Task ##P/C – Spike: Renting app

Goals:

This goal of the assignment is to create a simple renting app. This app need to include the implement of opening activity 2 and passing back the data from the activity 2 using intent and parcelable object. Also, this app is a great app to write a simple ui and unit test. Moreover, the app UI also being develop using android widget and validate the user input from the second activity
List if knowledge gap:

- Pass back data
- Validation user input
- Toast
- Unit and UI test
- UI element
- Style in app
- Sketch
- Intent and Parcelable object
- Command of IDE
  This is the link to my github classroom: https://github.com/SoftDevMobDev-2023-Classrooms/core2-bbi3mn4u69/tree/main

Tools and Resources Used

This section lists related software, tools, libraries, API's, and other resources used for this knowledge gap.

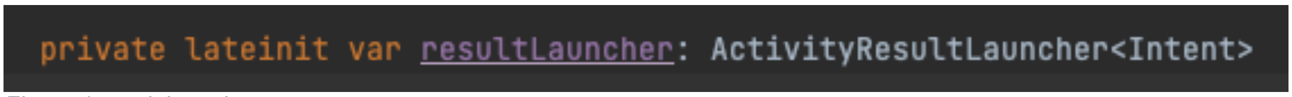
- Android studio
- Thakkar, R 2021, *UI Testing with Espresso in Android*, Mindful Engineering.
- *RatingBar* n.d., Android Developers, viewed 30 September 2023, <https://developer.android.com/reference/android/widget/RatingBar>.
- SeekBar n.d., Android Developers, viewed 30 September 2023, <https://developer.android.com/reference/android/widget/SeekBar?hl=en>.

Knowledge Gaps and Solutions

This section presents the listed knowledge gaps and their solutions with supporting images, screenshots and captions where appropriate/required.

Gap 1: Pass Back Data

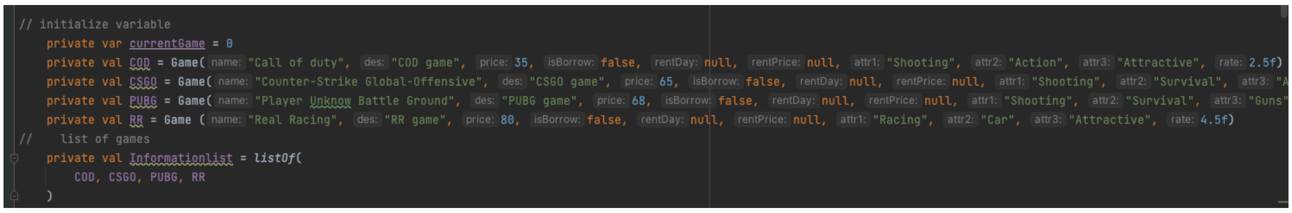
Since we are able to create a second activity, pussing data in and passing data back is important. In order to get the data back, I m using ActivityResultContract first we need to define a late initialization variable “result laucher”. This variable will hold an object of the ActivityResultLauncher type, which is a part of Android's Activity Result API. The <Intent> part indicates that this ActivityResultLauncher will be used for launching intents



```kotlin
private lateinit var resultLauncher: ActivityResultLauncher<Intent>
```

Figure 1 result launcher

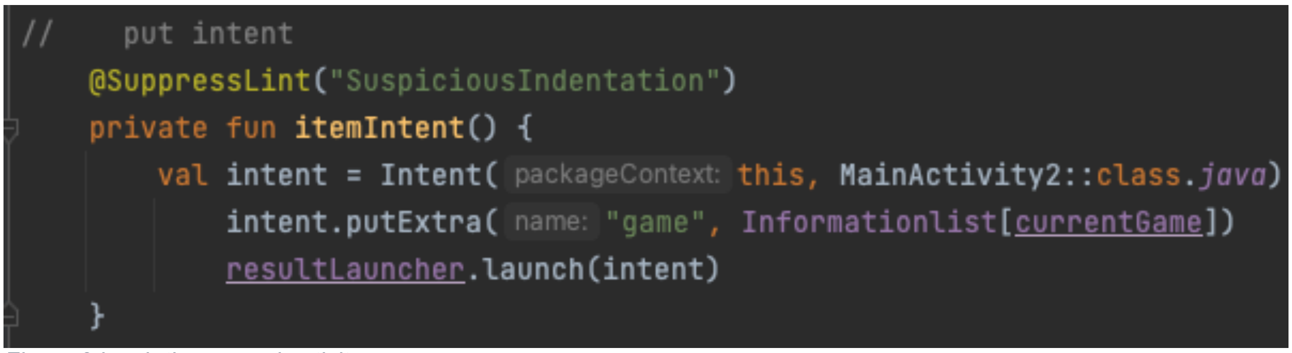
The result laucher will first used to lauch the second activity with passing the parcelable object (game) as an intent. The object here will be initialize at the beginning and being pushed in a list of game object (Information list).



```kotlin
// initialize variable
    private var currentGame = 0
    private val COD = Game( name: "Call of duty",  des: "COD game",  price: 35,  isBorrow: false,  rentDay: null,  rentPrice: null,  attr1: "Shooting",  attr2: "Action",  attr3: "Attractive",  rate: 2.5f)
    private val CSGO = Game( name: "Counter-Strike Global-Offensive",  des: "CSGO game",  price: 65,  isBorrow: false,  rentDay: null,  rentPrice: null,  attr1: "Shooting",  attr2: "Survival",  attr3: "A
    private val PUBG = Game( name: "Player Unknow Battle Ground",  des: "PUBG game",  price: 68,  isBorrow: false,  rentDay: null,  rentPrice: null,  attr1: "Shooting",  attr2: "Survival",  attr3: "Guns"
    private val RR = Game ( name: "Real Racing",  des: "RR game",  price: 80,  isBorrow: false,  rentDay: null,  rentPrice: null,  attr1: "Racing",  attr2: "Car",  attr3: "Attractive",  rate: 4.5f)
//   list of games
    private val Informationlist = listOf(
        COD, CSGO, PUBG, RR
    )
```

Figure 2 list of object

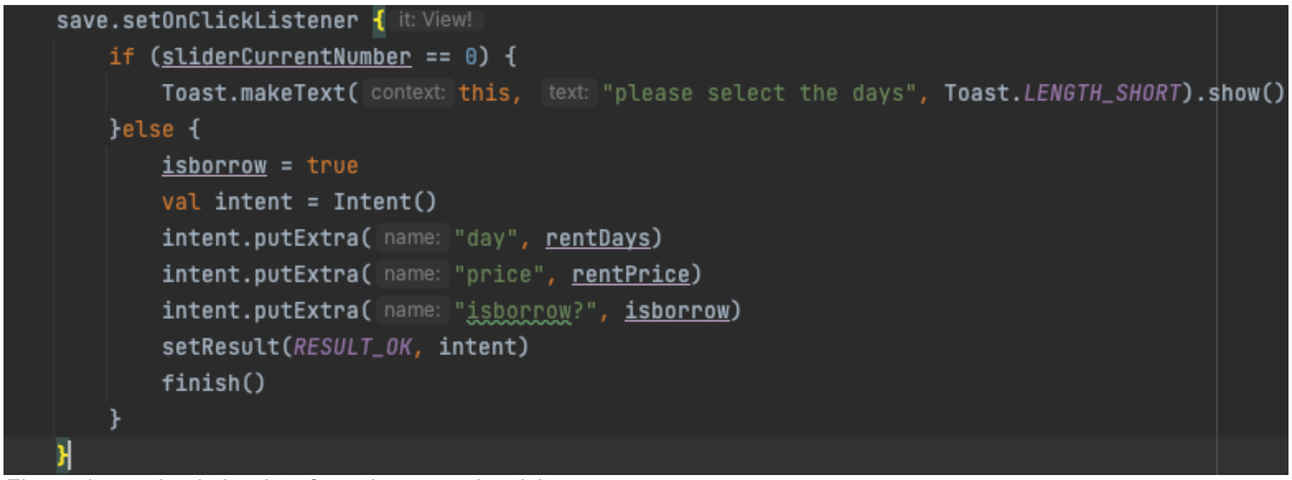
Here we are pushing the current game of that list aka the object at index number of that list of game. Pushing the current game (object) that the screen is showing and launching the second activity :



```kotlin
//    put intent
    @SuppressLint("SuspiciousIndentation")
    private fun itemIntent() {
        val intent = Intent( packageContext: this, MainActivity2::class.java)
            intent.putExtra( name: "game", Informationlist[currentGame])
            resultLauncher.launch(intent)
    }
```

Figure 3 lauch the second activity

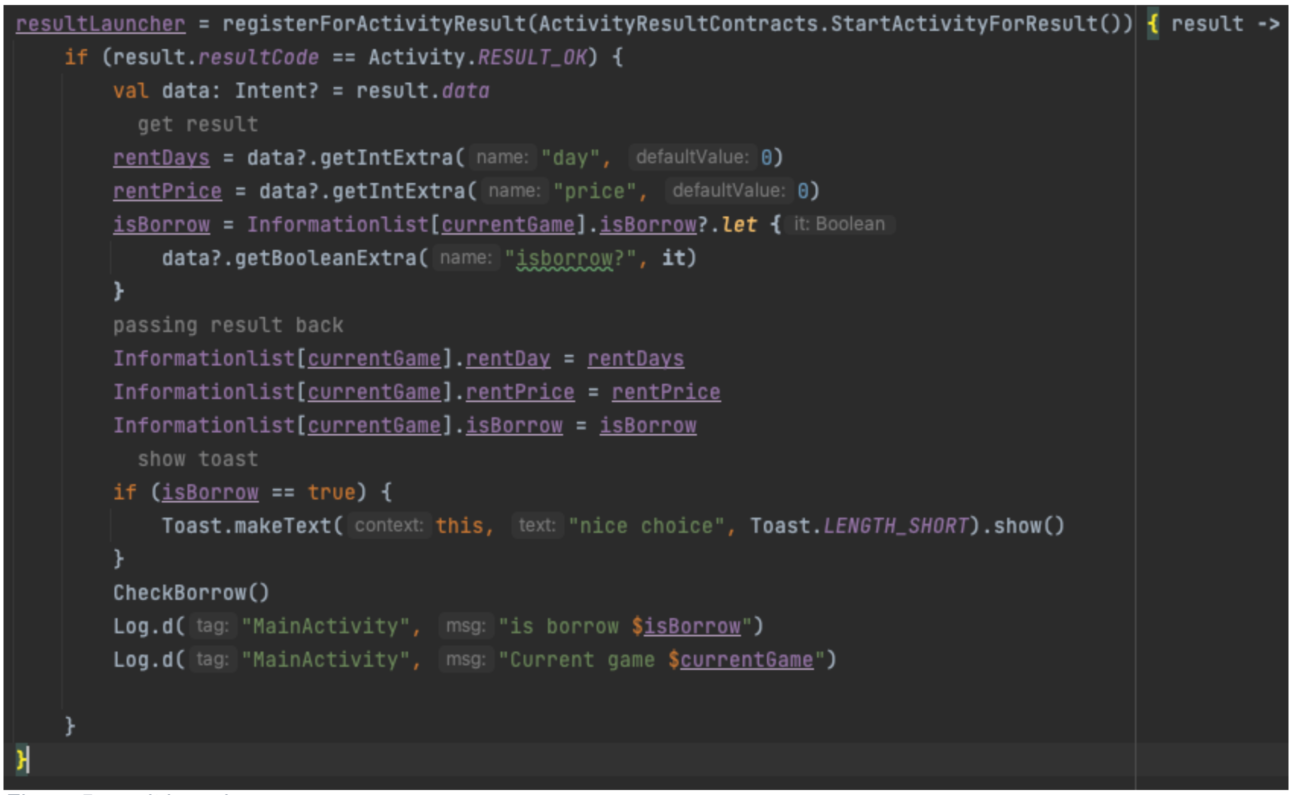
Second activity pass back the data:



```
save.setOnClickListener { it: View!
    if (sliderCurrentNumber == 0) {
        Toast.makeText( context: this,  text: "please select the days", Toast.LENGTH_SHORT).show()
    }else {
        isborrow = true
        val intent = Intent()
        intent.putExtra( name: "day", rentDays)
        intent.putExtra( name: "price", rentPrice)
        intent.putExtra( name: "isborrow?", isborrow)
        setResult(RESULT_OK, intent)
        finish()
    }
}
```

