CS4131
Mobile Application Development

Name: _________________________________ (   ) Date: ________________

Chapter 6: Intents on Fragments and Activities; Navigation

6.1 Introduction to Chapter 6

Intents are messaging objects which are used to **request an action**, and possibly **perform some form of data transfer** from one app or component to another. Although intents facilitate communication between communication between components in several ways, there are some fundamental uses:

- Starting a new activity, with or without expecting a result from a preceding activity
- Starting a service (a component that performs operations in the background)
- Delivering a broadcast which either any app or other phones can receive

Examples on how to create simple apps which apply intents can be found in Chapter 1.

There are two types of intents. **Explicit intents** specify which application will satisfy the intent, by supplying the target's app's package name or a fully qualified component class name. **Implicit intents** do not name a specific component, but instead declare a general action to perform, which allows a component from another app to handle it.

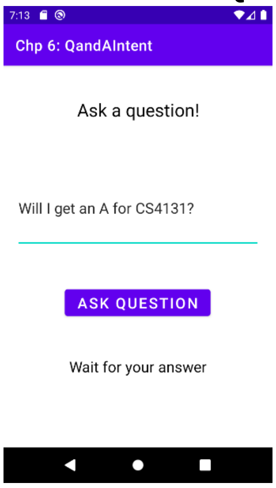
6.2 Explicit Intents

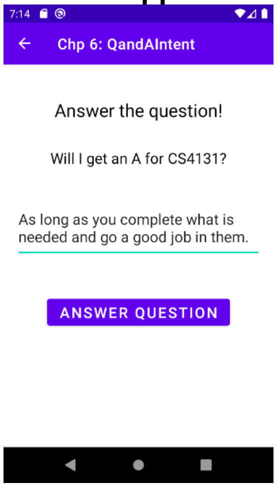
Explicit intents are usually used to start a component in your own app, like an activity in response to a user action or a service to download a file in the background, because **you know the exact class name** of the activity or service you want to start. The following example shows how an explicit intent is applied in the app to communicate, and transfer data, between activities.

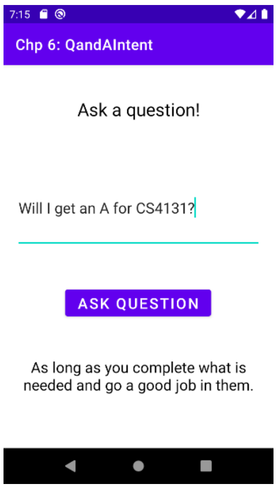
Example 1: QandAIntent

This opening page of the app will allow the user to type a question and click the button to ask the question, which will go to the next page (activity). The question can then be answered, and the button is clicked on to answer the question. Notice that the back button is present in the "answer page" of the app

Screenshots of QandAIntent app


7:13
Chp 6: QandAIntent
Ask a question!
Will I get an A for CS4131?
ASK QUESTION
Wait for your answer


7:14
Chp 6: QandAIntent
Answer the question!
Will I get an A for CS4131?
As long as you complete what is needed and go a good job in them.
ANSWER QUESTION


7:15
Chp 6: QandAIntent
Ask a question!
Will I get an A for CS4131?
ASK QUESTION
As long as you complete what is needed and go a good job in them.

strings.xml

```
<resources>
    <string name="app_name">Chp 6: QandAIntent</string>
    <string name="promptQn">Ask a question!</string>
    <string name="promptAns">Answer the question!</string>
    <string name="entertext">Enter Text</string>
    <string name="initalAns">Wait for your answer</string>
    <string name="initalQn">No question has been asked</string>
    <string name="qnBtn">Ask Question</string>
    <string name="ansBtn">Answer Question</string>
</resources>
```

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.qandaintent">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.QandAIntent">
        <activity android:name=".MainActivity2" android:parentActivityName=".MainActivity"
            android:exported="false"/>
        <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>
    </application>

</manifest>
```

Explanation:
If you have more than one activity present, ensure that your Android manifest has both activities present. If you wish to put the “back” icon on your action bar, you will have to define the “parent activity” so that the compiler knows how to navigate from one activity to another.

MainActivity.kt

```
import android.app.Activity
import android.content.Context
import android.content.Intent
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.view.MotionEvent
import android.view.inputmethod.InputMethodManager
import android.widget.Button
import android.widget.EditText
import android.widget.TextView
import androidx.activity.result.contract.ActivityResultContracts

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

18          var resultLauncher =
19              registerForActivityResult(ActivityResultContracts.StartActivityForResult()){
20                 result ->
21                  if (result.resultCode == Activity.RESULT_OK){
22                      val data: Intent? = result.data
23                      val ansTextView = findViewById<TextView>(R.id.ansTextView)
24                      val ansString = data?.extras?.getString("ansString")
25                      ansTextView.text = ansString
26                  }
27              }
28          val askBtn : Button = findViewById(R.id.buttonQn)
29          askBtn.setOnClickListener{ view ->
30              val qnString = findViewById<EditText>(R.id.editTextQn).text.toString()
31              val intent = Intent(this, MainActivity2::class.java)
32              intent.putExtra("qString", qnString)
33              resultLauncher.launch(intent)
34          }
35      }
36      // The method will force the touch keyboard to hide when user touches anywhere on screen
37      override fun onTouchEvent(event: MotionEvent?): Boolean {
38          val imm = getSystemService(Context.INPUT_METHOD_SERVICE) as InputMethodManager
39          if(imm.isAcceptingText) imm.hideSoftInputFromWindow(currentFocus!!.windowToken, 0)
40          return true
41      }
42  }
```

Explanation:

Line 18 to 27	The resultLauncher gets the activity ready to collect the result from MainActivity2, such that when the obtained result code is valid, which will be present in a successful resultant intent passed from MainActivity2, the data from the resultant intent (the answer to the question) will be displayed in the TextView. As the ansString is stored as a **key-value** pair in the intent, to obtain the ansString value, you will need to use the key “ansString” to obtain it.
Line 28 to 34	When the button is clicked, a listener is set up such that it readies the intent, stores the data and launches the intent. Note that in line 31, **the destination of the intent is known**, hence making this an **explicit intent**. The putExtra() method adds in a **key-value pair**, where the qnString (value) will be identified by “qString” (key) when the intent is processed by MainActivity2.

activity_main.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"
    android:paddingLeft="16dp"
    android:paddingTop="16dp"
    android:paddingRight="16dp"
    android:paddingBottom="16dp"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/ansTextView"
        android:layout_width="0dp"
        android:layout_height="110dp"
        android:layout_marginTop="48dp"
        android:text="@string/initalAns"
        android:textAlignment="center"
        android:textColor="@color/black"
        android:textSize="20sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/buttonQn"
        app:layout_constraintVertical_bias="0.0" />

    <EditText
        android:id="@+id/editTextQn"
        android:layout_width="0dp"
        android:layout_height="110dp"
        android:text="@string/entertext"
        android:textSize="20sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/textView"
        app:layout_constraintVertical_bias="0.091" />

    <TextView
        android:id="@+id/textView"
        android:layout_width="237dp"
        android:layout_height="68dp"
        android:text="@string/promptQn"
        android:textAlignment="center"
        android:textColor="@color/black"
        android:textSize="24sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.066" />

    <Button
        android:id="@+id/buttonQn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="48dp"
        android:text="@string/qnBtn"
        android:textSize="20sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.497"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/editTextQn"
        app:layout_constraintVertical_bias="0.0" />

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

MainActivity2.kt

```
import android.content.Context
import android.content.Intent
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.view.MotionEvent
import android.view.View
import android.view.inputmethod.InputMethodManager
import android.widget.Button
import android.widget.EditText
import android.widget.TextView

```

