CMPS 312 Mobile Application Development
Lab 6 – Navigation Component with Menus

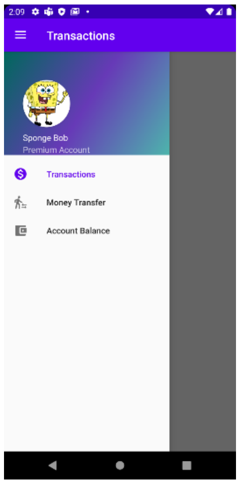
Objective

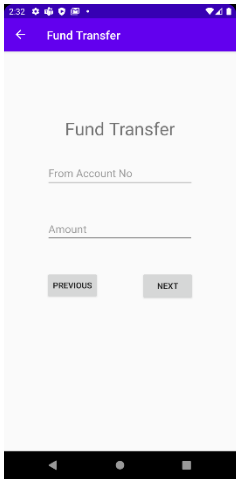
In this Lab, you will build **a Banking App** to practice the following skills:

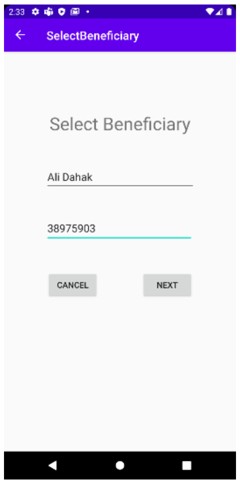
1. Create fragments.
2. Implement navigation using the Navigation Component, which is a set of libraries, a plugin, and tooling that simplifies app navigation.
3. Pass data between fragments using SafeArgs and Parcelable objects.
4. Implement Navigation Drawers for app navigation.

Overview

In the Lab, you will develop a Banking app to practice app navigation.

