity"
    android:id="@+id/background_screen">
    <TextView
        android:id="@+id/color_code_result_message"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintBottom_toBottomOf="parent"
        style="@style/text_display"
        android:textColor="@color/white"/>
</androidx.constraintlayout.widget.ConstraintLayout>
```

13. Your **activity_result.xml** should show a blank activity for now.
14. We've set up all of the activity layouts, now we can focus on the logic behind it. Let's start with the **MainActivity.kt** file. Update it to the code below.

```
class MainActivity : AppCompatActivity() {
    companion object {
        private const val COLOR_KEY = "COLOR_KEY"
    }

    private val submitButton: Button
        get() = findViewById(R.id.submit_button)

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        submitButton.setOnClickListener{
            val colorCode =
findViewById<TextInputEditText>(R.id.color_code_input_field).text.toString()

            if(colorCode.isNotEmpty()){
                if (colorCode.length < 6){
                    Toast
                        .makeText(this,
getString(R.string.color_code_input_wrong_length), Toast.LENGTH_LONG)
                        .show()
                }
                else{
                    val ResultIntent = Intent(this, ResultActivity::class.java)
                    ResultIntent.putExtra(COLOR_KEY, colorCode)
                    startActivity(ResultIntent)
```

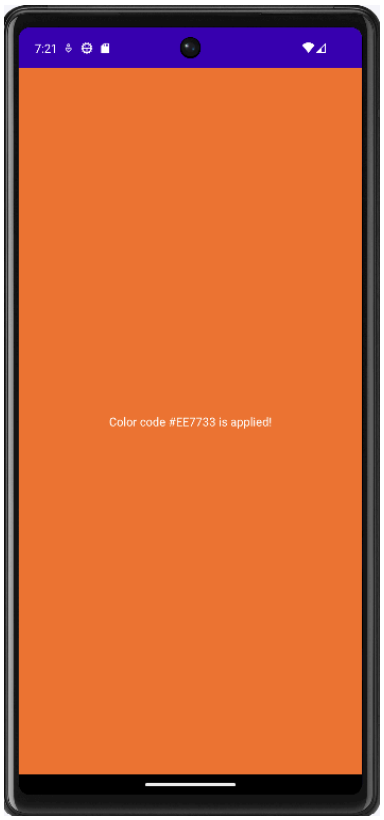
```
                }
            }
            else{
                Toast
                    .makeText(this, getString(R.string.color_code_input_empty),
Toast.LENGTH_LONG)
                    .show()
            }
        }
    }
}
```

15. To sum up, the above code accepts a color code input, which will be **passed** on to the **ResultActivity** through our **intent** (using extra). This color code will then be received and displayed as the background color of **ResultActivity**.
16. Next, let's set up our **ResultActivity.kt** as the receiver of the started intent. Update your **ResultActivity.kt** to the code below.

```
class ResultActivity : AppCompatActivity() {
    companion object {
        private const val COLOR_KEY = "COLOR_KEY"
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_result)

        if(intent != null){
            val colorCode = intent.getStringExtra(COLOR_KEY)

            val backgroundScreen =
findViewById<ConstraintLayout>(R.id.background_screen)
            backgroundScreen.setBackgroundColor(Color.parseColor("#$colorCode"))
            val resultMessage =
findViewById<TextView>(R.id.color_code_result_message)
            resultMessage.text = getString(R.string.color_code_result_message,
colorCode?.uppercase())
        }
    }
}
```

17. To sum up, the above code **receives** an intent with the **passed** on color code through **extra**. The color code will then be displayed as the background color of the activity.

18. Now try running your application, and insert a random color code. An intent should be started and you will be moved to another activity with the background color set to your chosen color.



19. Congratulations, you've successfully made a simple intent-based application. Unfortunately, there's 1 problem with the current version of the app. Any **invalid** color code combination will result in the app being **crashed**.

> **COMMIT** to **GITHUB** at this point. **Commit Message: "Commit No. 2 – Result activity [Intent]"**.

20. In order to fix this, we need to retrieve an **error message** back from **ResultActivity** to **MainActivity**. We can do this by using **startActivityResult** instead of **startActivity.**
21. In **MainActivity.kt**, add the code below above your **onCreate** method.

```
private val startForResult =
registerForActivityResult(ActivityResultContracts.StartActivityForResult()){
   activityResult ->
   val data = activityResult.data
   val error = data?.getBooleanExtra(ERROR_KEY, false)
   if(error == true){
```

```
        Toast
            .makeText(this, getString(R.string.color_code_input_invalid),
Toast.LENGTH_LONG)
            .show()
    }
}

override fun onCreate(savedInstanceState: Bundle?) {
        ...
}
```

22. The above code accepts an optional response from **ResultActivity** (in this case an error confirmation) and displays it as a **toast** message in the **MainActivity**.
23. Because we're using a new key for our **getBooleanExtra**, update your companion object to the code below.