```
12  class MainActivity2 : AppCompatActivity() {
13      override fun onCreate(savedInstanceState: Bundle?) {
14          super.onCreate(savedInstanceState)
15          setContentView(R.layout.activity_main2)
16
17          val extras = intent.extras ?: return
18          val textView = findViewById<TextView>(R.id.qnTextView)
19          textView.text = extras.getString("qString")
20
21          val answerBtn = findViewById<Button>(R.id.buttonAns)
22          answerBtn.setOnClickListener{ view ->
23              finish()
24          }
25      }
26
27      // The method will force the touch keyboard to hide when user touches anywhere on screen
28      override fun onTouchEvent(event: MotionEvent?): Boolean {
29          val imm = getSystemService(Context.INPUT_METHOD_SERVICE) as InputMethodManager
30          if(imm.isAcceptingText) imm.hideSoftInputFromWindow(currentFocus!!.windowToken, 0)
31          return true
32      }
33
34      override fun finish(){
35          val data = Intent()
36          val ansString = findViewById<EditText>(R.id.editTextAns).text.toString()
37          data.putExtra("ansString", ansString)
38          setResult(RESULT_OK, data)
39          super.finish()
40      }
41  }
```

Explanation:

Line 17 to 19	The data from the intent passed from MainActivity is retrieved from the extras (recall the putExtra() from MainActivity code) and note that the code will force a “stop” to the processing if there is no such extra data. The value of the key “qString” is obtained from the extras and placed into the TextView
Line 21 to 24 Line 34 to 40	When the answer button is clicked on, a call is made to finish up the activity on the Activity Lifecycle. Within the overridden finish() method, and intent is passed with an extra **key-value pair** containing the ansString (value) with the key “ansString”. The result is set to RESULT_OK so that the MainActivity knows that the correct result is given in the intent from MainActivity2. Note that finish() in the superclass must be called.

6.3 Implicit Intents

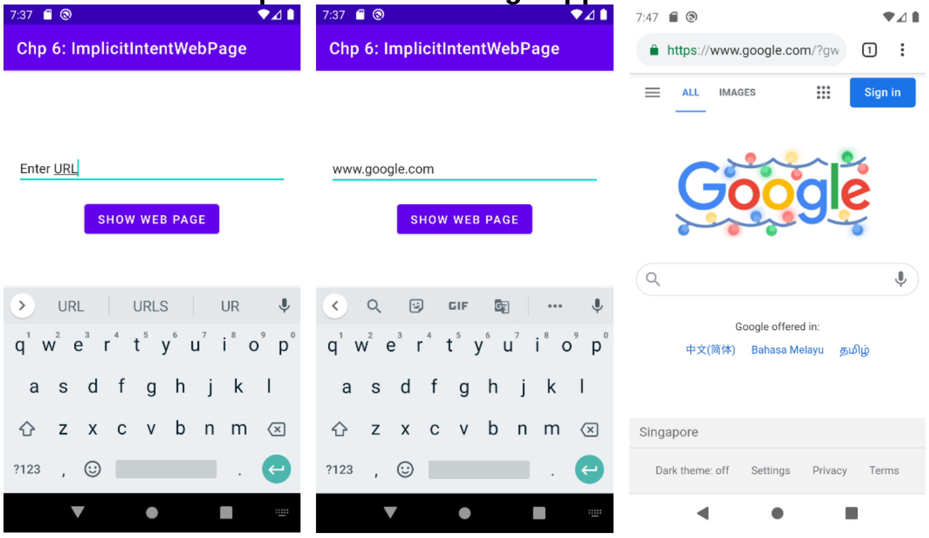
Implicit intents differ from explicit intents in a way that there are no specific components called to perform the action, and instead, it is left to the Android system to decide what is the most appropriate component to handle the task. The Android system compares the contents of the intent to **intent filters** declared in the **manifest file** of other apps on the device and if there is a match, the **system will start that component and deliver the Intent object to that component**. If multiple filters are compatible, the system will display a dialog for the user to pick which app to use. An intent filter is an expression in an app's manifest file that specifies the type of intents that the component would like to receive. **Note that if you do not declare any intent filters for an activity, then it can be started only with an explicit intent.**

The following example will show an implicit intent call to open a web page.

<u>Example 2: ImplicitIntentWebPage</u>

This app will take the user's input into the EditText for a website and after the "Show Web Page" button is clicked on, the website will show. Note that this app allows for automatic insertion of https:// if it is not given.

Screenshots of ImplicitIntentWebPage app


Chp 6: ImplicitIntentWebPage
Enter URL
SHOW WEB PAGE
Chp 6: ImplicitIntentWebPage
www.google.com
SHOW WEB PAGE
https://www.google.com/?gw
ALL
IMAGES
Sign in
Google offered in:
中文(简体) Bahasa Melayu தமிழ்
Singapore
Dark theme: off Settings Privacy Terms

strings.xml

```
<resources>
    <string name="app_name">Chp 6: ImplicitIntentWebPage</string>
    <string name="enterURL">Enter URL</string>
    <string name="webBtn">Show Web Page</string>
</resources>
```

MainActivity.kt

```
import android.content.Context
import android.content.Intent
import android.net.Uri
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.view.MotionEvent
import android.view.View
import android.view.inputmethod.InputMethodManager
import android.widget.EditText

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

    fun showWebPage(view : View){
        var webURL = findViewById<EditText>(R.id.editText).text.toString()
        if (!webURL.startsWith("http://") && !webURL.startsWith("https://"))
            webURL = "http://$webURL"
        val intent = Intent(Intent.ACTION_VIEW, Uri.parse(webURL))
        startActivity(intent)
    }

    // The method will force the touch keyboard to hide when user touches anywhere on screen
    override fun onTouchEvent(event: MotionEvent?): Boolean {
```

```
27          val imm = getSystemService(Context.INPUT_METHOD_SERVICE) as InputMethodManager
28          if(imm.isAcceptingText) imm.hideSoftInputFromWindow(currentFocus!!.windowToken, 0)
29          return true
30      }
31  }
```

Explanation:

Line 17 to 23	The listener code will first check if the substring https:// is inside the URL given. If it is not, it is automatically appended in front of the given URL and an intent is passed to open the URL through the Uri.parse() method. Note that the first parameter is known as an action parameter which will define the intended action to be performed. The list of possible actions for implicit intents will be covered in the next section. Note the difference in how the implicit intent is called here versus the explicit intent in Example 1

activity_main.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"
    android:padding="16dp"
    tools:context=".MainActivity">

    <EditText
        android:id="@+id/editText"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/enterURL"
        android:textSize="16sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.186" />

    <Button
        android:id="@+id/webBtn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="16dp"
        android:onClick="showWebPage"
        android:text="@string/webBtn"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.498"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/editText"
        app:layout_constraintVertical_bias="0.0" />

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

6.4 Application of Intents: Android Internal Services

As observed in the previous section's example, an implicit intent is called, defining the action to be taken to view the URL parsed from the string given in the EditText. There are in fact, many other **common actions** which can be performed under different calls of the implicit intent, all of which will call upon common phone apps to perform required tasks, shown in the following table:

Do ensure to take note of which implicit intents **require permissions. If they do, the coding is not as simple as just passing the implicit intent and will be covered in the next chapter.

Implicit Intent Action	**Description**
ACTION_SET_ALARM	Creation of a new alarm and specify alarm details such as the time and message using extras
ACTION_SET_TIMER	Creation of countdown timer and specify timer details such as the duration using extras
ACTION_INSERT	Add new event to user's calendar. Specify the data URI with Events.CONTENT_URI and then specify various event details using extras
ACTION_IMAGE_CAPTURE	Open camera app to catch image and specify the URI location where you'd like the camera to save the photo, in the EXTRA_OUTPUT extra. *****Permissions need to be granted to access camera and writing of data***
ACTION_VIDEO_CAPTURE	Open camera app to catch video and specify the URI location where you'd like the camera to save the video, in the EXTRA_OUTPUT extra. *****Permissions need to be granted to access camera and writing of data***
ACTION_PICK *ACTION_EDIT*	User can select (or also edit) contact information. MIME type can be set to pick contacts from different mediums: • **Pick from contacts with phone number:** CommonDataKinds.Phone.CONTENT_TYPE • **Pick from contacts with email address:** CommonDataKinds.Email.CONTENT_TYPE • **Pick from contacts with postal address:** CommonDataKinds.StructuredPostal.CONTENT_TYPE *****Permissions need to be granted to read contacts***
ACTION_GET_CONTENT *ACTION_OPEN_DOCUMENT* *ACTION_CREATE_DOCUMENT*	File operations such as selecting a file and returning the reference, or even creating or directly opening the file can be done here. ****Permissions need to be granted to read and/or write files.**
ACTION_CREATE_NOTE	Create a new note and specify note details such as the subject and text using extras.

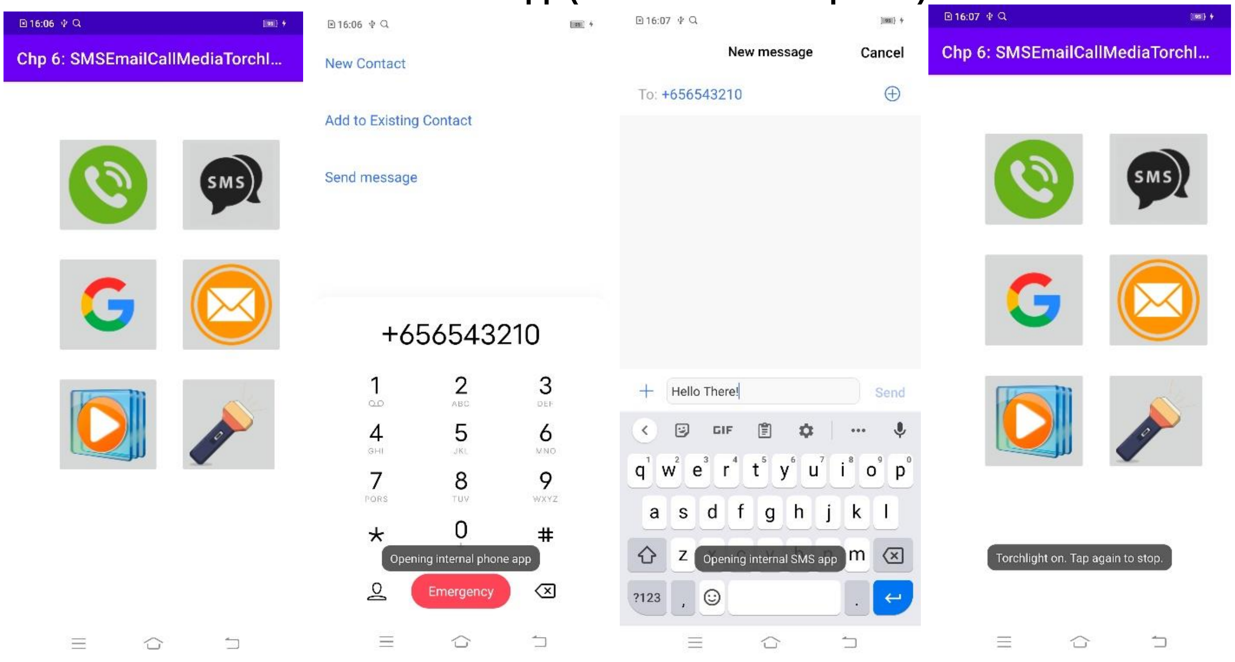
	*****It is required that apps ask for confirmation from user BEFORE action is completed.***
ACTION_DIAL *ACTION_CALL*	ACTION_DIAL merely opens the phone app and displays the phone number specified in the data URI scheme, and the user must press the call button. ACTION_CALL directly calls the phone number specified the data URI. **This require call permissions to be granted.** Data URI needs to be in the format, tel:<*phone number*> OR voicemail:<*phone number*>
ACTION_WEB_SEARCH	Search for something by specifying the search string in the SearchManager.QUERY extra.
ACTION_SETTINGS *ACTION_WIRELESS_SETTINGS* *ACTION_AIRPLANE_MODE_SETTINGS* *ACTION_WIFI_SETTINGS* *ACTION_APN_SETTINGS* *ACTION_BLUETOOTH_SETTINGS* *ACTION_DATE_SETTINGS* *ACTION_LOCALE_SETTINGS* *ACTION_INPUT_METHOD_SETTINGS* *ACTION_DISPLAY_SETTINGS* *ACTION_SECURITY_SETTINGS* *ACTION_LOCATION_SOURCE_SETTINGS* *ACTION_INTERNAL_STORAGE_SETTINGS* *ACTION_MEMORY_CARD_SETTINGS*	Open a screen in the system settings when your app requires the user to change something, respective to the action name. *****Permissions need to be granted for some of the settings***
ACTION_SENDTO *ACTION_SEND* *ACTION_SEND_MULTIPLE*	Initiate composing of an email (action will differ depending on number of attachments) and specify recipient details and subject in the extras. Also initiate composing of an SMS or MMS message and specify message details such as the phone number, subject, and message body using the extras. Note that the data URI needs to be in the format, smsto:<*phone number*> OR mmsto:<*phone number*>
ACTION_VIEW	Open a web page and specify the web URL in the intent data This can also apply to media files, such as audio and video files, as well as open a map. ***If files stored in the phone running the app are to be run, permissions need to be granted to read files.***

You may read more about the common implicit intents here:
https://developer.android.com/guide/components/intents-common

Example 3: SMSEmailCallIntent

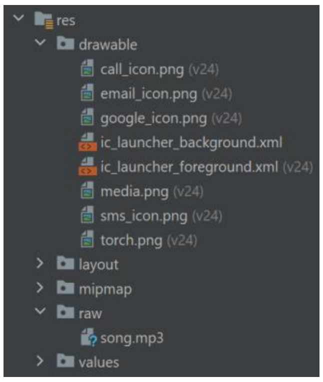
This app features buttons which calls for various implicit intents to occur, such as call, SMS, opening Google, email, playing media as well as the utilizing the phone's flash as a torch. For the media and torch functions, tapping the button alternates between switching it on or off.

Screenshots of SMSEmailCallIntent app (taken from actual phone)



strings.xml