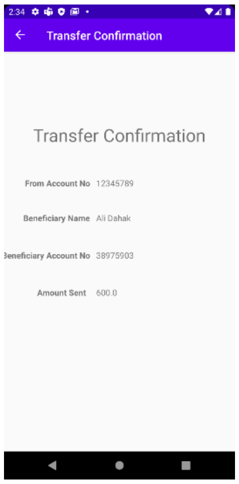




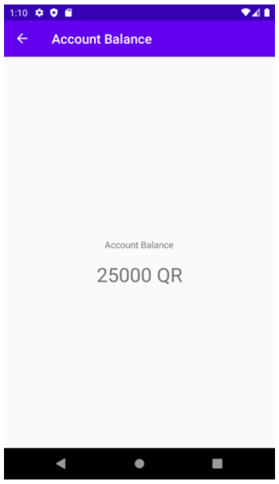


Ali Dahak

38975903



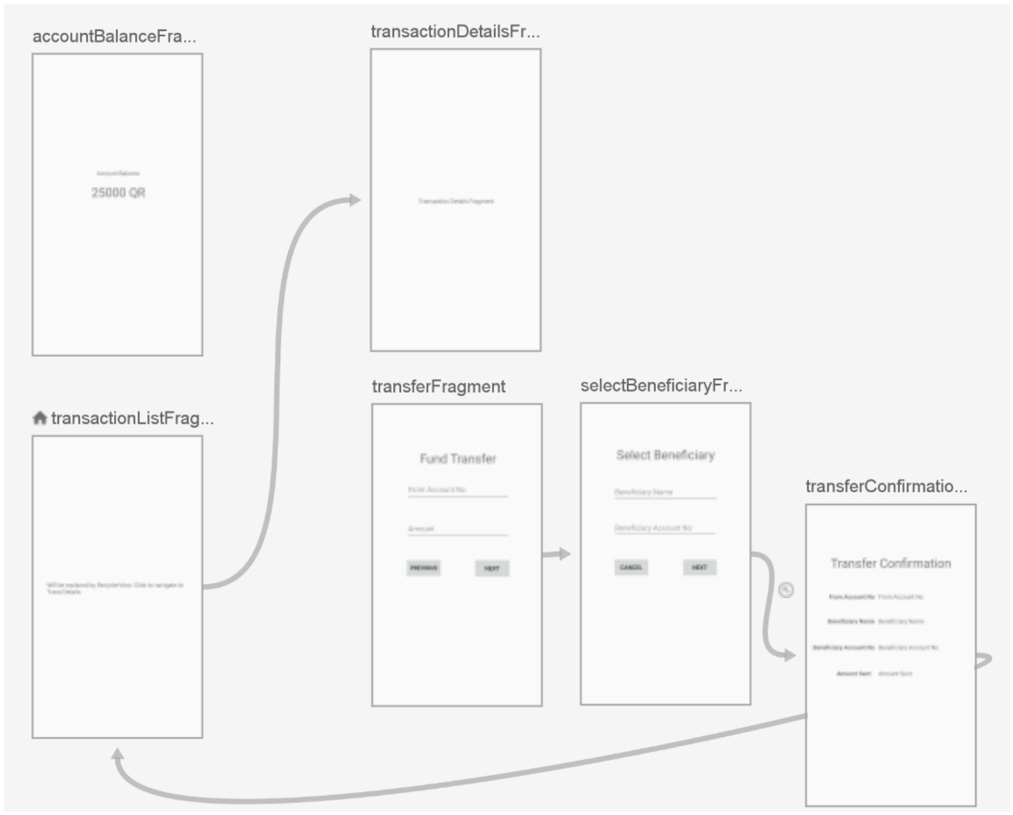
From Account No	12345789
Beneficiary Name	Ali Dahak
Beneficiary Account No	38975903
Amount Sent	600.0



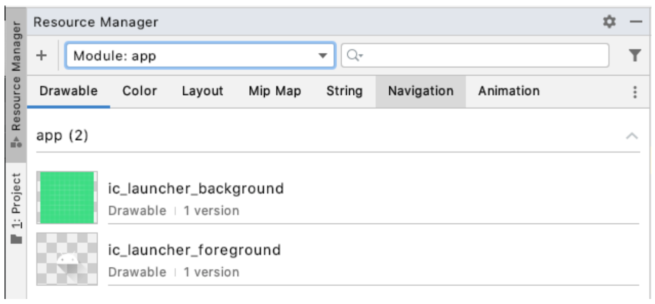
25000 QR

Part 1-Creating the Fragments and Configuring the Navigation Graph

1. Sync the Lab GitHub repo and copy the **Lab6-Navigation** folder into your repository.
2. Open the project ***Bank App*** in Android Studio.
   The project has the following folders and files:
   - **layout**: the activity layout has been provided. Explore it and note the ids of the views you will access programmatically.
   - **drawable:** images needed for this app.
3. Create the app navigation graph, named **nav_graph**, having the following destinations and actions (i.e., possible paths between destinations).



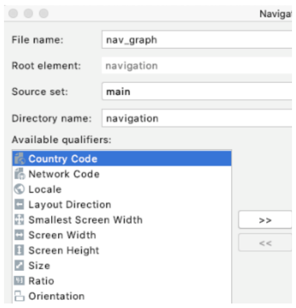
3.1. On the Project window, right-click and select **New > Android Resource File or** use the Resource Manager. The **New Resource File** dialog appears. Select Navigation as the Resource type.