Figure 4 pass back the data from the second activity

Initialize the result laucher for result from the second activity
After the make sure the result ok, the second activity will pass the days, price and isborrow back to the main activity. These variable now will replace the object rentday, price and isborrow for further display to the screen. A toast will show if the user confirm that they want to borrow the game later on.

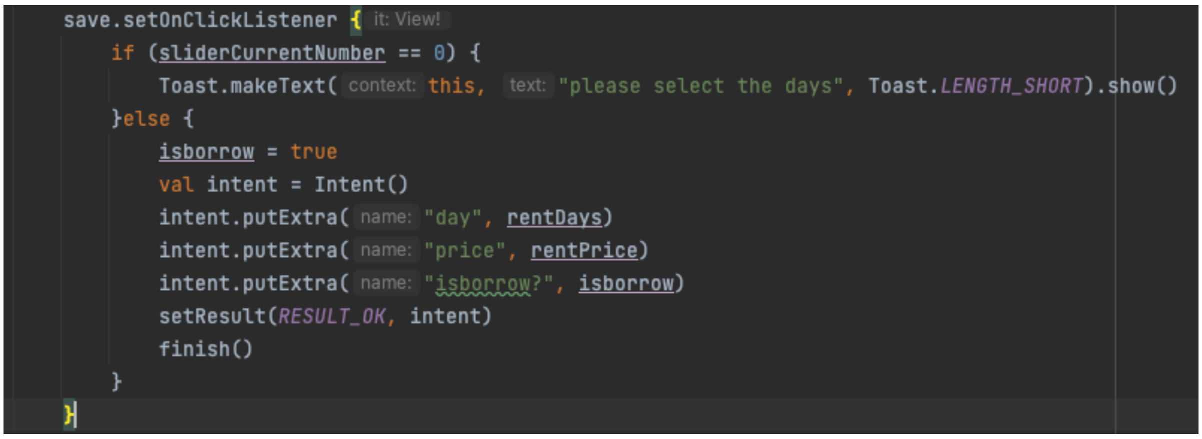


```
resultLauncher = registerForActivityResult(ActivityResultContracts.StartActivityForResult()) { result ->
    if (result.resultCode == Activity.RESULT_OK) {
        val data: Intent? = result.data
          get result
        rentDays = data?.getIntExtra( name: "day",  defaultValue: 0)
        rentPrice = data?.getIntExtra( name: "price",  defaultValue: 0)
        isBorrow = Informationlist[currentGame].isBorrow?.let { it: Boolean
            data?.getBooleanExtra( name: "isborrow?", it)
        }
        passing result back
        Informationlist[currentGame].rentDay = rentDays
        Informationlist[currentGame].rentPrice = rentPrice
        Informationlist[currentGame].isBorrow = isBorrow
          show toast
        if (isBorrow == true) {
            Toast.makeText( context: this,  text: "nice choice", Toast.LENGTH_SHORT).show()
        }
        CheckBorrow()
        Log.d( tag: "MainActivity",  msg: "is borrow $isBorrow")
        Log.d( tag: "MainActivity",  msg: "Current game $currentGame")

    }
}
```

Figure 5 result launcher

Gap 2: Validation of User input

For validation the user input, I have set up the code: the save button can only pressed if the user make change on the seek bar, if the back button is pressed, that consider to be the user cancel it:

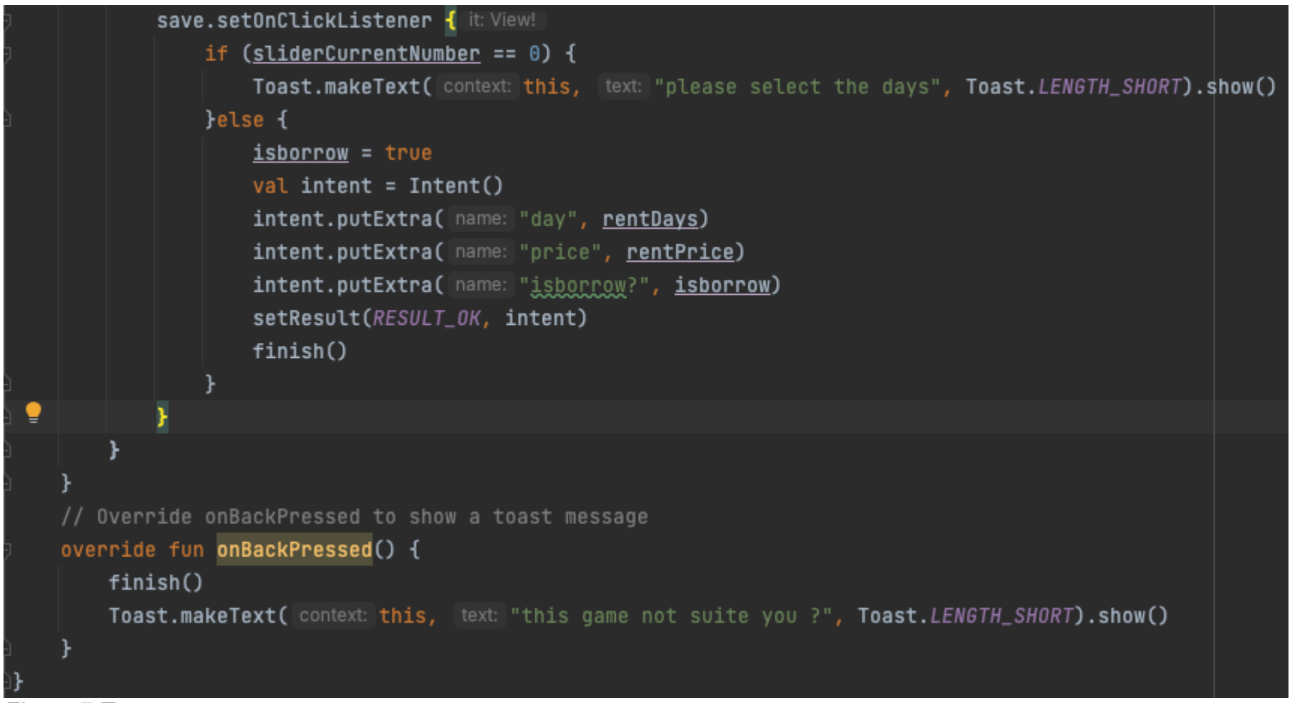


```
save.setOnClickListener { it: View!
    if (sliderCurrentNumber == 0) {
        Toast.makeText( context: this,  text: "please select the days", Toast.LENGTH_SHORT).show()
    }else {
        isborrow = true
        val intent = Intent()
        intent.putExtra( name: "day", rentDays)
        intent.putExtra( name: "price", rentPrice)
        intent.putExtra( name: "isborrow?", isborrow)
        setResult(RESULT_OK, intent)
        finish()
    }
}
```

Figure 6 validation of user input

Gap 3: Toast

I have implement the toast in this code, the toast will show up when the user press the save button without making any change to the seek bar. Also, when the user is make change to the seek bar to set the days they borrow the game and press the save button to go back, another toast will appear saying "good choice". If they press the back button, the program will not save the borrow information and display a toast.



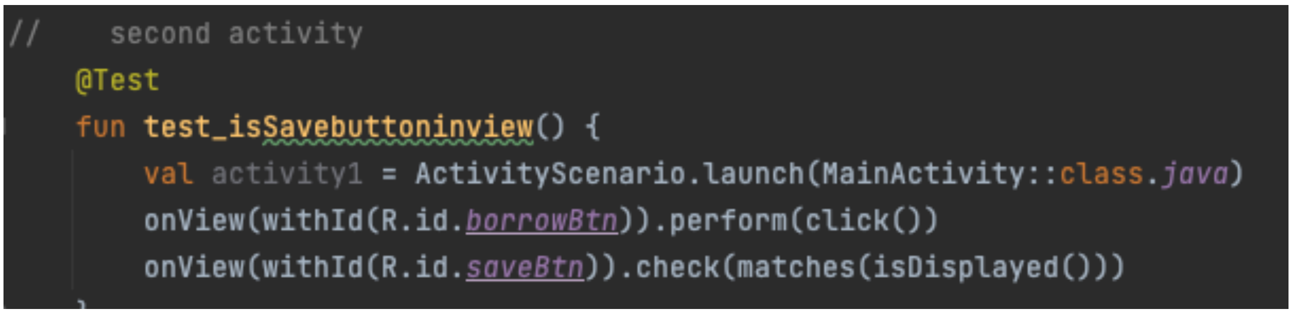
```
            save.setOnClickListener { it: View!
                if (sliderCurrentNumber == 0) {
                    Toast.makeText( context: this,  text: "please select the days", Toast.LENGTH_SHORT).show()
                }else {
                    isborrow = true
                    val intent = Intent()
                    intent.putExtra( name: "day", rentDays)
                    intent.putExtra( name: "price", rentPrice)
                    intent.putExtra( name: "isborrow?", isborrow)
                    setResult(RESULT_OK, intent)
                    finish()
                }
            }
        }
    }
    // Override onBackPressed to show a toast message
    override fun onBackPressed() {
        finish()
        Toast.makeText( context: this,  text: "this game not suite you ?", Toast.LENGTH_SHORT).show()
    }
}
```

Figure 7 Toast

In this app, I opted for using toast messages instead of snackbars because the messages displayed here don't affect or require any action from the user in response. Additionally, snackbars persist until the user takes an action or a timeout is set programmatically, which can result in an inconvenient user experience and unnecessary, redundant code. In contrast, toast messages simply appear for a brief 2-3 seconds, making them a more efficient choice.