```
companion object {
    private const val COLOR_KEY = "COLOR_KEY"
    private const val ERROR_KEY = "ERROR_KEY"
}
```

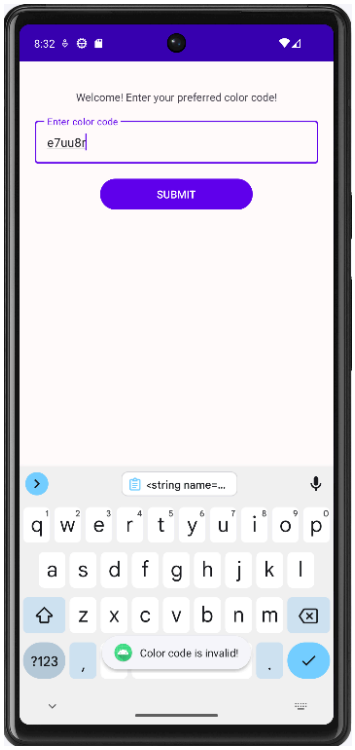
24. Also update your input validation to the code below.

```
if(colorCode.isNotEmpty()){
    if (colorCode.length < 6){
        Toast
            .makeText(this, getString(R.string.color_code_input_wrong_length),
Toast.LENGTH_LONG)
            .show()
    }
    else{
        val ResultIntent = Intent(this, ResultActivity::class.java)
        ResultIntent.putExtra(COLOR_KEY, colorCode)
        //startActivity(ResultIntent)
        startForResult.launch(ResultIntent)
    }
}
```

25. Now instead of using **startActivity()**, we use **startForResult.lauch()**. With this, a **response** can be passed back from **ResultActivity** back to **MainActivity**.
26. Next, let's update our **ResultActivity.kt** so it can pass back an error argument back to **MainActivity.kt**. Update your **ResultActivity.kt** to the code below.

```
class ResultActivity : AppCompatActivity() {
   companion object {
       private const val COLOR_KEY = "COLOR_KEY"
       private const val ERROR_KEY = "ERROR_KEY"
   }

   override fun onCreate(savedInstanceState: Bundle?) {
       super.onCreate(savedInstanceState)
       setContentView(R.layout.activity_result)

       if(intent != null){
           val colorCode = intent.getStringExtra(COLOR_KEY)

           val backgroundScreen =
findViewById<ConstraintLayout>(R.id.background_screen)
           try {
               backgroundScreen.setBackgroundColor(Color.parseColor("#$colorCode"))
           }
           catch (ex: IllegalArgumentException){
               Intent().let{
                   errorIntent ->
                   errorIntent.putExtra(ERROR_KEY, true)
                   setResult(Activity.RESULT_OK, errorIntent)
                   finish()
               }
           }
           val resultMessage =
findViewById<TextView>(R.id.color_code_result_message)
           resultMessage.text = getString(R.string.color_code_result_message,
colorCode?.uppercase())
       }
   }
}
```

27. Now our validation will send back an error response back to **MainActivity** if the color code is invalid.
28. Try running your application and input an invalid color code. An error message should be shown as a toast message in your **MainActivity**.



29. Congratulations, you've successfully made a fully functional intent-based application.

> **COMMIT** to **GITHUB** at this point. **Commit Message: "Commit No. 3 – Fix application crash when user input wrong color code [Intent]"**.

Part 3 - Launch Modes

1. Create a **New Project.**
2. Choose the **Empty Views Activity** to start with.
3. Name your project **"LAB_WEEK_02_C"**.
4. Set the minimum SDK to **"API 24: Android 7.0 (Nougat)"**.
5. Click **Finish**, and let your android application build itself.
6. Create new **GitHub** repository for this project and commit.
7. In this part, we will be focusing on how **launch mode** works in Android. Update your **activity_main.xml** to the code below.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.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">

    <include layout="@layout/activity_standard"
        android:id="@+id/layout_standard"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintBottom_toTopOf="@+id/layout_single_top"/>

    <include layout="@layout/activity_single_top"
        android:id="@+id/layout_single_top"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:layout_constraintTop_toBottomOf="@+id/layout_standard"
        app:layout_constraintBottom_toBottomOf="parent"/>

</androidx.constraintlayout.widget.ConstraintLayout>
```

8. Next, create 2 new activities called **StandardActivity** and **SingleTopActivity**.
9. After creating your activities, always remember to add it to the **AndroidManifest.xml** file.

```
<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>
<activity
    android:name=".StandardActivity"
    android:launchMode="standard">
</activity>
<activity
    android:name=".SingleTopActivity"
    android:launchMode="singleTop">
</activity>
```