app (2)

ic_launcher_background
Drawable | 1 version

ic_launcher_foreground
Drawable | 1 version

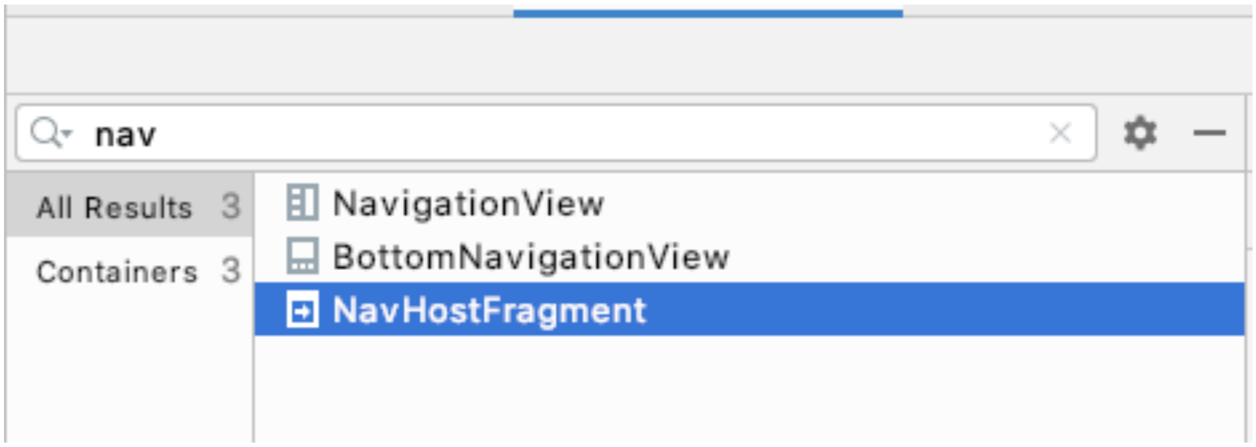


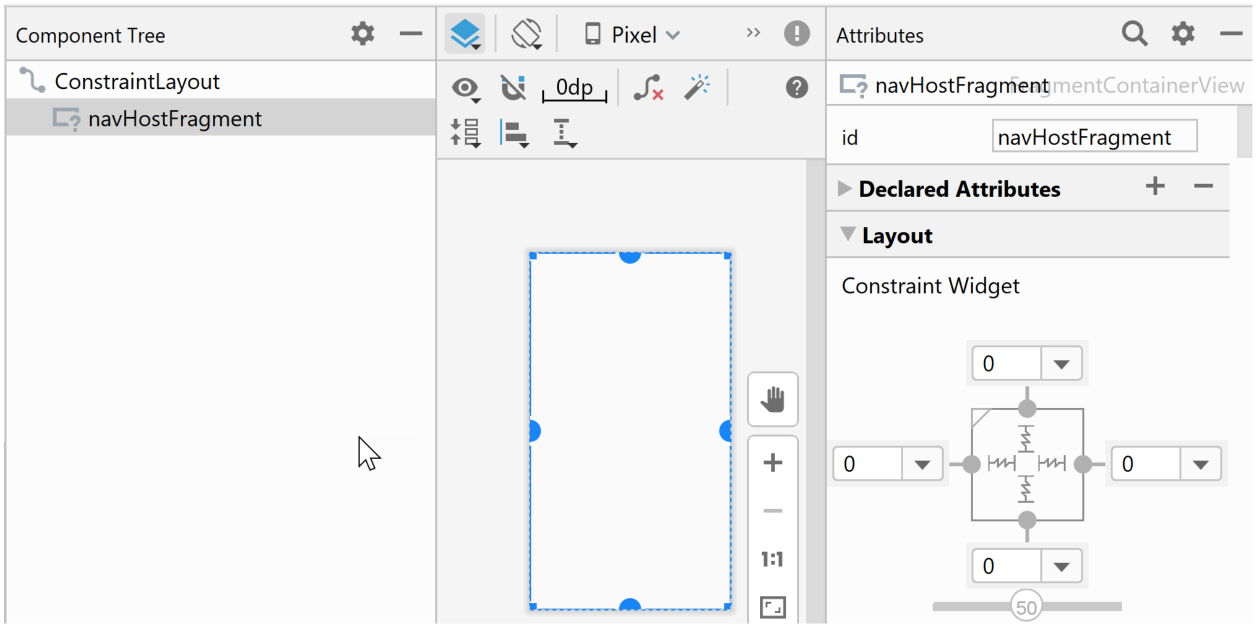
File name: nav_graph
Root element: navigation
Source set: main
Directory name: navigation
Available qualifiers:

3.2. Open the Navigation Editor and create the following 6 destinations/fragments . Note that destinations have the same name as fragments, but the first letter is lower case.