Gap 4: Unit and UI test

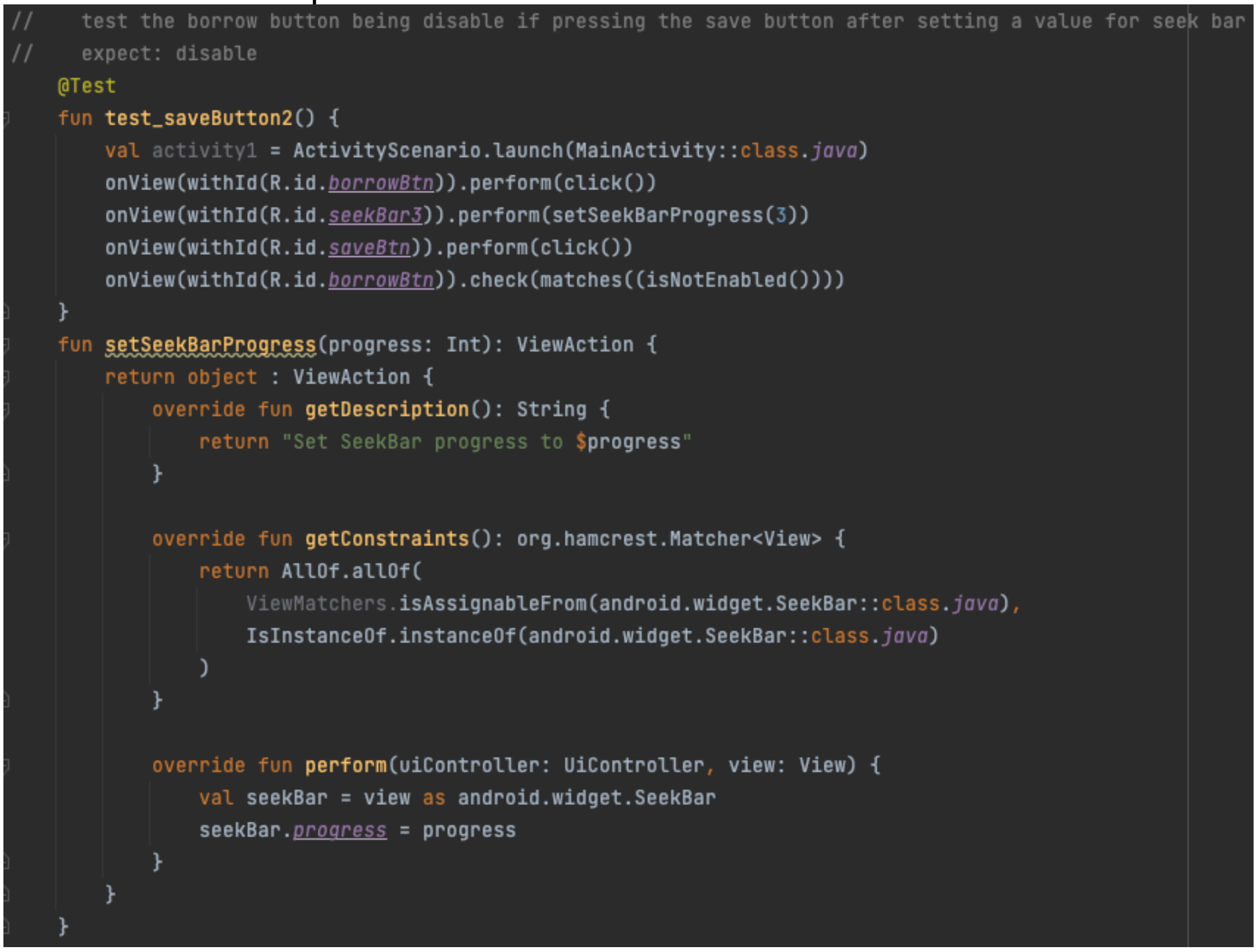
I have write a UI test for the program, tested if the text and the button display on the activity correctly: for example the save button will be appear if we press the borrow button and go to the second activity, etc



```
//    second activity
    @Test
    fun test_isSavebuttoninview() {
        val activity1 = ActivityScenario.launch(MainActivity::class.java)
        onView(withId(R.id.borrowBtn)).perform(click())
        onView(withId(R.id.saveBtn)).check(matches(isDisplayed()))
```

Figure 8 UI Test

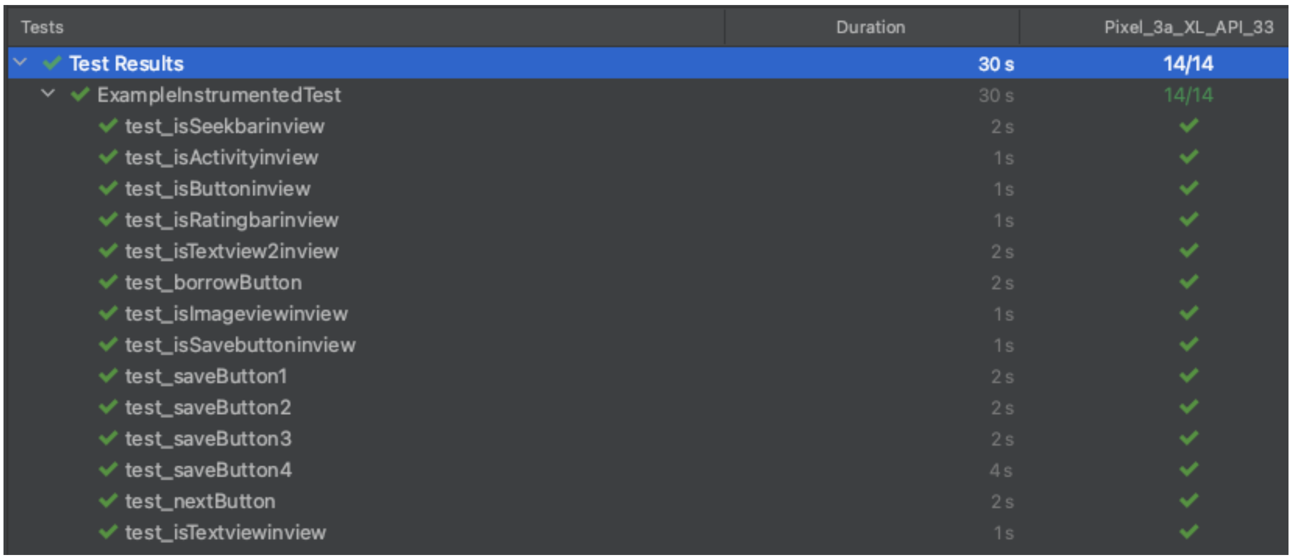
Also, a unit test is being implement. This unit test is make sure that the individual component behaviour in the way that we expected
For example: this test is when press the borrow button and set the values for the seek bar, then press save, the app return to the main activity and the borrow button is expected to be disable



```
//    test the borrow button being disable if pressing the save button after setting a value for seek bar
//    expect: disable
    @Test
    fun test_saveButton2() {
        val activity1 = ActivityScenario.launch(MainActivity::class.java)
        onView(withId(R.id.borrowBtn)).perform(click())
        onView(withId(R.id.seekBar3)).perform(setSeekBarProgress(3))
        onView(withId(R.id.saveBtn)).perform(click())
        onView(withId(R.id.borrowBtn)).check(matches((isNotEnabled())))
    }
    fun setSeekBarProgress(progress: Int): ViewAction {
        return object : ViewAction {
            override fun getDescription(): String {
                return "Set SeekBar progress to $progress"
            }

            override fun getConstraints(): org.hamcrest.Matcher<View> {
                return AllOf.allOf(
                    ViewMatchers.isAssignableFrom(android.widget.SeekBar::class.java),
                    IsInstanceOf.instanceOf(android.widget.SeekBar::class.java)
                )
            }

            override fun perform(uiController: UiController, view: View) {
                val seekBar = view as android.widget.SeekBar
                seekBar.progress = progress
            }
        }
    }
```

Figure 9 Unit Test

All the test is passed:

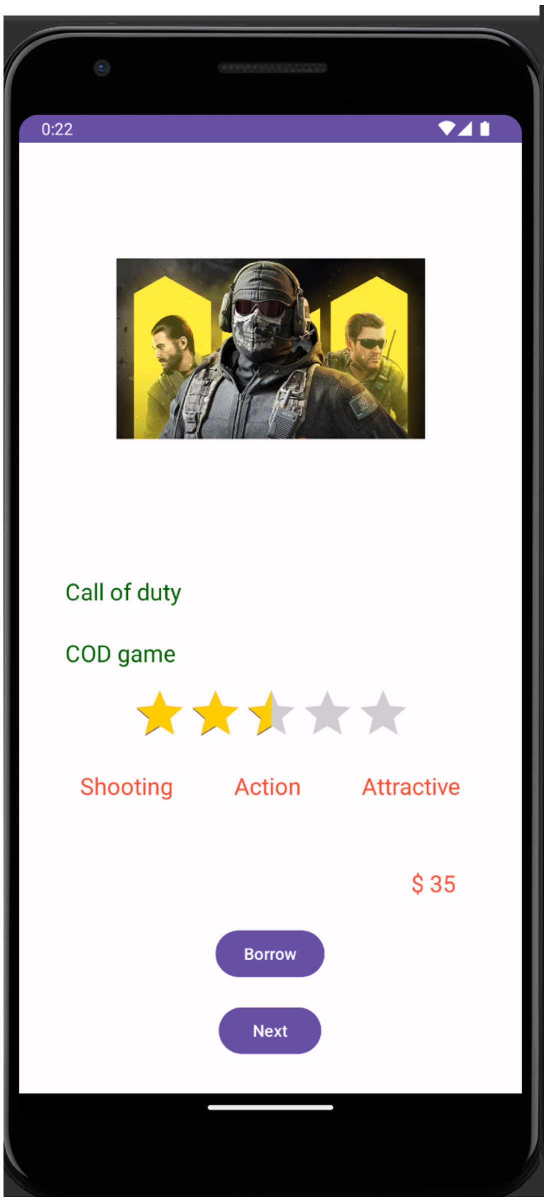


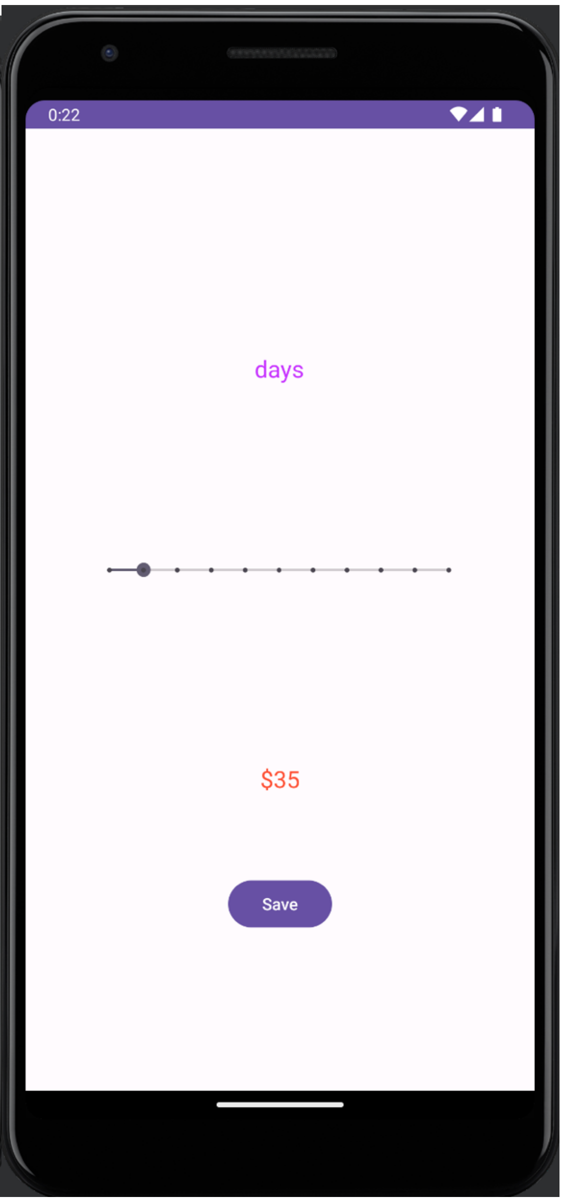
Tests	Duration	Pixel_3a_XL_API_33
✓ Test Results	30 s	14/14
✓ ExampleInstrumentedTest	30 s	14/14
✓ test_isSeekbarinview	2 s	✓
✓ test_isActivityinview	1 s	✓
✓ test_isButtoninview	1 s	✓
✓ test_isRatingbarinview	1 s	✓
✓ test_isTextview2inview	2 s	✓
✓ test_borrowButton	2 s	✓
✓ test_isImageviewinview	1 s	✓
✓ test_isSavebuttoninview	1 s	✓
✓ test_saveButton1	2 s	✓
✓ test_saveButton2	2 s	✓
✓ test_saveButton3	2 s	✓
✓ test_saveButton4	4 s	✓
✓ test_nextButton	2 s	✓
✓ test_isTextviewinview	1 s	✓

Figure 10 Test passed

Gap 5: UI element

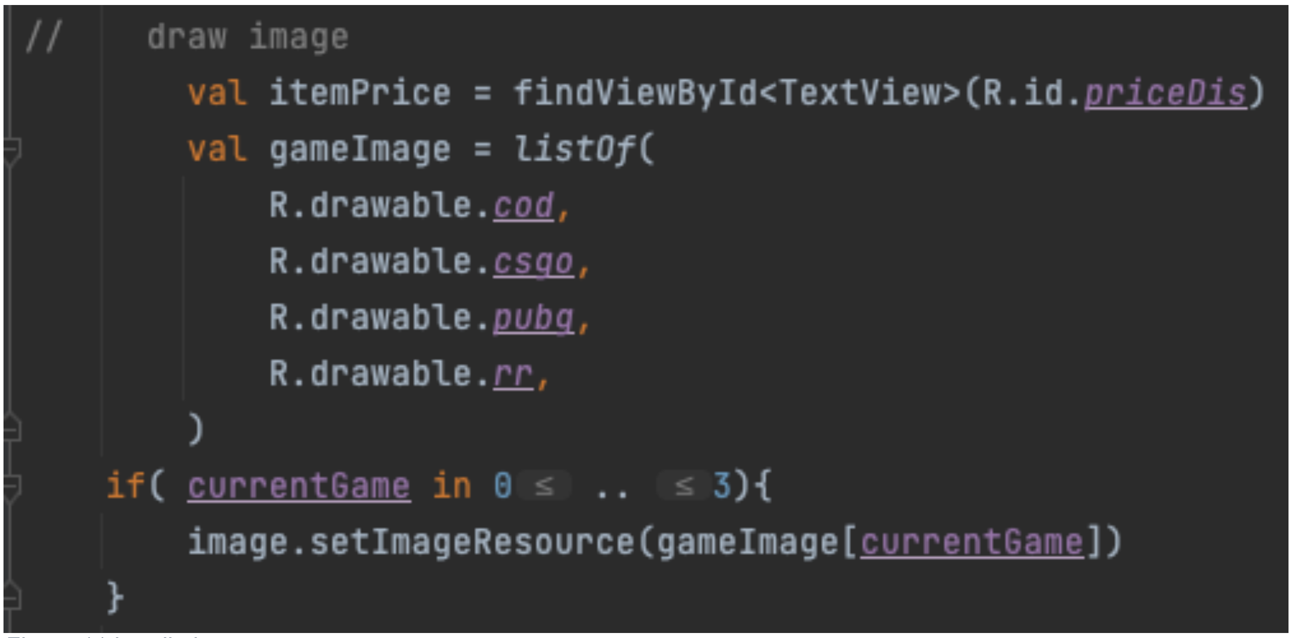
In this app, I have use a wide range of UI element, most typical is the seek bar and the rating bar the rating bar is used for the user to see the rating of the games that they will choose to borrow. Meanwhile the seek bar is used to set the days that the user borrow the game.





Gap 6: Work with resource

The image resource is included in a list of drawable resources. The "current game" variable corresponds to the index number of a list of games. The resource is accessed and displayed based on the value of the "current game" variable. For instance, if the current game is set to 0, the first resource (representing "COD")
will be drawn and displayed.



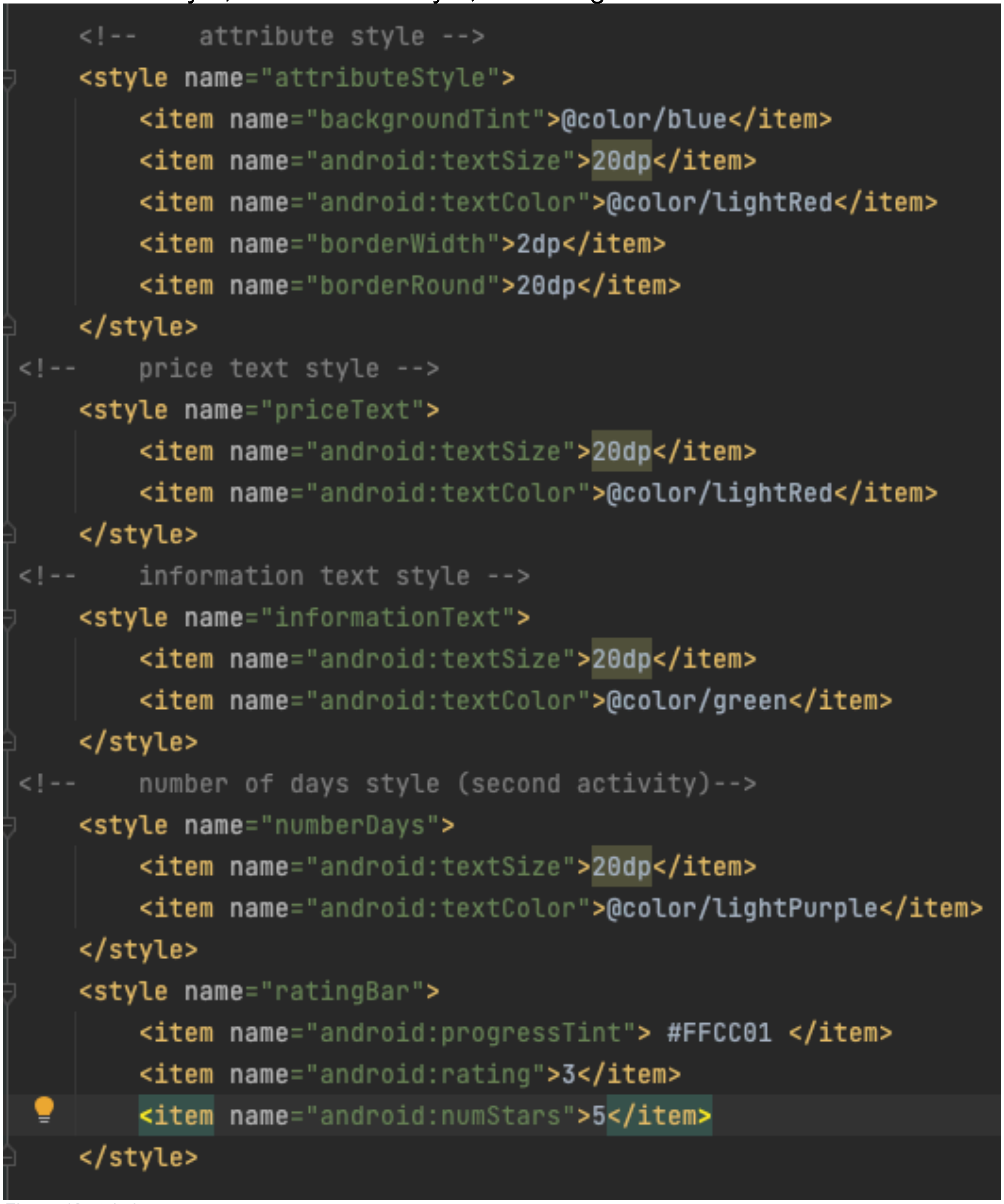
```
//      draw image
          val itemPrice = findViewById<TextView>(R.id.priceDis)
          val gameImage = listOf(
              R.drawable.cod,
              R.drawable.csgo,
              R.drawable.pubg,
              R.drawable.rr,
          )
      if( currentGame in 0 ≤ .. ≤ 3){
          image.setImageResource(gameImage[currentGame])
      }
```

Figure 11 handle image resource

In this approach, the code will be more clean and less redundant code rather than using when.