```
<resources>
    <string name="app_name">Chp 6: SMSEmailCallMediaTorchIntent</string>
    <string name="call">Opening internal phone app</string>
    <string name="sms">Opening internal SMS app</string>
    <string name="google">Opening Google search</string>
    <string name="email">Opening Email app</string>
    <string name="mediaStart">Playing song now. Tap again to stop.</string>
    <string name="mediaStop">Stopped playing song.</string>
    <string name="torchStart">Torchlight on. Tap again to stop.</string>
    <string name="torchStop">Torchlight off..</string>
    <string name="torchFail">Torchlight unavailable</string>
    <string name = "fail">Operation failed...</string>
</resources>
```



```
res
  drawable
    call_icon.png (v24)
    email_icon.png (v24)
    google_icon.png (v24)
    ic_launcher_background.xml
    ic_launcher_foreground.xml (v24)
    media.png (v24)
    sms_icon.png (v24)
    torch.png (v24)
  layout
  mipmap
  raw
    song.mp3
  values
```

MainActivity.kt

```kotlin
import android.content.Context
import android.content.Intent
import android.media.MediaPlayer
import android.net.Uri
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.view.View
import android.widget.Toast
import android.content.pm.PackageManager
import android.hardware.camera2.CameraManager

class MainActivity : AppCompatActivity() {
    private val phoneNum = "+656543210"
    private val smsNum = "+656543210"
    private val smsMsg = "Hello There!"
    private val emailSubj = "Hello From MyApp"
    private val emailMsg = "This is a greeting email! Hello There!"
    private val webURL = "http://www.google.com"
    private val emailAdd = "nhsljyl@nus.edu.sg"
    private lateinit var mediaPlayer : MediaPlayer
    private var mediaPlaying = false
    private var lighted = false

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        mediaPlayer = MediaPlayer.create(this, R.raw.song)
    }

    fun processOnClick(view : View){
        lateinit var intent : Intent
        try {
            var toastText : String
            when (view.id) {
                R.id.webBtn -> {
                    intent = Intent(Intent.ACTION_VIEW, Uri.parse(webURL))
                    toastText = getString(R.string.google)
                }
                R.id.smsBtn -> {
                    intent = Intent(Intent.ACTION_SENDTO)
                    intent.data = Uri.parse("smsto:$smsNum")
                    intent.putExtra("sms_body", smsMsg)
                    toastText = getString(R.string.sms)
                }
                R.id.phoneBtn -> {
                    intent = Intent(Intent.ACTION_DIAL)
                    intent.data = Uri.parse("tel:$phoneNum")
                    toastText = getString(R.string.call)
                }
                R.id.emailBtn -> {
                    intent = Intent(Intent.ACTION_SENDTO)
                    intent.data = Uri.parse("mailto:")
                    intent.putExtra(Intent.EXTRA_EMAIL, emailAdd)
                    intent.putExtra(Intent.EXTRA_SUBJECT, emailSubj)
                    intent.putExtra(Intent.EXTRA_TEXT, emailMsg)
                    toastText = getString(R.string.email)
                }
                R.id.mediaBtn -> {
                    if (mediaPlaying){
                        mediaPlayer.stop()
                        getString(R.string.mediaStop)
                        mediaPlaying = false
                        toastText = getString(R.string.mediaStop)
                    } else {
                        mediaPlayer = MediaPlayer.create(this, R.raw.song)
                        mediaPlayer.start()
                        getString(R.string.mediaStart)
                        mediaPlaying = true
                        toastText = getString(R.string.mediaStart)
                    }
                }
                R.id.torchBtn -> {
                    val isFlashAvailable = applicationContext.packageManager
                        .hasSystemFeature(PackageManager.FEATURE_CAMERA_FLASH)
                    if (!isFlashAvailable) {
                        toastText = getString(R.string.torchFail)
                    } else {
                        try {
                            val mCameraManager = getSystemService(Context.CAMERA_SERVICE)
                                                        as CameraManager
                            val mCameraID = mCameraManager.cameraIdList[0]
```