Destinations/Fragments	
`TransactionListFragment`	`toTransactionDetails` `toAccountBalance`
`TransactionDetailsFragment`	
`AccountBalanceFragment`	
`TransferFragment`	`toSelectBeneficiary`
`SelectBeneficiaryFragment`	`toTransferConfirmation`
`TransferTransferConfirmationFragment`	`toHome (this should return to the TransactionListFragment)`

4. Copy the fragment layouts from the Lab repo *fragments* subfolder into the *layout* folder of your project (this will save time designing the fragment layouts).
5. Set the `TransactionListFragment` as the start destination. When the user clicks 'Transaction Details' navigate to `toTransactionDetails.` If 'Fund Transfer' is clicked navigate to `TransferFragment.`
6. Open the **activity_main** and add a **NavHostFragment** and connect it with the **nav_graph** you created. Its id should be navHostFragment. Modify its layout constraint to fill the whole screen.





id navHostFragment

7. Modify the fragment classes to implement the navigation as described in Table 1. For example, in the `TransferFragment` implement navigate to `SelectBeneficiaryFragment`. Remove all the code generated by Android studio and implement the navigation as shown below.

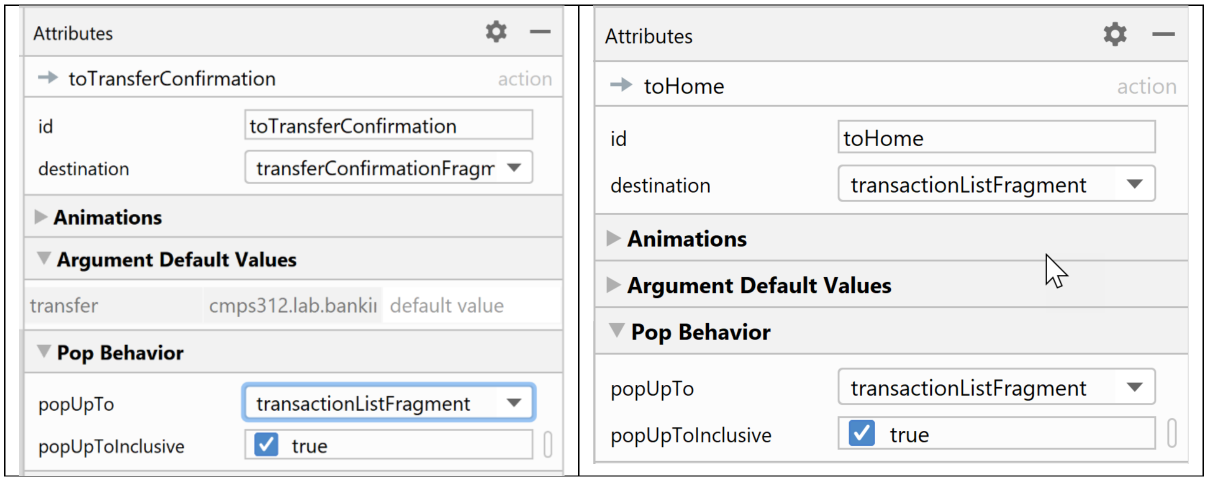
```
class TransferFragment : Fragment(R.layout.fragment_transfer) {
    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        view.nextBtn.setOnClickListener {
            findNavController().navigate(R.id.toSelectBeneficiary)
        }
    }
}
```

8. Call setupActionBarWithNavController in the MainActivity onCreate to show the **Navigate Up** button and the **label** of the current fragment on the Action Bar

   navController = *findNavController*(R.id.*navHostFragment*)
   setupActionBarWithNavController(**this**, **navController**)

   Handle *Navigate Up* event
   override fun onSupportNavigateUp() = **navController**.navigateUp()

9. Set **popUpTo and popUpToInclusive** for `toTransferConfirmation and toHome` to clear the back stack and ensure the user is NOT returned to the transfer screens after conformation.