10. In **activity_standard.xml**, update the file to the code below.

```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.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">

   <Button
       android:id="@+id/button_standard"
       android:layout_width="200dp"
       android:layout_height="wrap_content"
       android:text="@string/standard_button"
       android:padding="20dp"
       app:layout_constraintBottom_toBottomOf="parent"
       app:layout_constraintLeft_toLeftOf="parent"
       app:layout_constraintRight_toRightOf="parent"
       app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

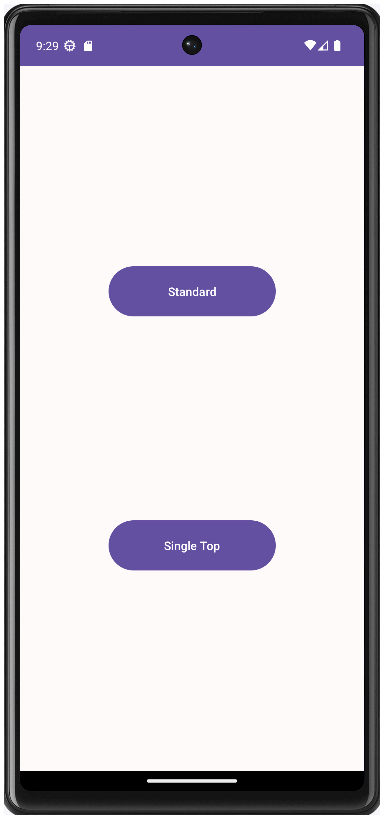
11. In **activity_single_top.xml**, update the file to the code below.

```xml
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.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">

   <Button
       android:id="@+id/button_single_top"
       android:layout_width="200dp"
       android:layout_height="wrap_content"
       android:text="@string/single_top_button"
       android:padding="20dp"
       app:layout_constraintBottom_toBottomOf="parent"
       app:layout_constraintLeft_toLeftOf="parent"
       app:layout_constraintRight_toRightOf="parent"
       app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

12. Don’t forget to also update your **strings.xml** to the code below.

```
<resources>
   <string name="app_name">LAB_WEEK_02_C</string>
   <string name="standard_button">Standard</string>
   <string name="single_top_button">Single Top</string>
</resources>
```

13. Your layout should now look like this.



14. Our layouts are done, now update your **StandardActivity.kt** to the code below.

```
class StandardActivity : AppCompatActivity() {
   companion object{
       private const val DEBUG = "DEBUG"
   }

   override fun onCreate(savedInstanceState: Bundle?) {
       super.onCreate(savedInstanceState)
       setContentView(R.layout.activity_standard)
       Log.d(DEBUG, "onCreate")
```

```
        findViewById<Button>(R.id.button_standard).setOnClickListener{
            startActivity(
                Intent(this,
                    StandardActivity::class.java)
            )
        }
    }

    override fun onNewIntent(intent: Intent?) {
        super.onNewIntent(intent)
        Log.d(DEBUG, "onNewIntent")
    }
}
```

15. Next, update your **SingleTopActivity.kt** to the code below.

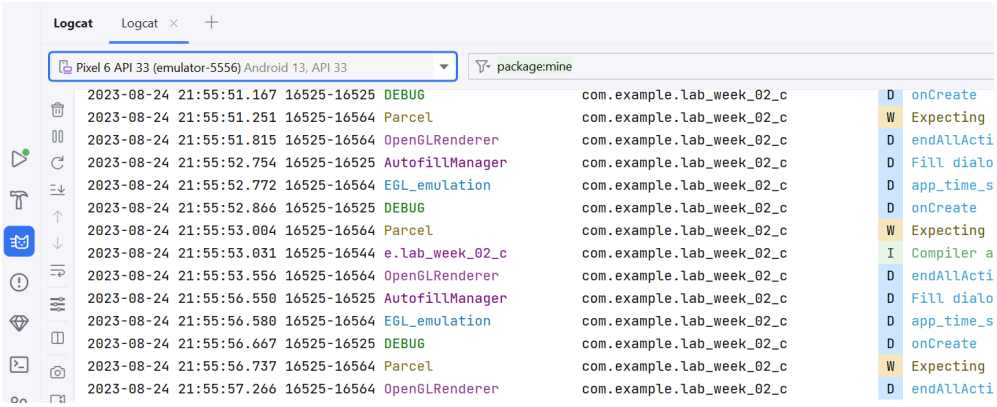
```
class SingleTopActivity : AppCompatActivity() {
    companion object{
        private const val DEBUG = "DEBUG"
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_single_top)
        Log.d(DEBUG, "onCreate")

        findViewById<Button>(R.id.button_single_top).setOnClickListener{
            startActivity(
                Intent(this,
                    SingleTopActivity::class.java)
            )
        }
    }

    override fun onNewIntent(intent: Intent?) {
        super.onNewIntent(intent)
        Log.d(DEBUG, "onNewIntent")
    }
}
```