```
82                              if (!lighted){
83                                  mCameraManager.setTorchMode(mCameraID, true)
84                                  toastText = getString(R.string.torchStart)
85                                  lighted = true
86                              } else {
87                                  mCameraManager.setTorchMode(mCameraID, false)
88                                  toastText = getString(R.string.torchStop)
89                                  lighted = false
90                              }
91                          } catch (e : Exception){
92                              toastText = getString(R.string.torchFail)
93                              e.printStackTrace()
94                          }
95                      }
96                  }
97                  else -> {
98                      toastText = getString(R.string.fail)
99                  }
100             }
101             Toast.makeText(this@MainActivity, toastText, Toast.LENGTH_SHORT).show()
102             if(intent.resolveActivity(packageManager) != null)
103                 startActivity(intent)
104         } catch (ex : Exception){
105             Toast.makeText(this@MainActivity,
106                 "Operation failed, please try later", Toast.LENGTH_SHORT).show()
107         }
108     }
109 }
```

Explanation:

Line 35 to 57	The implicit intents here are more straightforward and are quite self-explanatory in what they do. The key parts to take note are how and what URIs to pass in for the intents, and the extras required using the putExtras() method.
Line 58 to 71	Media is played using the MediaPlayer class and note that the taps on the button toggles between starting and stopping the audio file which is inside the resources (R.raw.song)
Line 72 to 95	The torch requires that flash is available on the camera (you will be surprised, some users today actually have phones with no cameras or even flashes). If flash is available, then the device’s CameraManager is called upon to support in toggling the torch on or off. Note that the try-catch statement is there to catch any exceptions in case either the CameraManager is faulty or the phone has no camera in the first place.

activity_main.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"
    android:onClick="processOnClick"
    tools:context=".MainActivity">

    <ImageButton
        android:id="@+id/phoneBtn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:layout_marginTop="64dp"
        android:onClick="processOnClick"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.217"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.0"
        app:srcCompat="@drawable/call_icon" />

    <ImageButton
        android:id="@+id/mediaBtn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:layout_marginTop="24dp"
        android:onClick="processOnClick"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="@+id/phoneBtn"
        app:layout_constraintHorizontal_bias="1.0"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/webBtn"
        app:layout_constraintVertical_bias="0.0"
        app:srcCompat="@drawable/media" />

    <ImageButton
        android:id="@+id/webBtn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:layout_marginTop="24dp"
        android:onClick="processOnClick"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="@+id/phoneBtn"
        app:layout_constraintHorizontal_bias="1.0"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/phoneBtn"
        app:layout_constraintVertical_bias="0.0"
        app:srcCompat="@drawable/google_icon" />

    <ImageButton
        android:id="@+id/torchBtn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="24dp"
        android:layout_marginTop="24dp"
        android:layout_marginEnd="16dp"
        android:onClick="processOnClick"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.0"
        app:layout_constraintStart_toEndOf="@+id/mediaBtn"
        app:layout_constraintTop_toBottomOf="@+id/emailBtn"
        app:layout_constraintVertical_bias="0.0"
        app:srcCompat="@drawable/torch" />

    <ImageButton
        android:id="@+id/smsBtn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="24dp"
        android:layout_marginTop="64dp"
        android:layout_marginEnd="16dp"
        android:onClick="processOnClick"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.0"
        app:layout_constraintStart_toEndOf="@+id/phoneBtn"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.0"
        app:srcCompat="@drawable/sms_icon" />

    <ImageButton
        android:id="@+id/emailBtn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="24dp"
        android:layout_marginTop="24dp"
        android:layout_marginEnd="16dp"
        android:onClick="processOnClick"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.0"
        app:layout_constraintStart_toEndOf="@+id/webBtn"
        app:layout_constraintTop_toBottomOf="@+id/smsBtn"
        app:layout_constraintVertical_bias="0.0"
        app:srcCompat="@drawable/email_icon" />

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

6.5 More Internal Services: Android Notifications

An extensive example application of the use of intents into a key internal service will be the notifications. Notifications provide a way for an app to convey a message to the user when the app is either not running or is currently in the background. A messaging app, might, for example, issue a notification to let the user know that a new message has arrived from a contact. Notifications can be categorized as being either local or remote. A local notification is triggered by the app itself on the device on which it is running. Remote notifications, on the other hand, are initiated by a remote server and delivered to the device for presentation to the user.

Notifications appear in the notification drawer that is pulled down from the status bar of the screen and each notification can include actions such as a button to open the add that sent the notification. Android also allows the user to type in and submit a response to a notification from within the notification panel.

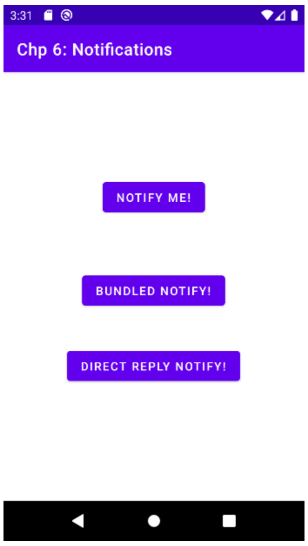
The following example will explore the different possibilities of notifications available for implementations.

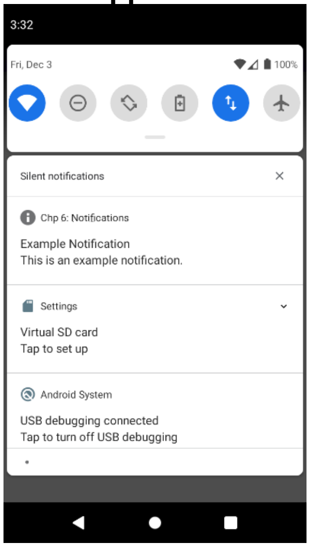
Example 4: Notifications
The app showcases three different types of notifications that can be implemented in Android studio. They are:

- One simple single notification sent from the app
- A bundle of three notifications sent at one shot
- A notification which allows a reply option (similar to what you may see for Whatsapp or Telegram notifications) which can be sent as a result to the app.

Note that for the app's notifications to work ensure that while the app is running, send it to the background and go to Settings → Apps and Notifications → App info. The show notifications should toggle switch should be on and notification dots should be allowed.

Screenshots of Notifications app

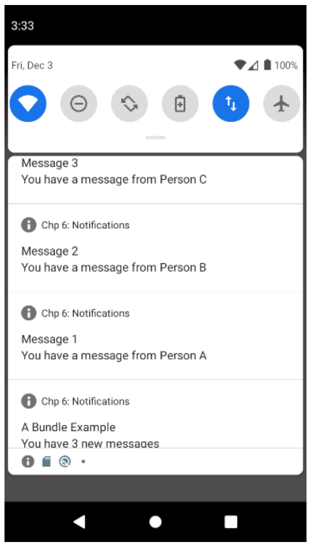




Chp 6: Notifications
Example Notification
This is an example notification.

Settings
Virtual SD card
Tap to set up

Android System
USB debugging connected
Tap to turn off USB debugging

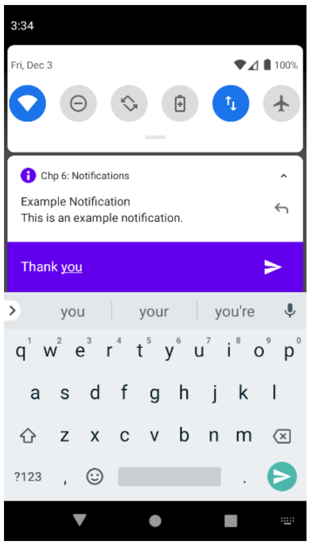


Message 3
You have a message from Person C

Chp 6: Notifications
Message 2
You have a message from Person B

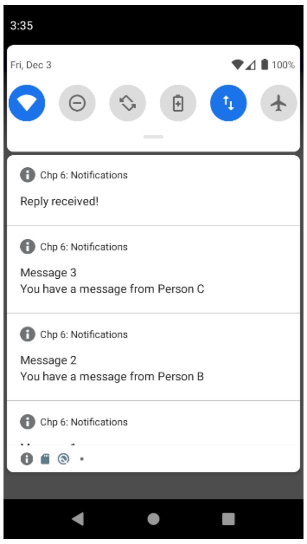
Chp 6: Notifications
Message 1
You have a message from Person A

Chp 6: Notifications
A Bundle Example
You have 3 new messages



Chp 6: Notifications
Example Notification
This is an example notification.

Thank you

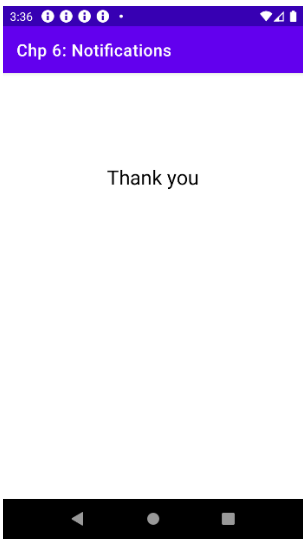


Chp 6: Notifications
Reply received!

Chp 6: Notifications
Message 3
You have a message from Person C

Chp 6: Notifications
Message 2
You have a message from Person B

Chp 6: Notifications



Thank you

AndroidManifest.xml

```
...
        <activity
            android:name=".ResultActivity"
            android:exported="false" />
         <activity
            android:name=".MainActivity"
            android:exported="true">
...
```

Recall in the above that as the app has more than one activity, both activities need to be reflected in the Android Manifest.

strings.xml

```
<resources>
    <string name="app_name">Chp 6: Notifications</string>
    <string name="notify">Notify Me!</string>
    <string name="notifyB">Bundled Notify!</string>
    <string name="notifyDirect">Direct Reply Notify!</string>
    <string name="result">Result Activity</string>
    <string name="channelName">Notification Demo News</string>
    <string name="channelDesc">Example News Channel</string>
    <string name="egNote">Example Notification</string>
    <string name="egText">This is an example notification.</string>
    <string name="bundleTitle">A Bundle Example</string>
    <string name="bundleText">You have 3 new messages</string>
    <string name="bundleTitle1">Message 1</string>
    <string name="bundleMsg1">You have a message from Person A</string>
    <string name="bundleTitle2">Message 2</string>
    <string name="bundleMsg2">You have a message from Person B</string>
    <string name="bundleTitle3">Message 3</string>
    <string name="bundleMsg3">You have a message from Person C</string>
    <string name="replyPrompt">Enter your reply here:</string>
    <string name="replyRec">Reply received!</string>
</resources>
```

MainActivity.kt

```
import android.app.*
import android.content.Context
import android.content.Intent
import android.graphics.Color
import android.graphics.drawable.Icon
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.view.View
import androidx.core.content.ContextCompat

class MainActivity : AppCompatActivity() {
    private lateinit var notificationManager: NotificationManager