Attributes
toTransferConfirmation action
id: toTransferConfirmation
destination: transferConfirmationFragm
Animations
Argument Default Values
transfer | cmps312.lab.bankii | default value
Pop Behavior
popUpTo: transactionListFragment
popUpToInclusive: ✓ true

Attributes
toHome action
id: toHome
destination: transactionListFragment
Animations
Argument Default Values
Pop Behavior
popUpTo: transactionListFragment
popUpToInclusive: ✓ true

10. Run and test the app as you make progress [This is important to test before moving to Part 2]

Part 2 – Transferring Data Between Fragments using SafeArgs

To pass data between destinations, you need to first add an argument by adding it to the **destination that receives it**. In the banking app the two fragments that will be receiving arguments are [`SelectBeneficiaryFragment` **and** `TransferConfirmationFragment`].

1. Inside the **model** package create **Transfer** class that implements **Parcelable** and annotate the class with **@parcelize**.

2. Open the navigation graph and add a transfer object as an argument for both `SelectBeneficiaryFragment` and `TransferConfirmationFragment` destinations. Name the argument "**transfer**"

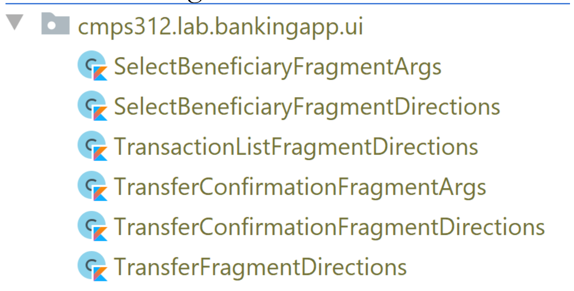
3. Once you add the arguments you will need to configure the SafeArgs plugin. Add the following dependency to the project build.gradle file

```
def nav_version = "2.3.0"
classpath "androidx.navigation:navigation-safe-args-gradle-plugin:$nav_version"
```

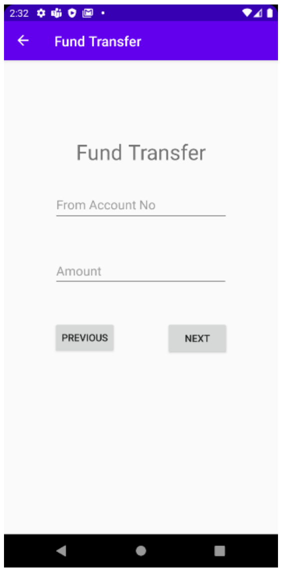
Then add this plug int to **your app module** build.gradle (app) file:

```
apply plugin: "androidx.navigation.safeargs.kotlin"
```

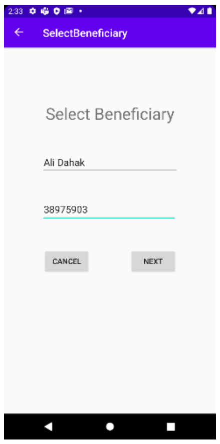
4. **Sync then Build your project**. You should see a new folder called "**java(generated)**" that looks like the image shown below. You will be using these classes for navigation and for sending and receiving data between fragments.


cmps312.lab.bankingapp.ui
- SelectBeneficiaryFragmentArgs
- SelectBeneficiaryFragmentDirections
- TransactionListFragmentDirections
- TransferConfirmationFragmentArgs
- TransferConfirmationFragmentDirections
- TransferFragmentDirections

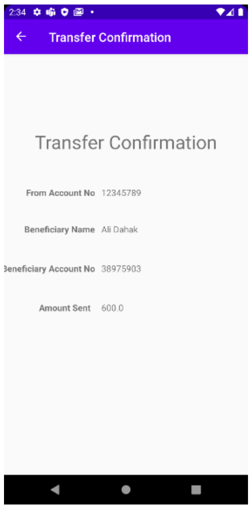
5. In the **TransferFragment** class when the user clicks Next button create a transfer object and set the fromAccount and the amount. Send the transfer object when navigating to `SelectBeneficiaryFragment`.