Gap 7: Style in app

For this app, im using a wide range of style for different component. Including the attribute style, the text view style, the rating bar



```
    <!--    attribute style -->
    <style name="attributeStyle">
        <item name="backgroundTint">@color/blue</item>
        <item name="android:textSize">20dp</item>
        <item name="android:textColor">@color/lightRed</item>
        <item name="borderWidth">2dp</item>
        <item name="borderRound">20dp</item>
    </style>
<!--    price text style -->
    <style name="priceText">
        <item name="android:textSize">20dp</item>
        <item name="android:textColor">@color/lightRed</item>
    </style>
<!--    information text style -->
    <style name="informationText">
        <item name="android:textSize">20dp</item>
        <item name="android:textColor">@color/green</item>
    </style>
<!--    number of days style (second activity)-->
    <style name="numberDays">
        <item name="android:textSize">20dp</item>
        <item name="android:textColor">@color/lightPurple</item>
    </style>
    <style name="ratingBar">
        <item name="android:progressTint"> #FFCC01 </item>
        <item name="android:rating">3</item>
        <item name="android:numStars">5</item>
    </style>
```

Figure 12 style in app

Gap 8: Sketch

For this app, I have sketch it on iPad, the interface will look like this:
First possible layout for the main activity and the second activity:

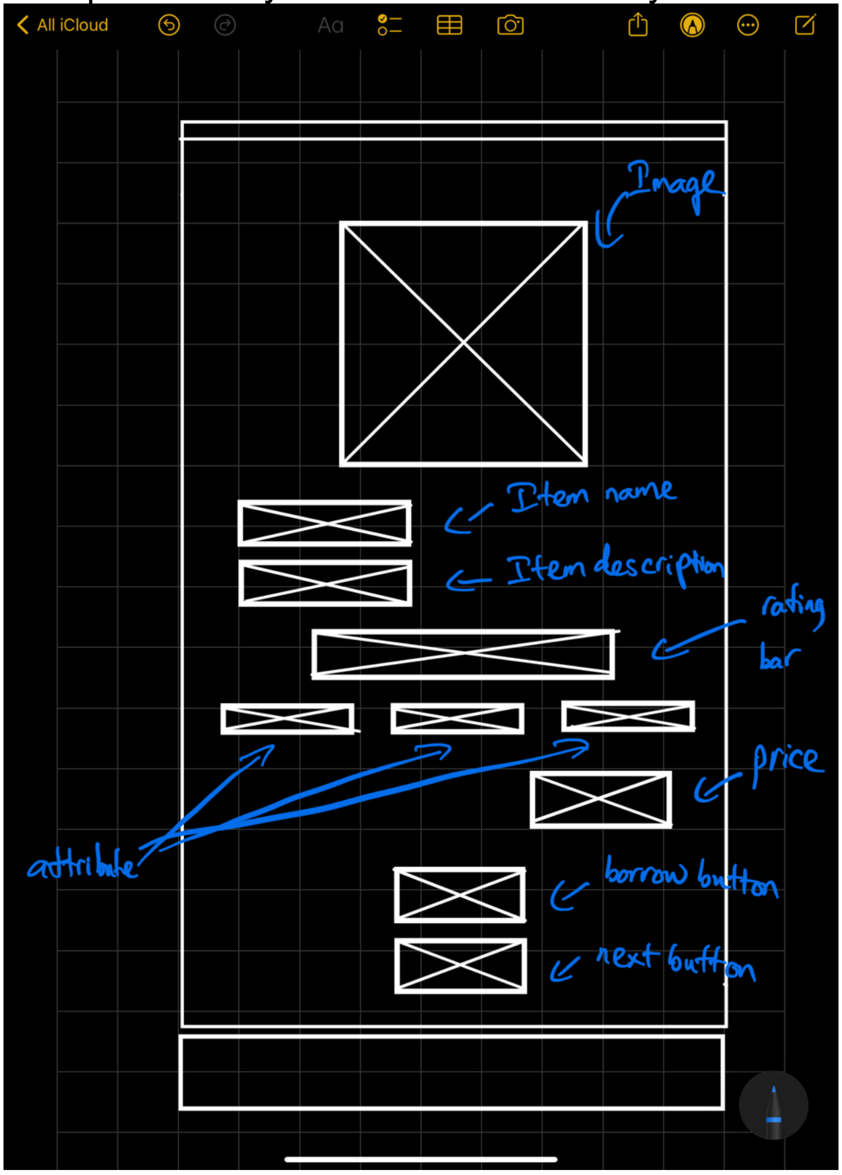


Figure 13 initial desgin

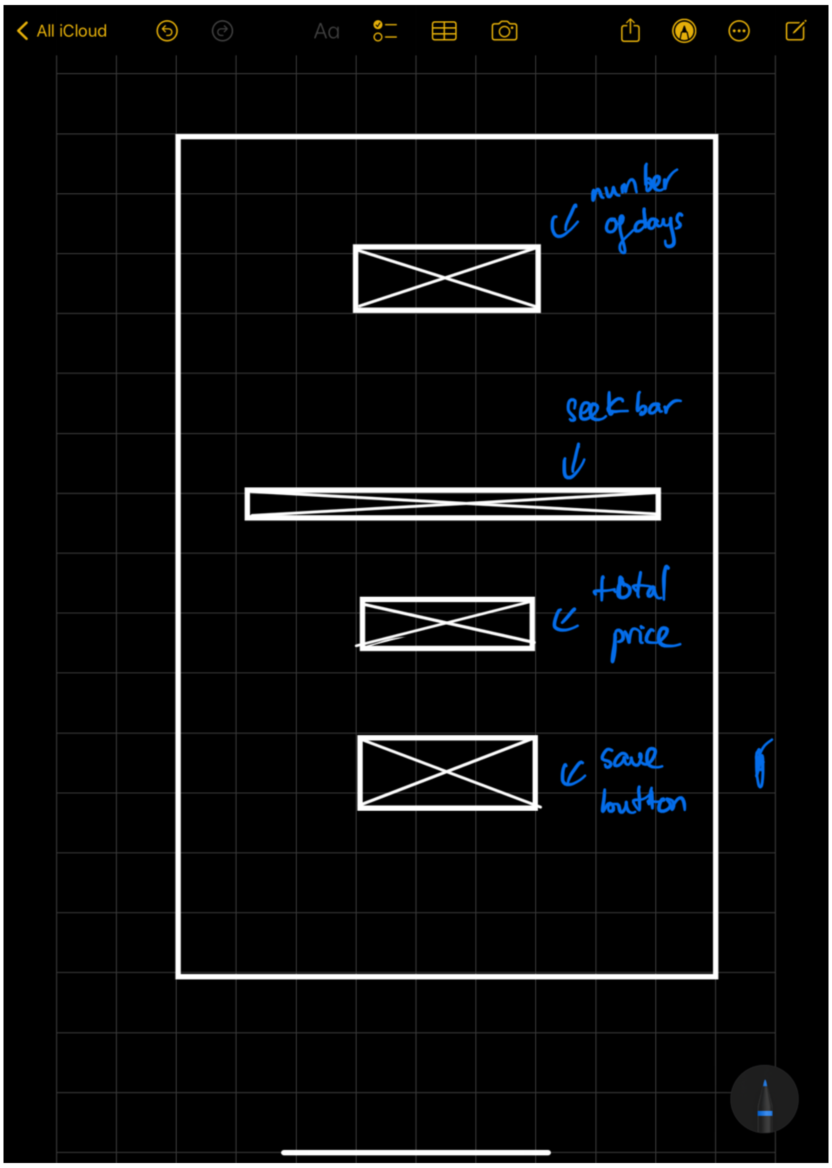


Figure 14 initial desgin

The second possible layout

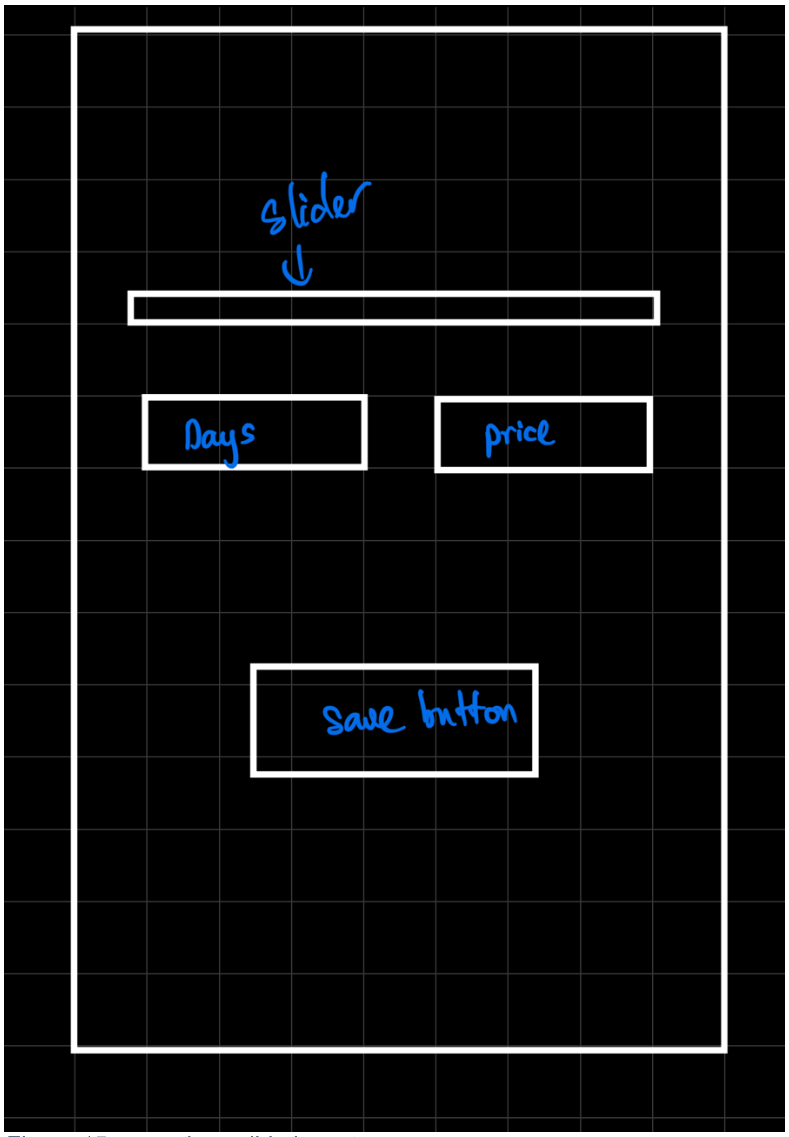


Figure 15 second possible layout

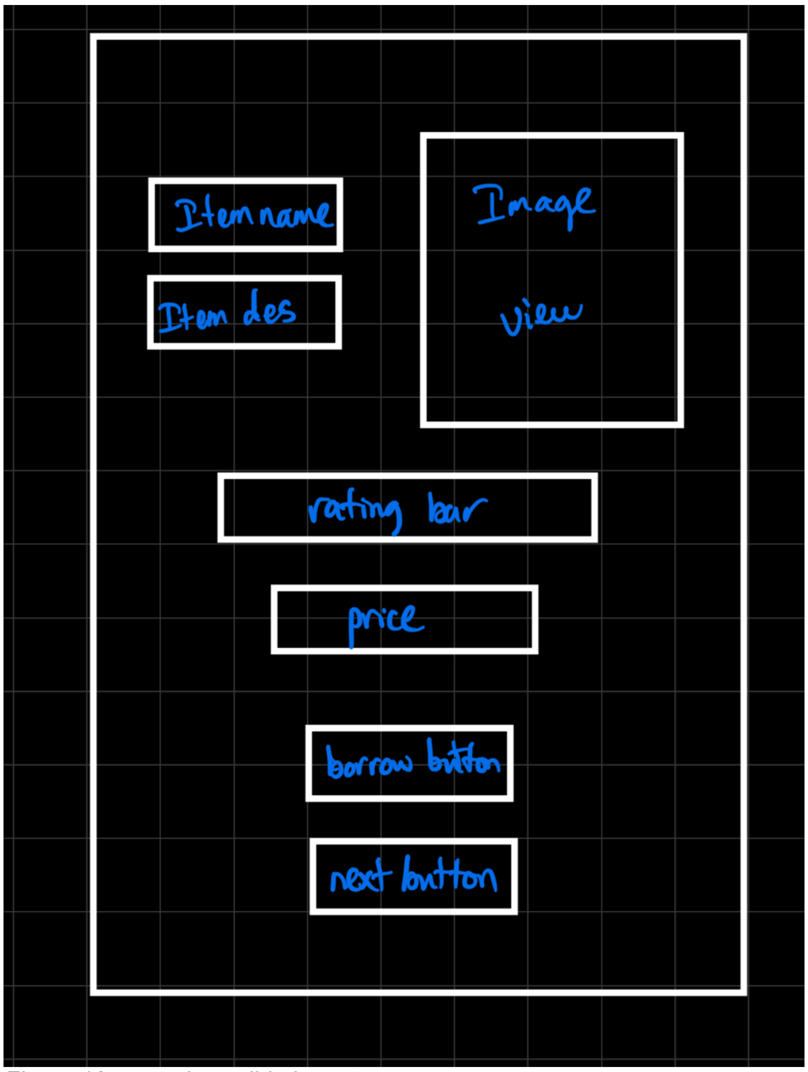


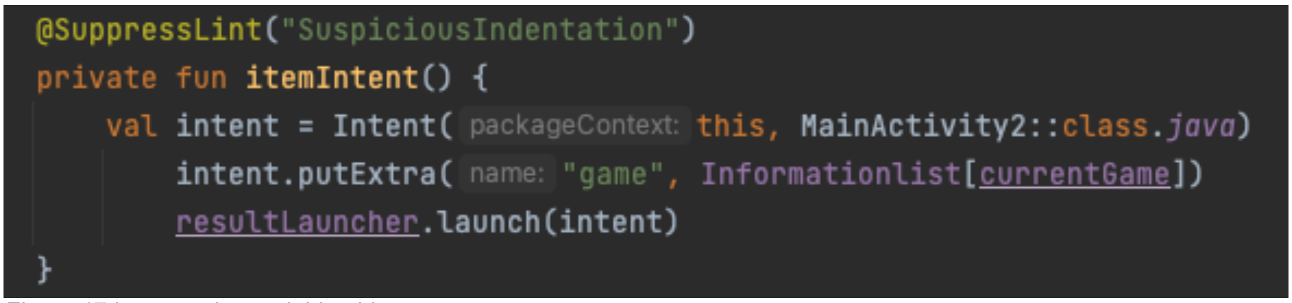
Figure 16 second possible layout

Considering both layout, the first layout seem to provide to the user more information than the sencod possible layout. Moreover, with a bigger space for image, that layout will fit both square or rectangle image while not being crop of. On the other hand, the space for image in layout two a little bit small, if it a rectangle image, the detail on the image will be small, provide a inconvenient for the user experience. After all, the first layout is the best choice for the app.

Gap 9: Intent and parceable object

In android development, intent is a fundamental component of the android operating system. Intent is also a mechanisim for different action happen on the program. Its can be used to pass the data and also lauch an activity:

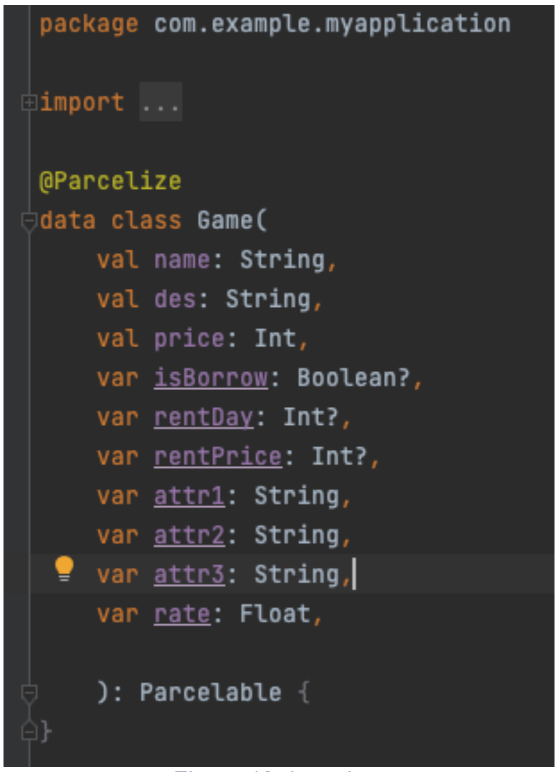
For example in the code below, the intent is being created in the current activity (activity 1) and the target component is the activity2
The intent here will being taking with the Extra data. In this extra data, game is the key name of the data, which used to identify the data, Informationlist[currentGame]: This is the value associated with the key. It is an element from a list called Informationlist at the index current Game



```
@SuppressLint("SuspiciousIndentation")
private fun itemIntent() {
    val intent = Intent( packageContext: this, MainActivity2::class.java)
        intent.putExtra( name: "game", Informationlist[currentGame])
        resultLauncher.launch(intent)
}
```

Figure 17 intent and parcelable object

Im also using a parcelable object. An object that implements the android.os.Parcelable interface is known as a Parcelable object. It enables efficient serialisation and deserialization of custom objects for use in passing them across various Android components and processes.
As here, the game data class will be implemented to be used as parcelable, so all the object create based on this data class will be consider as parcelable object.



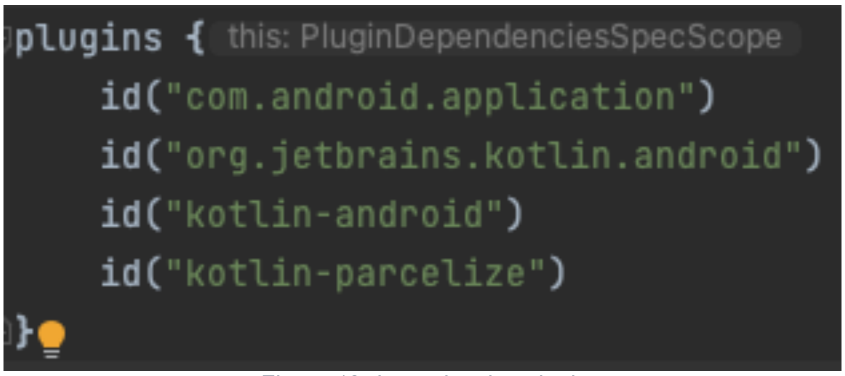
```kotlin
package com.example.myapplication

import ...

@Parcelize
data class Game(
    val name: String,
    val des: String,
    val price: Int,
    var isBorrow: Boolean?,
    var rentDay: Int?,
    var rentPrice: Int?,
    var attr1: String,
    var attr2: String,
    var attr3: String,
    var rate: Float,

    ): Parcelable {
}
```

Figure 18 data class

The advantage of Parcelable object: When transferring data through intents on Android, using Parcelable objects has many benefits. By contrast to conventional serialisation techniques, Parcelable objects are designed for Android's inter-component communication, enhancing performance and efficiency in the first place. Complex data structures may be efficiently transferred between tasks, services, or even between separate apps. Additionally, Parcelable objects are memory management optimised for Android, which is important for mobile devices with limited resources. Additionally, because the Parcelable mechanism is designed specifically for Android, it can be seamlessly integrated with system functions and services, making it a crucial option for effective data transfer and communication in Android app development.