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        notificationManager = getSystemService(Context.NOTIFICATION_SERVICE)
                              as NotificationManager
        createNotificationChannel("com.example.notifydemo.news",
       getString(R.string.channelName),getString(R.string.channelDesc))
    }
    private fun createNotificationChannel(id: String, name: String, description: String){
        val importance = NotificationManager.IMPORTANCE_LOW
//        val importance = NotificationManager.IMPORTANCE_HIGH
        val channel = NotificationChannel(id, name, importance)
        channel.description = description
        channel.enableLights(true)
        channel.lightColor = Color.RED
        channel.enableVibration(true)
        channel.vibrationPattern = longArrayOf(100, 200, 300, 400, 500, 400, 300, 200, 400)
        notificationManager.createNotificationChannel(channel)
    }
    fun sendNotification(view: View){
        val notificationID = 101
        val channelID = "com.example.notifydemo.news"
        val intentResult = Intent(this, ResultActivity::class.java)
        val pendingIntent = PendingIntent.getActivity(this@MainActivity, 0,
            intentResult, PendingIntent.FLAG_UPDATE_CURRENT)

        val notification = Notification.Builder(this@MainActivity, channelID)
            .setContentTitle(getString(R.string.egNote))
            .setContentText(getString(R.string.egText))
            .setSmallIcon(android.R.drawable.ic_dialog_info)
            .setContentIntent(pendingIntent)
            .setChannelId(channelID).build()
        notificationManager.notify(notificationID, notification)
    }
    fun sendBundledNotification(view: View){
        val channelID = "com.example.notifydemo.news"
        val GROUP_KEY_NOTIFY = "group_key_notify"
51          val builderSummary = Notification.Builder(this@MainActivity, channelID)
52              .setSmallIcon(android.R.drawable.ic_dialog_info)
53              .setContentTitle(getString(R.string.bundleTitle))
54              .setContentText(getString(R.string.bundleText))
55              .setGroup(GROUP_KEY_NOTIFY).setGroupSummary(true)
56          val builder1 = Notification.Builder(this@MainActivity, channelID)
57              .setSmallIcon(android.R.drawable.ic_dialog_info)
58              .setContentTitle(getString(R.string.bundleTitle1))
59              .setContentText(getString(R.string.bundleMsg1))
60              .setGroup(GROUP_KEY_NOTIFY).setGroupSummary(true)
61          val builder2 = Notification.Builder(this@MainActivity, channelID)
62              .setSmallIcon(android.R.drawable.ic_dialog_info)
63              .setContentTitle(getString(R.string.bundleTitle2))
64              .setContentText(getString(R.string.bundleMsg2))
65              .setGroup(GROUP_KEY_NOTIFY).setGroupSummary(true)
66          val builder3 = Notification.Builder(this@MainActivity, channelID)
67              .setSmallIcon(android.R.drawable.ic_dialog_info)
68              .setContentTitle(getString(R.string.bundleTitle3))
69              .setContentText(getString(R.string.bundleMsg3))
70              .setGroup(GROUP_KEY_NOTIFY).setGroupSummary(true)
71          notificationManager.notify(81, builder1.build())
72          notificationManager.notify(82, builder2.build())
73          notificationManager.notify(83, builder3.build())
74          notificationManager.notify(80, builderSummary.build())
75      }
76      fun directReplyNotification(view : View){
77          val channelID = "com.example.notifydemo.news"
78          val KEY_TEXT_REPLY = "key_text_reply"
79
80          val remoteInput = RemoteInput.Builder(KEY_TEXT_REPLY)
81              .setLabel(getString(R.string.replyPrompt)).build()
82          val intentResult = Intent(this, ResultActivity::class.java)
83          val pendingIntent = PendingIntent.getActivity(this@MainActivity, 0,
84              intentResult, PendingIntent.FLAG_UPDATE_CURRENT)
85          val icon = Icon.createWithResource(this@MainActivity,
86              android.R.drawable.ic_dialog_info)
87          val replyAction = Notification.Action.Builder(icon, "Reply", pendingIntent)
88              .addRemoteInput(remoteInput).build()
89          val notification = Notification.Builder(this@MainActivity, channelID)
90              .setColor(ContextCompat.getColor(this@MainActivity,
91                  R.color.design_default_color_primary))
92              .setContentTitle(getString(R.string.egNote))
93              .setContentText(getString(R.string.egText))
94              .setSmallIcon(android.R.drawable.ic_dialog_info)
95              .setChannelId(channelID)
96              .setActions(replyAction).build()
97          notificationManager.notify(101, notification)
98      }
99  }
```

Explanation:

Line 17 to 32	The notification manager is set up and a custom notification channel is made. The channel's properties are configured with its id, name and importance (low, high, max, min, none) and even the light when the notification occurs and the vibration pattern can be set! After it is set up, any notifications that pass through the channel with that id set up will follow the mentioned the properties configured to the channel. Note how each of the individual channel properties are setup

Line 33 to 47	This sends a single notification through the custom notification channel set up. Each notification is tagged to an ID and while there are no conventions as to what the ID can be set to, it is up to the **developer's end to iron out such IDs because an app,** would have many different IDs for many different objects. Note that notifications are set up by the Notification.Builder() and intents can be even sent through the notification (means that you will be able to activate implicit intents through the notification). Many of its properties like its icon, text, title, etc, can be set up before calling the notify() method to pump the notification down.
Line 48 to 75	This portion of the code sets up a bundled notification. This bundle can be thought of as a series of 4 distinct notifications: one summary and three other notifications. A builder is set up for each notification and notice the notification IDs in lines 71 to 74. **These IDs will determine the order at which the bundled notifications will appear** such that the one with the lower ID value will be pushed into the notification stack first, and the end result is that the one with the highest ID value will be on the top of the stack, as observed by the screenshot.
Line 76 to 98	This notification is an example of a notification **with a remote input**. The RemoteInput class allows a request for user input to be included in the PendingIntent object along with the intent. The intent will pass the reply string to ResultActivity with "Reply" as the key to retrieve it. Note that the notification action is built with the Notification.Action.Builder() and contains the RemoteInput object.

activity_main.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/notifyDirect"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="directReplyNotification"
        android:text="@string/notifyDirect"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.497"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/notifyB"
        app:layout_constraintVertical_bias="0.236" />

    <Button
        android:id="@+id/notifyB"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="50dp"
        android:onClick="sendBundledNotification"
        android:text="@string/notifyB"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.497"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/notify"
        app:layout_constraintVertical_bias="0.057" />

    <Button
        android:id="@+id/notify"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="sendNotification"
        android:text="@string/notify"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.498"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.27" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

ResultActivity.kt

```
import android.app.Notification
import android.app.NotificationChannel
import android.app.NotificationManager
import android.app.RemoteInput
import android.content.Context
import android.graphics.Color
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.TextView

class ResultActivity : AppCompatActivity() {
    private lateinit var notificationManager: NotificationManager
    private lateinit var textView : TextView

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_result)
        textView = findViewById(R.id.textView)
        notificationManager = getSystemService(Context.NOTIFICATION_SERVICE)
                       as NotificationManager
        createNotificationChannel("com.example.notifydemo.news",
       getString(R.string.channelName), getString(R.string.channelDesc))
        handleIntent()
    }
    private fun handleIntent(){
        val channelID = "com.example.notifydemo.news"
        val KEY_TEXT_REPLY = "key_text_reply"
        val intent = this.intent
        val remoteInput = RemoteInput.getResultsFromIntent(intent)
        if (remoteInput != null){
            val inputString = remoteInput.getCharSequence(KEY_TEXT_REPLY).toString()
            textView.text = inputString
        }
        // Update the notification after receiving input
        val repliedNotification = Notification.Builder(this@ResultActivity, channelID)
            .setContentText(getString(R.string.replyRec))
            .setSmallIcon(android.R.drawable.ic_dialog_info).build()