6. In **SelectBeneficiaryFragment** get the received *transfer* object using the safeArgs generated class.
When the user clicks Next button then set the receiverName and receiverAccount. Then pass that object when navigating to TransferConfirmationFragment.



7. Finally, in the TransferConfirmationFragment get the received *transfer* object and display it.



Transfer Confirmation

Transfer Confirmation

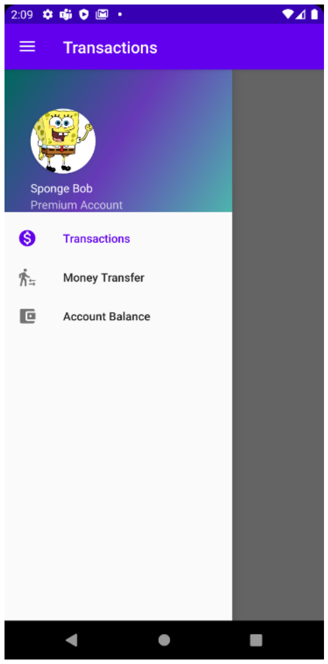
From Account No	12345789
Beneficiary Name	Ali Dahak
Beneficiary Account No	38975903
Amount Sent	600.0

8. When user presses on **Up or back** buttons the system should auto navigate the user to the home destination.
9. Test the app as you make progress.

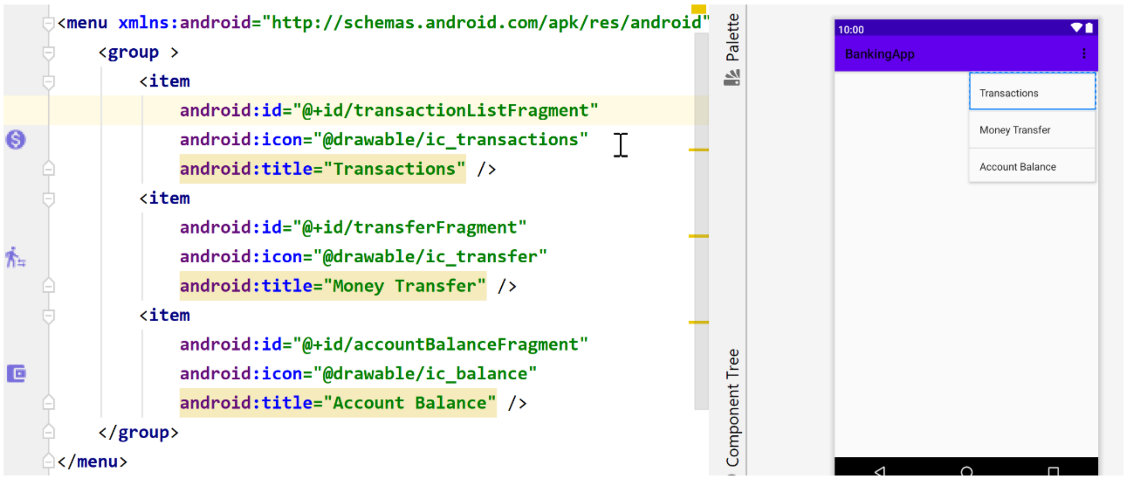
Part 3 – Creating the app navigation using navigation drawer

For the current application we create a navigation menu similar to the one shown in the image below for the top-level level destinations:

- TransactionListFragment
- TransferFragment
- AccountBalanceFragment



1. First create a menu named **menu_drawer_nav** which has three menu items. Make sure the menu items IDs are identical to the fragment Ids in the navigation graph.