Gap 10: Command of IDE

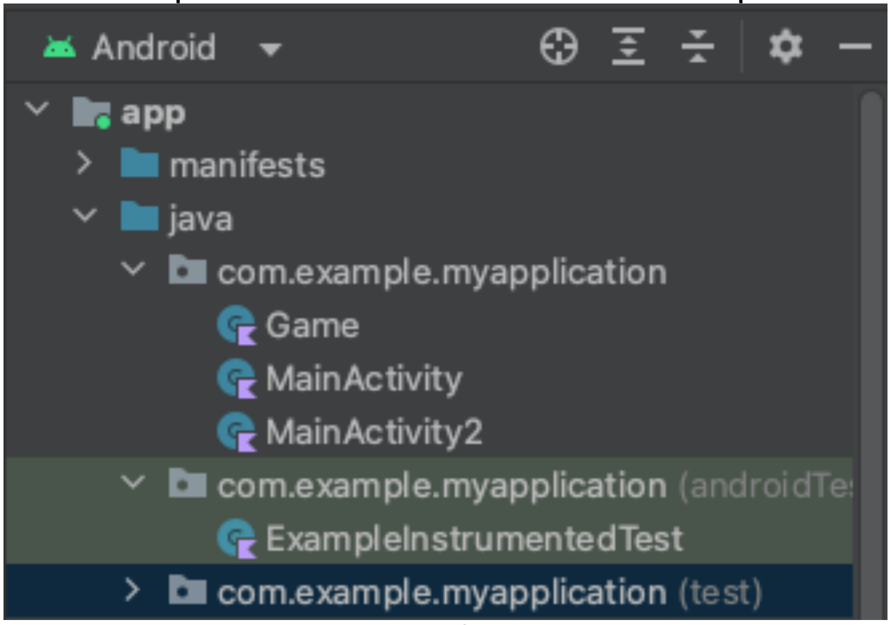
For is app to build successfully, we need to add more plugin dependencies To be more specific, kotlin-parcelize is being added, this is a Kotlin plugin specifically for enabling the @Parcelize annotation. The @Parcelize annotation is used to generate Parcelable implementations for Kotlin classes automatically



```
plugins { this: PluginDependenciesSpecScope
    id("com.android.application")
    id("org.jetbrains.kotlin.android")
    id("kotlin-android")
    id("kotlin-parcelize")
}
```

Figure 19 dependencies plugin

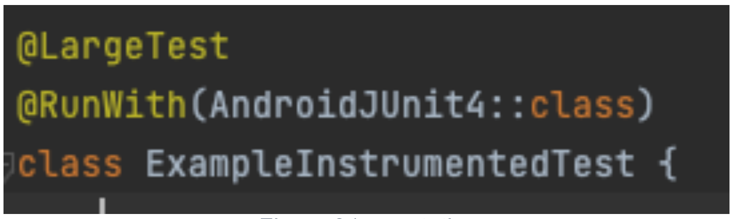
Moreover, this task require to write the test so this is the path to the test file



```
Android
app
  manifests
  java
    com.example.myapplication
      Game
      MainActivity
      MainActivity2
    com.example.myapplication (androidTe
      ExampleInstrumentedTest
    com.example.myapplication (test)
```

Figure 20 file path

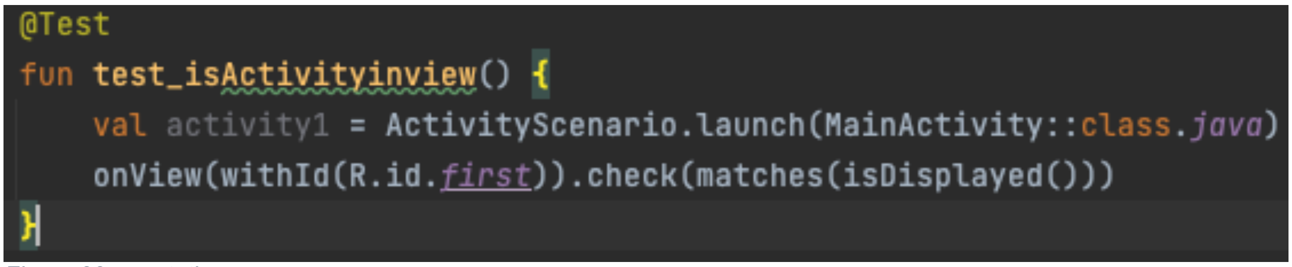
Inside the test file:
The @RunWith annotation in Android is used to specify the test runner class that should be used when running Android instrumentation tests. In code, @RunWith(AndroidJUnit4::class) is specifying the test runner class as AndroidJUnit4, which is a commonly used test runner for Android instrumented tests.



```
@LargeTest
@RunWith(AndroidJUnit4::class)
class ExampleInstrumentedTest {
```

Figure 21 annotation

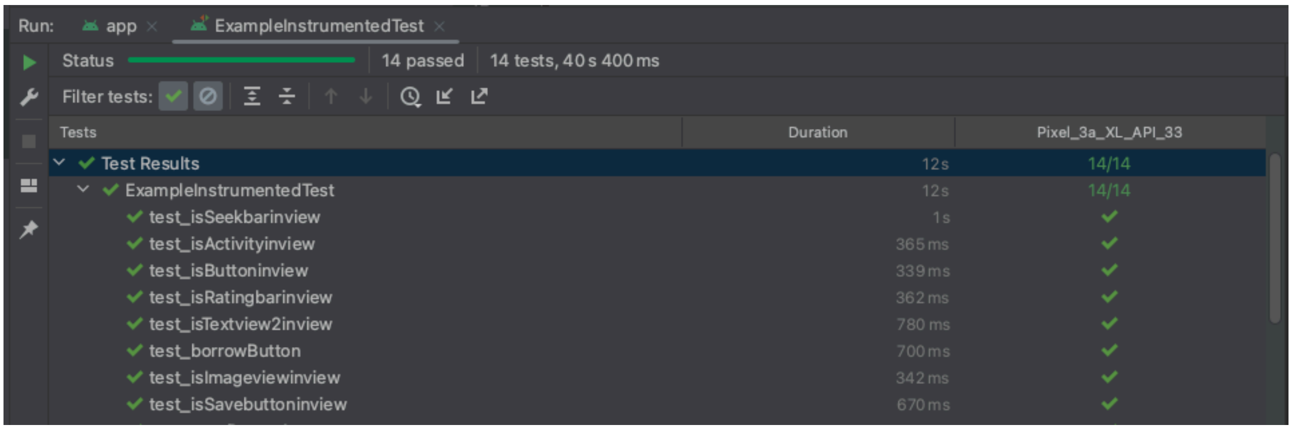
The @Test annotation used to mark a method as a test method. This annotation indicates that the method contains test logic that should be executed when running tests.



```
@Test
fun test_isActivityinview() {
    val activity1 = ActivityScenario.launch(MainActivity::class.java)
    onView(withId(R.id.first)).check(matches(isDisplayed()))
}
```

Figure 22 annotation

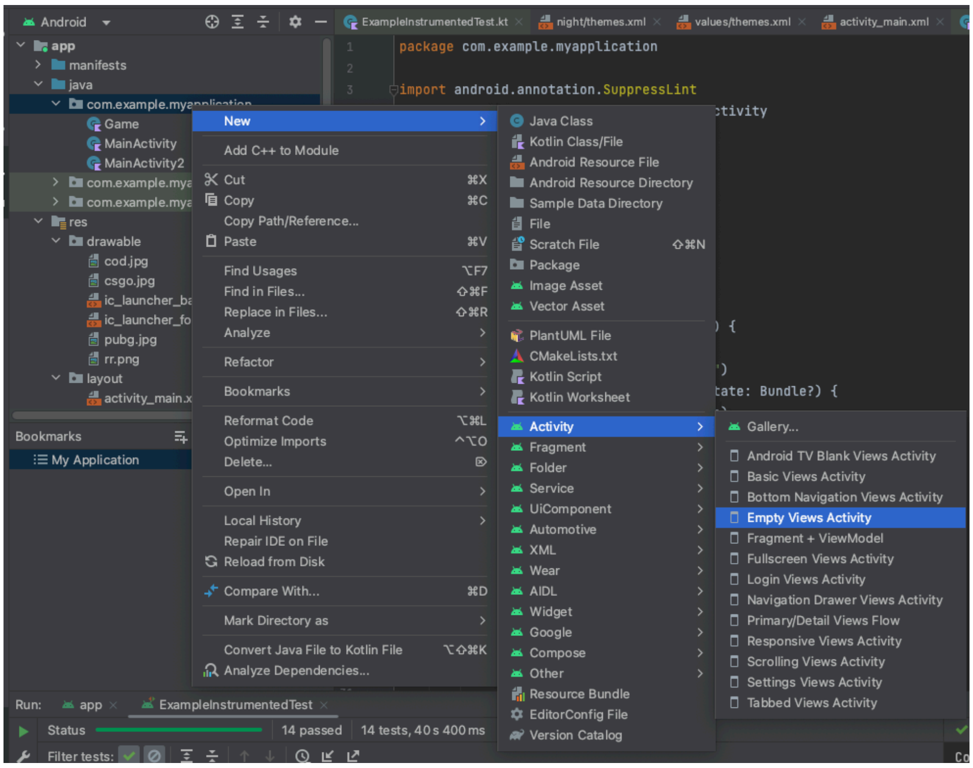
We can see the test result here:



```
Run:  app ×   ExampleInstrumentedTest ×
Status            14 passed   14 tests, 40 s 400 ms
Filter tests:
Tests                                   Duration   Pixel_3a_XL_API_33
✓ Test Results                              12 s   14/14
  ✓ ExampleInstrumentedTest                 12 s   14/14
    ✓ test_isSeekbarinview                   1 s   ✓
    ✓ test_isActivityinview               365 ms   ✓
    ✓ test_isButtoninview                 339 ms   ✓
    ✓ test_isRatingbarinview              362 ms   ✓
    ✓ test_isTextview2inview              780 ms   ✓
    ✓ test_borrowButton                   700 ms   ✓
    ✓ test_isImageviewinview              342 ms   ✓
    ✓ test_isSavebuttoninview             670 ms   ✓
```

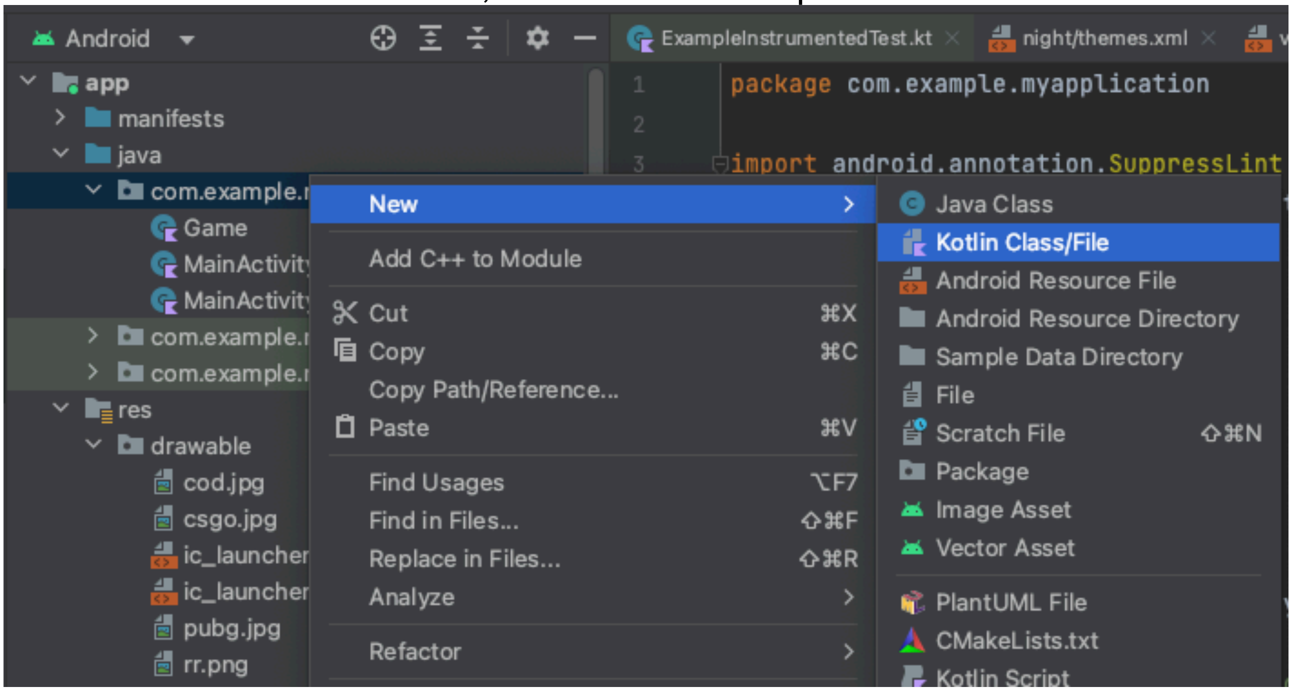
Figure 23 result test

To create another activity, we will follow this path:



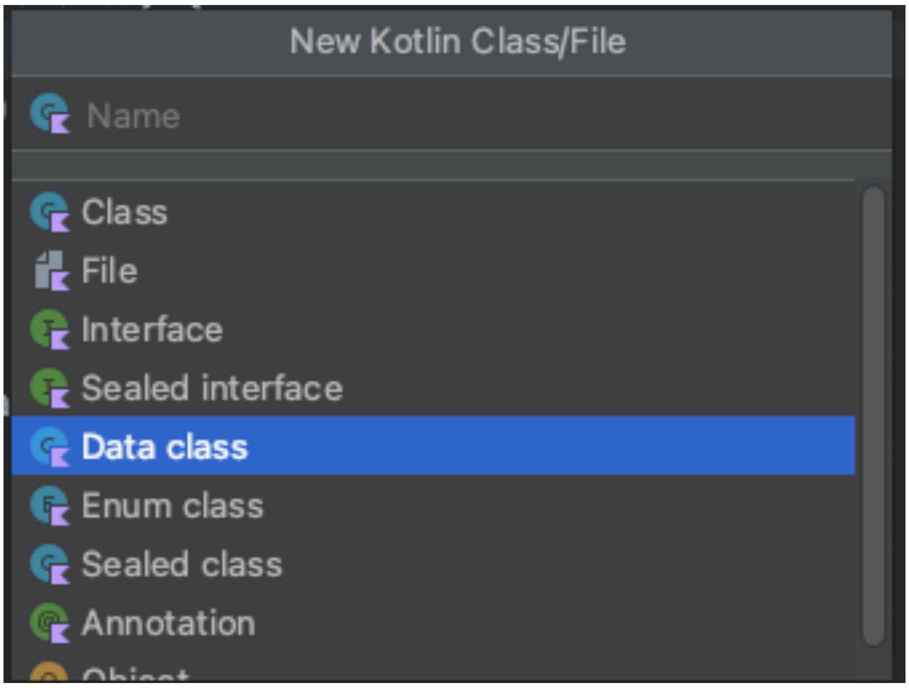
```
package com.example.myapplication

import android.annotation.SuppressLint
```

To create the new data class, we will follow this path:

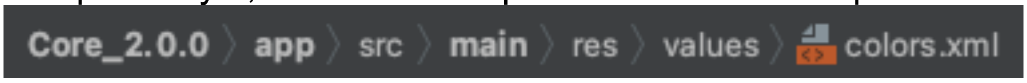


```
package com.example.myapplication

import android.annotation.SuppressLint
```

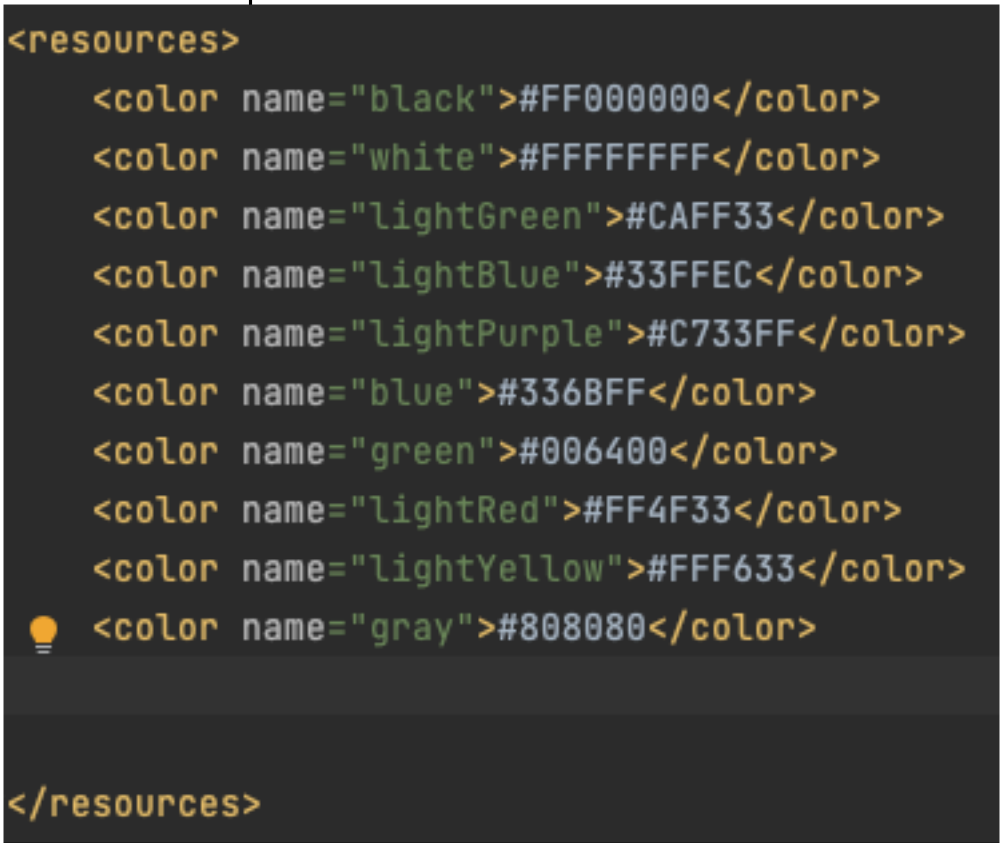
Also choose the data class and rename it:



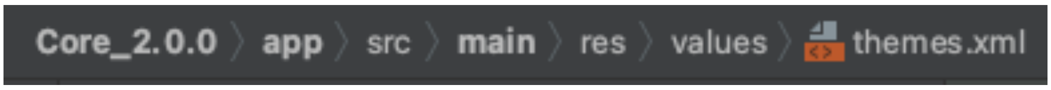
Also to set up the style, first I will set up the color in this file path:


Core_2.0.0 〉 app 〉 src 〉 main 〉 res 〉 values 〉 colors.xml

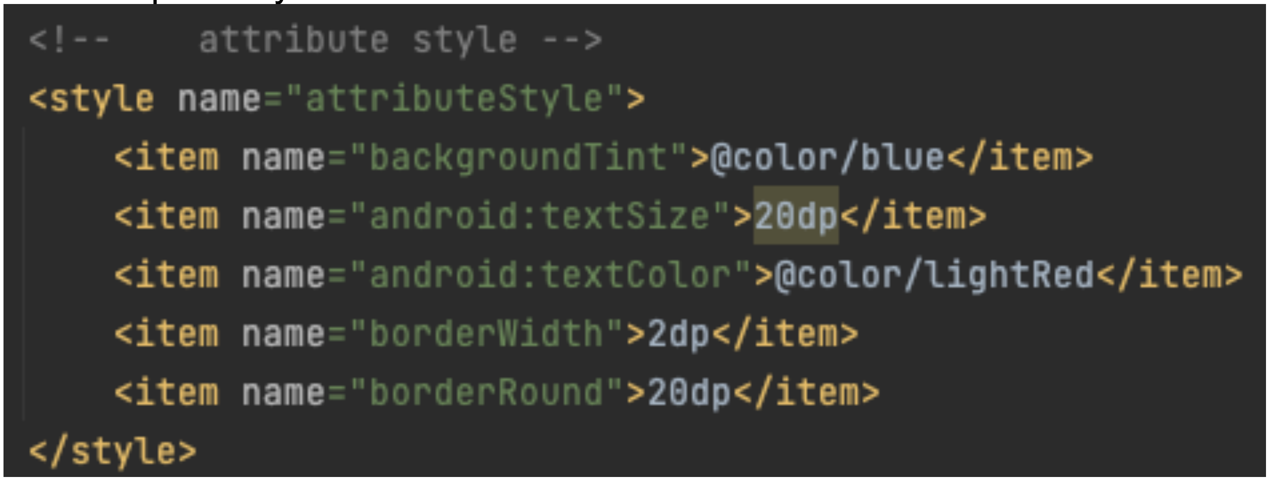
After that set up the color:


```
<resources>
    <color name="black">#FF000000</color>
    <color name="white">#FFFFFFFF</color>
    <color name="lightGreen">#CAFF33</color>
    <color name="lightBlue">#33FFEC</color>
    <color name="lightPurple">#C733FF</color>
    <color name="blue">#336BFF</color>
    <color name="green">#006400</color>
    <color name="lightRed">#FF4F33</color>
    <color name="lightYellow">#FFF633</color>
    <color name="gray">#808080</color>



</resources>
```

And then used it in a file style follow this path:


Core_2.0.0 〉 app 〉 src 〉 main 〉 res 〉 values 〉 themes.xml

And setup the style:



```xml
<!--    attribute style -->
<style name="attributeStyle">
    <item name="backgroundTint">@color/blue</item>
    <item name="android:textSize">20dp</item>
    <item name="android:textColor">@color/lightRed</item>
    <item name="borderWidth">2dp</item>
    <item name="borderRound">20dp</item>
</style>
```

Open Issues and Recommendations

The only open issue it that I can not passing the data from second acitivity back as a form of parcelable object, I have tried but its keep getting error, so instead, I choose to pass like normal data though intent. This will create redundant code.
In the future, I will address this problem.