        notificationManager.notify(101, repliedNotification)
    }
41      private fun createNotificationChannel(id: String, name: String, description: String) {
42          val importance = NotificationManager.IMPORTANCE_LOW
43  //        val importance = NotificationManager.IMPORTANCE_HIGH
44          val channel = NotificationChannel(id, name, importance)
45          channel.description = description
46          channel.enableLights(true)
47          channel.lightColor = Color.RED
48          channel.enableVibration(true)
49          channel.vibrationPattern = longArrayOf(100, 200, 300, 400, 500, 400, 300, 200, 400)
50          notificationManager.createNotificationChannel(channel)
51      }
52  }
```

Explanation:

Line 19 to 22 Line 41 to 51	Note that for ResultActivity, the notification channel is set to the same channel as MainActivity with the exact same settings for consistency.
Line 23 to 40	When the intent comes in (the reply using the RemoteInput), two things will be done. The first is that if there is indeed a reply, that same reply will be reflected in the TextView when ResultActivity opens. The second is that a reply notification will be given to indicate to the user that the reply has indeed been received by the app.

activity_result.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=".ResultActivity">

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/result"
        android:textColor="@color/black"
        android:textSize="24sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.498"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.227" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

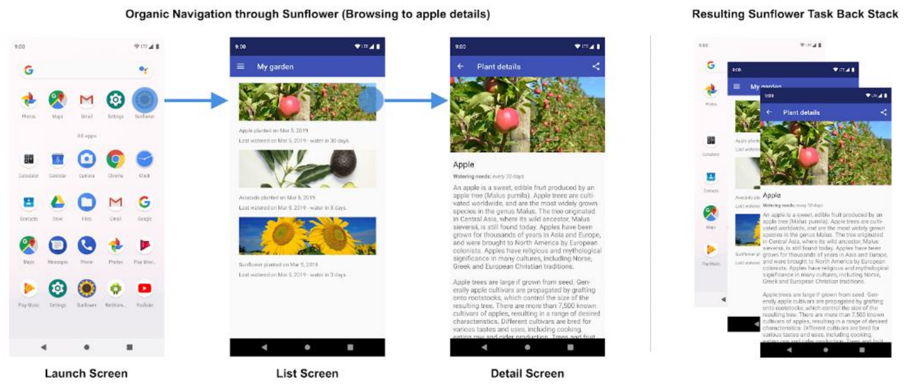
Note that this example only features text-based notifications. In fact, there are more kinds of notifications you can do such as implementing a progress bar (for downloads) or even adding action buttons to perform a task (implicit intents). You may find out more in the link below:
https://developer.android.com/training/notify-user/build-notification

6.6 Android Jetpack Navigation and the ViewModel

Now that we have the required knowledge to build intents and pass information from one activity to another, it will now be appropriate to go through how navigation works in the app. Navigation refers to the interactions that allow users to navigate across, into, and back out from the different pieces of content within your app. **Android Jetpack's Navigation component** helps you implement navigation, from simple button clicks to more complex patterns, such as app bars and the navigation drawer.

To ensure a consistent and predictable user experience, an established set of principles will need to be followed:

- Every app must have a **fixed start destination**, which is the first screen the user sees when they launch the app. Note that login screens, onboarding or one-time setup screens are not considered start destinations as users only see these screens in very specific, usually one-time-only cases.
- Navigation is represented as a **stack of destinations**, where the start destination is at the bottom of the stack and the very top of the stack will be the current screen. Hence, you can think of the stack as the history of where you have navigated in the app. Any newer destinations or operations that affect the destinations already in the stack will be pushed on top of the stack or popping (removing) the top-most destination off the stack


Organic Navigation through Sunflower (Browsing to apple details)
Resulting Sunflower Task Back Stack
My garden
Plant details
Apple
Launch Screen
List Screen
Detail Screen

- Within the app, the Up button (on the action bar, refer to Example 1 on how to implement it) and the Back button of your phone can both be used to navigate backward.
- In addition to the above point, the Up button will never exit the app but the phone's Back button will.
- Deep links (calling implicit intents) or manual navigation to a specific destination, allows for the user to click on the Up or Back button to go backwards in the navigation stack. For implicit intents (deep links), any existing navigation stack will be removed and replaced with the deep-linked stack. You may read more here: https://developer.android.com/guide/navigation/navigation-principles

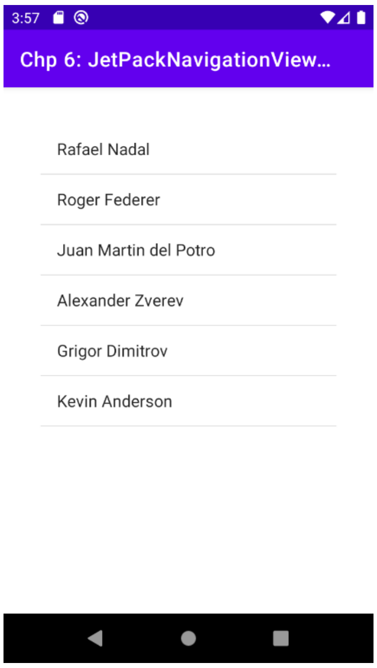
You will notice that the above behaviours are present in most, if not all commercial apps today, and the Android Jetpack's Navigation component will automatically implement the above guidelines, unless you opt to change the behaviours programmatically. The following example will run through how navigation works, tied with the **ViewModel**, which acts as data management medium to enforce data persistence even through navigation.

Example 5: JetPackNavigationViewModel

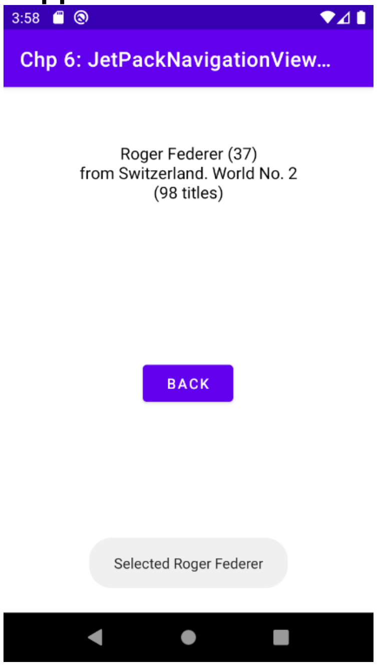
This app will allow the user to select Tennis players from the list and a new fragment is output the details on the player's statistics.

Note that this app was created starting with the Basic Activity template.

Screenshots of Notifications app

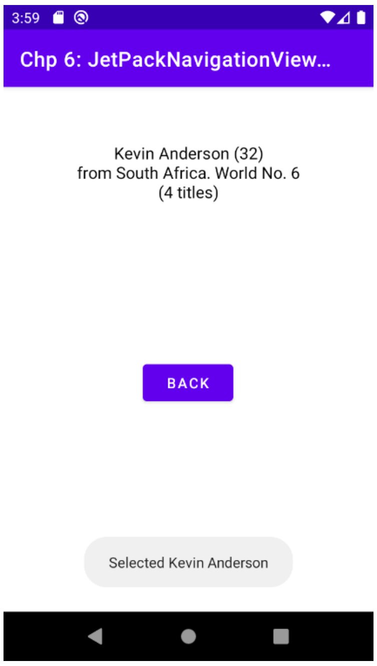


Rafael Nadal
Roger Federer
Juan Martin del Potro
Alexander Zverev
Grigor Dimitrov
Kevin Anderson



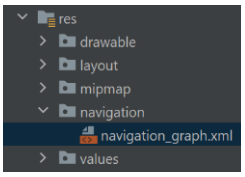
Roger Federer (37)
from Switzerland. World No. 2
(98 titles)

Selected Roger Federer



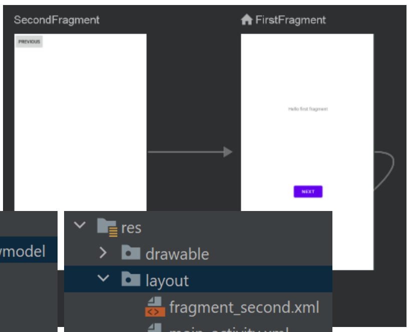
Kevin Anderson (32)
from South Africa. World No. 6
(4 titles)

Selected Kevin Anderson



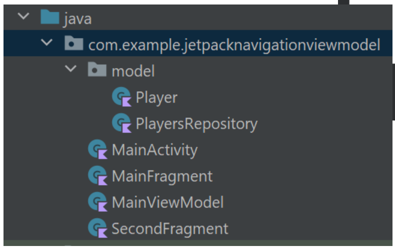
```
res
  drawable
  layout
  mipmap
  navigation
    navigation_graph.xml
  values
```

The key idea here in this app is to setup the navigation between fragments (note that the app works such that MainFragment and SecondFragment will run on the same Activity window). To set up the navigation, we will need to go into the navigation folder in res and open the navigation_graph.xml. (if it is open from the Basic Activity template, it will be called nav_graph.xml)

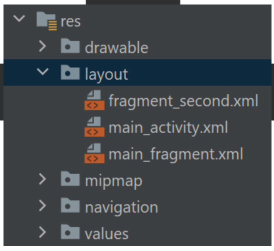


When navigation_graph.xml is open in Design mode, you will find that the default fragments are linked to each other, represented like a directed graph.

For this app, you will want to first modify it such that you have the following files in your project structure. Essentially what we want to do is to "navigate" from MainFragment to SecondFragment, while passing data stored in the MainViewModel.