16. Lastly, update your **MainActivity.kt** to the code below.

```
class MainActivity : AppCompatActivity() {
    companion object{
        private const val DEBUG = "DEBUG"
```

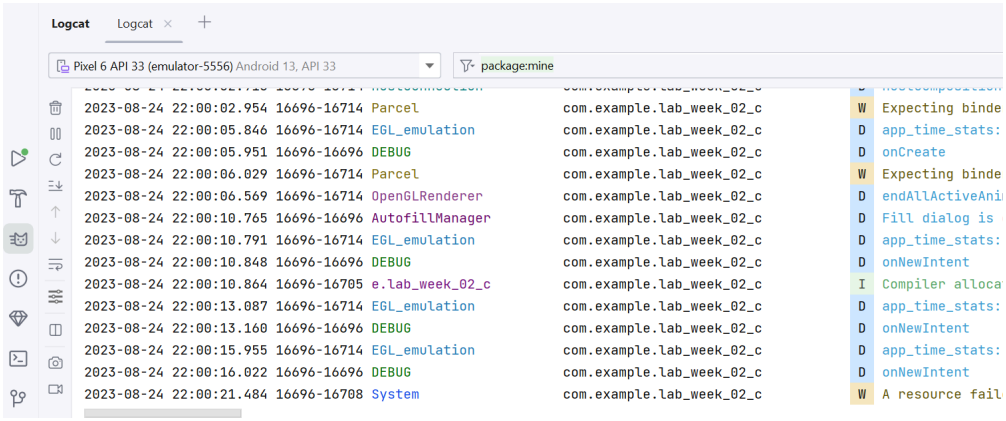
```
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        Log.d(DEBUG, "onCreate")

        val buttonClickListener = View.OnClickListener { view ->
            when (view.id) {
                R.id.button_standard -> startActivity(
                    Intent(this,
                        StandardActivity::class.java)
                )
                R.id.button_single_top -> startActivity(
                    Intent(this,
                        SingleTopActivity::class.java)
                )
            }
        }


findViewById<Button>(R.id.button_standard).setOnClickListener(buttonClickListener)

findViewById<Button>(R.id.button_single_top).setOnClickListener(buttonClickListener
)
    }
}
```

17. Now try running your application and click the **standard button** repetitively.
18. Open your **Logcat** and the **onCreate** log should be shown as many times as you press the button.



19. Now, try clicking the **Single Top Button** repetitively.

20. Open your **Logcat** and the **onCreate** log should only be shown once in the beginning. After that, the **onNewIntent** log will be shown every time the button is pressed again. This shows with **single top mode**, activity is not being recreated every time the same one is called.

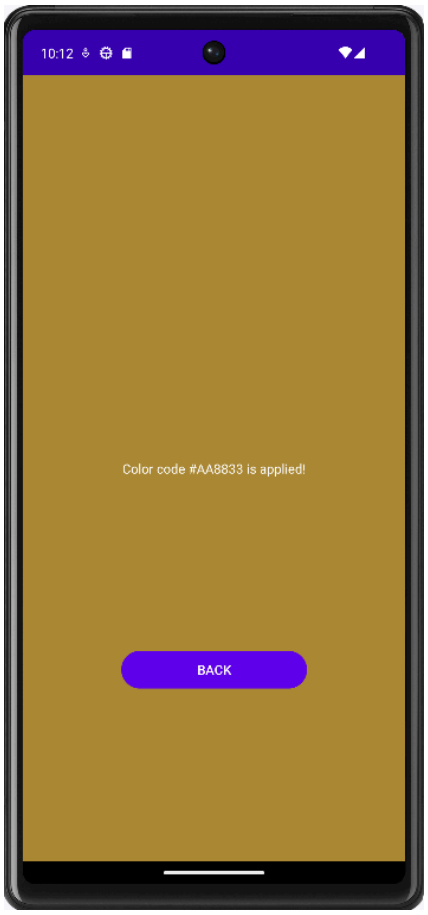


COMMIT to **GITHUB** at this point. **Commit Message: "Commit No. 4 – Standard activity and single top activity"**.

ASSIGNMENT

Continue your **LAB_WEEK_02_B** project, and add a **back button** in **ResultActivity** so the user can submit another color code without rerunning the application.

COMMIT to **GITHUB** at this point. **Commit Message: "Commit No. 5 – Fix ResultActivity to allow resubmitting color code without app restart"**.


10:12
Color code #AA8833 is applied!
BACK