```
java
  com.example.jetpacknavigationviewmodel
    model
      Player
      PlayersRepository
    MainActivity
    MainFragment
    MainViewModel
    SecondFragment
```



```
res
  drawable
  layout
    fragment_second.xml
    main_activity.xml
    main_fragment.xml
  mipmap
  navigation
  values
```

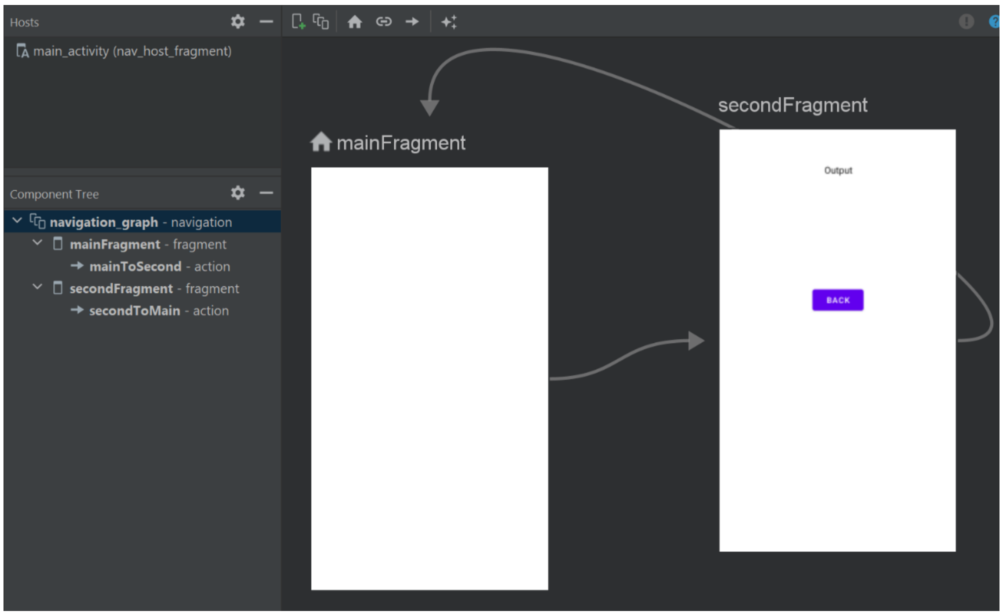
The MainViewModel extends the ***ViewModel*** class, which is designed to store and manage UI-related data in a lifecycle conscious way, enforcing **data persistence** even if the Android system destroys or re-creates a UI controller. Ideally, we will want the UI controller to handle only the UI aspects and not be also responsible for data related work, which hence makes sense that we have a helper class for the UI controller which is responsible for preparing data for the UI. Note that ViewModel objects are automatically retained during configuration changes so that data they hold is immediately available to the next activity or fragment instance.

To implement the ViewModel, you will need to modify the build.gradle (app) file an implement the following inside the **dependencies** block:

```
implementation 'androidx.lifecycle:lifecycle-livedata-ktx:2.4.0'
implementation 'androidx.lifecycle:lifecycle-viewmodel-ktx:2.4.0'
```

For the exact code for the different classes and their associated layouts, you may go ahead to take a look at the code. For now, we will focus on the navigation graph. The navigation graph is modified to now show the following:

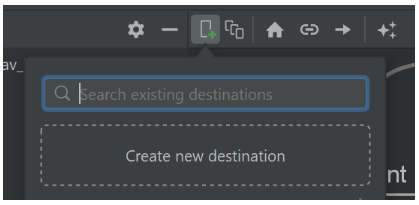
navigation_graph.xml



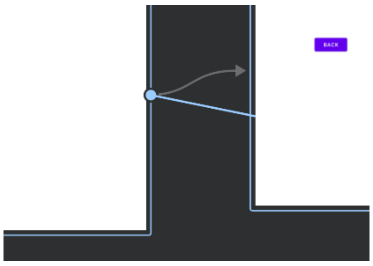
```
<?xml version="1.0" encoding="utf-8"?>
<navigation xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/navigation_graph"
    app:startDestination="@id/mainFragment">

    <fragment
        android:id="@+id/mainFragment"
        android:name="com.example.jetpacknavigationviewmodel.MainFragment"
        android:label="main_fragment"
        tools:layout="@layout/main_fragment" >
        <action
            android:id="@+id/mainToSecond"
            app:destination="@id/secondFragment" />
    </fragment>
    <fragment
        android:id="@+id/secondFragment"
        android:name="com.example.jetpacknavigationviewmodel.SecondFragment"
        android:label="fragment_second"
        tools:layout="@layout/fragment_second" >
        <argument
            android:name="message"
            app:argType="string"
            android:defaultValue="No Message" />
        <action
            android:id="@+id/secondToMain"
            app:destination="@id/mainFragment" />
    </fragment>
</navigation>
```

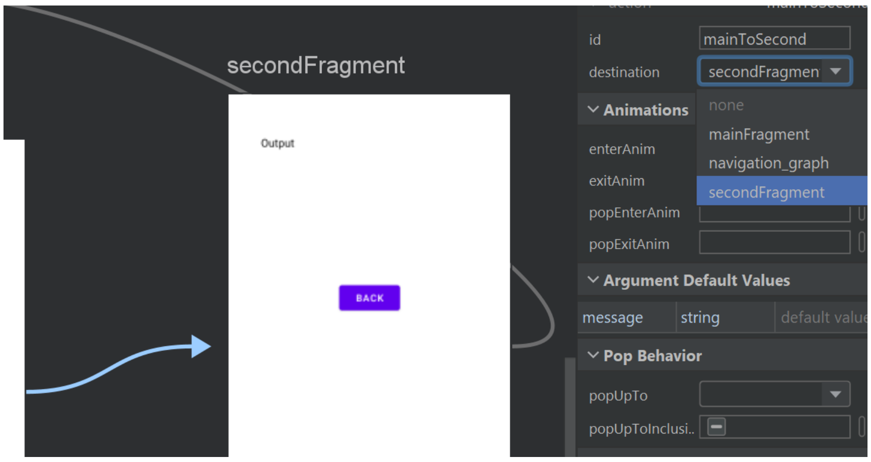
In Design mode, you will notice that the main_activity layout is labelled as the "nav_host_fragment", which is the host of all the fragments the navigation graph is for.

To add a fragment to be part of the navigation graph, click on the icon to add a fragment to be part of the navigation. If the fragment has yet to be created, you can create a new fragment with the options given.



To create the navigation flow, you can click and drag the circle on the right edge of the fragment towards the edge of the child fragment in the navigation flow. After the arrow is produced, you can click on the arrow to give the **action** (the arrow) an id and the destination (which you would have already defined through connecting the fragments).



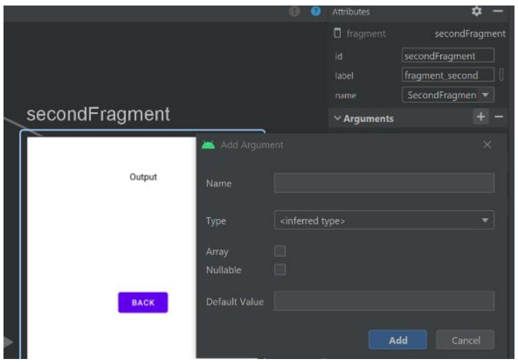


id: mainToSecond
destination: secondFragment
enterAnim
exitAnim
popEnterAnim
popExitAnim
message string default value
popUpTo
popUpToInclusive

For this app, the action from mainFragment to secondFragment is id as mainToSecond and the action from the secondFragment to mainFragment is id as secondToMain. Note that you will be able to add transition animations between fragments through this as well.

To tell the compiler that secondFragment is to receive data from the mainFragment, click on the secondFragment itself and you will see the fragment's attributes being displayed. Click on the + icon on the "Arguments" section and a pop-up window will appear. In this app, the argument added was:

- Name: message
- Type: String
- Default Value: "No message"



fragment secondFragment
id: secondFragment
label: fragment_second
name: SecondFragmen
Name:
Type: <inferred type>
Array
Nullable
Default Value:

Now the navigation graph is fully ready. Note that more complex navigations are possible but that will be left for you to find out more about. You will notice a "Deep Links" attribute in each fragment. Essentially it sends an implicit intent to perform tasks like opening a website or sending a text message.

You may find out more here: https://developer.android.com/training/app-links/deep-linking
Read ahead for the code implemented. strings.xml will not be shown as the code is trivial

Player.kt

```
data class Player (val name: String, val age: Int, val country: String, val rank: Int,
                   val titles: Int)
```

PlayersRepository.kt

```
1   class PlayersRepository {
2       private var players = arrayOf("Rafael Nadal", "Roger Federer", "Juan Martin del Potro",
3           "Alexander Zverev", "Grigor Dimitrov", "Kevin Anderson")
4       private var playerDetails: HashMap<String, Player> = HashMap()
5
6       init{
7           playerDetails["Rafael Nadal"] = Player("Rafael Nadal", 32, "Spain", 1, 80)
8           playerDetails["Roger Federer"] = Player("Roger Federer", 37,
9               "Switzerland", 2, 98)
10          playerDetails["Juan Martin del Potro"] = Player("Juan Martin del Potro", 29,
11              "Argentina", 3, 22)
12          playerDetails["Alexander Zverev"] = Player("Alexander Zverev", 21,
13              "Germany", 4, 9)
14          playerDetails["Grigor Dimitrov"] = Player("Grigor Dimitrov", 27,
15              "Bulgaria", 5, 8)
16          playerDetails["Kevin Anderson"] = Player("Kevin Anderson", 32,
17              "South Africa", 6, 4)
18      }
19
20      fun getPlayers(): Array<String> { return players }
21      fun getPlayerDetails(name: String): Player? { return playerDetails[name]}
22  }
```

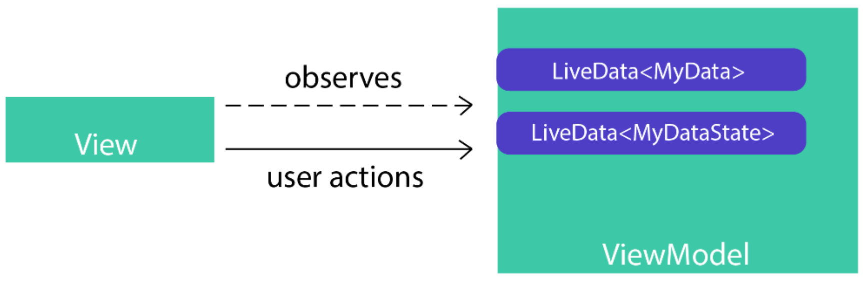
Explanation:

Line 2 to 18	The above classes defines the actual model for the app, where in the PlayerRepository, a hashmap is defined such that you define **key-value pairs**, where the keys are the player names and the value will be the individual Player object. Accessors are also set up to retrieve the data. Notice how the constructor for the PlayerRepository class is coded using the init keyword, which is very similar to how Python constructors are coded. Note that Java has the same HashMap class which you may want to use for your Java programs in the future.

MainViewModel.kt

```
import androidx.lifecycle.MutableLiveData
import androidx.lifecycle.ViewModel
import com.example.jetpacknavigationviewmodel.model.Player
import com.example.jetpacknavigationviewmodel.model.PlayersRepository

class MainViewModel : ViewModel() {
    private val selectedPlayer = MutableLiveData<String>()
    private val repository = PlayersRepository()

    fun selectPlayer(playerName : String){selectedPlayer.value = playerName }
    fun getSelectedPlayer() : MutableLiveData<String> {return selectedPlayer}
    fun getPlayerList() : MutableList<String> {
        return (repository.getPlayers()).toMutableList()}
    fun getPlayerDetails(name : String) : Player {return repository.getPlayerDetails(name)!!
}
```

Explanation:

Line 7 Line 11 to 13	Within the ViewModel, you will notice the that data from the model is now put into MutableLiveData objects. In general, LiveData is an observable data holder class that is used to observe the changes of a ViewModel and update those changes. LiveData is lifecycle-aware, which means that whenever data is updated or changed, the changes are only applied to the specific app components that are in an active state.  observes View user actions LiveData<MyData> LiveData<MyDataState> ViewModel Note that MutableLiveData is a subclass of LiveData which data can be changed. In this context, changes are made to the selection of the player in the MainFragment, this will cause actions to be taken such that the SecondFragment knows and receives the required update. This will be elaborated on later.

SecondFragment.kt

```
import android.net.Uri
import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.Button
import android.widget.TextView
import androidx.fragment.app.Fragment
import androidx.navigation.Navigation

class SecondFragment : Fragment() {
    private lateinit var argText : TextView
    private lateinit var button : Button
    interface OnFragmentInteractionListener{ fun onFragmentInteraction(uri : Uri) }

    override fun onStart() {
        super.onStart()
        argText = requireView().findViewById(R.id.argText)
        button = requireView().findViewById(R.id.button)
        button.setOnClickListener{ _ ->
            Navigation.findNavController(requireView()).navigate(R.id.secondToMain)
        }
        val args = SecondFragmentArgs.fromBundle(requireArguments())
        val msg = args.message
        argText.text = msg
    }
    override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?): View? {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.fragment_second, container, false)
    }
}
```

Explanation:

Line 14 Line 20 to 25	Note that the interface OnFragmentInteractionListener is implemented and is meant to handle the communication between activity and fragment. This is added in case I will require the MainActivity to communicate with SecondFragment, especially since the text in the TextView of SecondFragment changes while navigation between fragments (within the same activity) is taking place. The navigation data from MainActivity will contain the output string required for display in the TextView and hence, retrieval of the data from the navigation action will be required and the text is displayed. For the button, when the button is clicked on, the navigation action secondToMain will be located in the navigation graph to know the destination fragment to go to.

fragment_second.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:id="@+id/constraintLayout2"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="48dp"
    tools:context=".SecondFragment">

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="24dp"
        android:text="@string/back"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.498"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/argText"
        app:layout_constraintVertical_bias="0.0" />

    <TextView
        android:id="@+id/argText"
        android:layout_width="0dp"
        android:layout_height="187dp"
        android:textAlignment="center"
        android:textColor="@color/black"
        android:textSize="16sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.489"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.024"
        tools:text="@string/output" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

MainFragment.kt

```
1    import androidx.lifecycle.ViewModelProvider
2    import android.os.Bundle
3    import androidx.fragment.app.Fragment
4    import android.view.LayoutInflater
```

```
5   import android.view.View
6   import android.view.ViewGroup
7   import androidx.navigation.Navigation
8   import android.widget.*
9
10  class MainFragment : Fragment() {
11      private lateinit var viewModel: MainViewModel
12      private lateinit var listView: ListView
13
14      override fun onStart() {
15          super.onStart()
16          viewModel = ViewModelProvider(this).get(MainViewModel::class.java)
17          listView.adapter = ArrayAdapter(this.requireContext(),
18                        android.R.layout.simple_list_item_1, viewModel.getPlayerList())
19          listView.setOnItemClickListener { _, itemView, _, _ ->
20              val textView = itemView as TextView
21              Toast.makeText(
22                  this.requireContext(), "Selected " + textView.text.toString(),
23                  Toast.LENGTH_SHORT
24              ).show()
25              viewModel.selectPlayer(textView.text.toString())
26              viewModel.getSelectedPlayer().observe(viewLifecycleOwner) { item ->
27                  val player = viewModel.getPlayerDetails(item)
28                  val str = player.name + " (" + player.age + ")\nfrom " + player.country +
29                          ". World No. " + player.rank + "\n(" + player.titles + " titles)"
30                  val action = MainFragmentDirections.mainToSecond()
31                  action.message = str
32                  Navigation.findNavController(requireView()).navigate(action)
33              }
34          }
35      }
36      override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,
37          savedInstanceState: Bundle?): View {
38          val view: View = inflater.inflate(R.layout.main_fragment, container, false )
39          listView = view.findViewById<View>(R.id.players_lv) as ListView
40          return view
41      }
42  }
```

Explanation:

Line 11 to 12 Line 16 to 34	The ViewModel is retrieved and the ListView is set up with the player names from the ViewModel, with a listener to tell the user what was selected on the ListView. When the new player is selected through the selectPlayer() method, the MutableLiveData change is observed, and the code within the observe() method is run such that the details of the player will be sent to SecondFragment as an intent for the SecondFragment to perform the update. The navigation to SecondFragment is located through the navigation action's name (mainToSecond), with the message string attached to the navigation and will be pushed down to the SecondFragment.

fragment_main.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:id="@+id/constraintLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="36dp"
    tools:context=".MainFragment">

    <ListView
        android:id="@+id/players_lv"
        android:layout_width="0dp"
        android:layout_height="0dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent">

    </ListView>

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

MainActivity.kt

```
import android.net.Uri
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle

class MainActivity : AppCompatActivity(), SecondFragment.OnFragmentInteractionListener {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.main_activity)
    }
    override fun onFragmentInteraction(uri : Uri){ }
}
```

Explanation:
Note that to communicate with the SecondFragment, you will need to implement the onFragmentInteractionListener and override the onFragmentInteraction method. Note that as I do not need to perform any tasks during the communication of MainActivity and SecondFragment, I can leave the internal code blank.

main_activity.xml

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout 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:id="@+id/container"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity" >

    <androidx.fragment.app.FragmentContainerView
        android:id="@+id/nav_host_fragment"
        android:name="androidx.navigation.fragment.NavHostFragment"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:defaultNavHost="true"
        app:navGraph="@navigation/navigation_graph" />
</FrameLayout>
```

Now you have the means to perform navigation from one fragment to another within the same activity. Note that you can couple this with going back and forth between activities (defining the parent activities in the Android Manifest and programmatically setting the intents and activities to go to) and you will be able to develop a complex app filled with multiple pages and screens, navigating from one page to another. You will get more practice on implementing navigations (and some intents) in Lab 6.

[Reference]

[1] Android Intents: https://developer.android.com/guide/components/intents-filters
[2] Android Intent Filters: https://www.tutorialspoint.com/android/android_intents_filters.htm
[3] Common Intents: https://developer.android.com/guide/components/intents-common
[3] Android ViewModel class: https://developer.android.com/topic/libraries/architecture/viewmodel?gclid=Cj0KCQiAnaeNBhCUARIsABEee8V1wyWg0MbGxiKY_2gw53NoR8SlWpDOj7Z_8k4Ub61U0_Un43bzHccaAv0OEALw_wcB&gclsrc=aw.ds
[4] Navigation in Android: https://developer.android.com/guide/navigation?gclid=Cj0KCQiAnaeNBhCUARIsABEee8UhygwNSzlYJBVWvMLQAT4_u7FoX_-e8pjVJbk3ldu6P4bHsJ298MsaAuQhEALw_wcB&gclsrc=aw.ds
[5] Android Notifications: https://developer.android.com/guide/topics/ui/notifiers/notifications
[6] Andoid Jetpack Navigation: https://developer.android.com/guide/navigation?gclid=Cj0KCQiA47GNBhDrARIsAKfZ2rB71KgPpBhboiz5KCXgN8RjNb-0YjXXq-EZskQsmf6LR9Efuw-pMXMaAvmpEALw_wcB&gclsrc=aw.ds
[7] Navigation Principles: https://developer.android.com/guide/navigation/navigation-principles
[8] How ViewModels Work on Android: https://betterprogramming.pub/everything-to-understand-about-viewmodel-400e8e637a58
[9] LiveData and MutableLiveData in Android: https://www.innominds.com/blog/introduction-to-livedata-in-android
[10] MutableLiveData Class: https://developer.android.com/reference/androidx/lifecycle/MutableLiveData