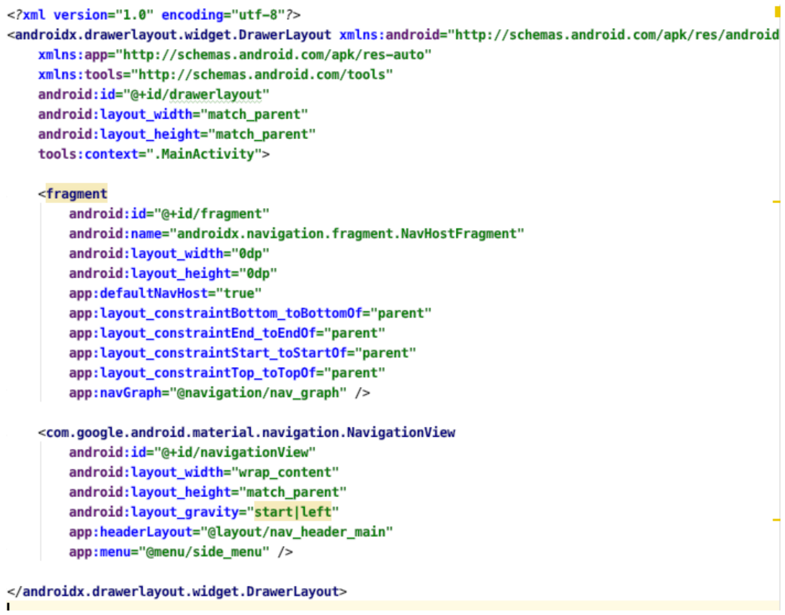


```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    <group >
        <item
            android:id="@+id/transactionListFragment"
            android:icon="@drawable/ic_transactions"
            android:title="Transactions" />
        <item
            android:id="@+id/transferFragment"
            android:icon="@drawable/ic_transfer"
            android:title="Money Transfer" />
        <item
            android:id="@+id/accountBalanceFragment"
            android:icon="@drawable/ic_balance"
            android:title="Account Balance" />
    </group>
</menu>
```

2. Open the main **activity layout** and Wrap the root element in a **DrawerLayout**.
3. Then add a NavigationView and configure it as follows:

android:layout_gravity="start|left" and
android:height="match_parent" and width=”wrap-content”
app:headerLayout="@layout/nav_header_main"
app:menu="@menu/side_menu"

The final layout xml should look like this:



```
<?xml version="1.0" encoding="utf-8"?>
<androidx.drawerlayout.widget.DrawerLayout 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/drawerlayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <fragment
        android:id="@+id/fragment"
        android:name="androidx.navigation.fragment.NavHostFragment"
        android:layout_width="0dp"
        android:layout_height="0dp"
        app:defaultNavHost="true"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:navGraph="@navigation/nav_graph" />

    <com.google.android.material.navigation.NavigationView
        android:id="@+id/navigationView"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:layout_gravity="start|left"
        app:headerLayout="@layout/nav_header_main"
        app:menu="@menu/side_menu" />

</androidx.drawerlayout.widget.DrawerLayout>
```

4. Open the main activity class and add the following three declarations.

```
private lateinit var drawerLayout: DrawerLayout
private lateinit var appBarConfiguration: AppBarConfiguration
private lateinit var navController: NavController
```

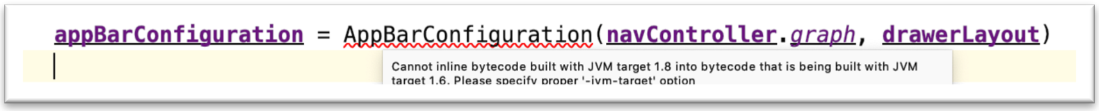
5. Inside the OnCreate function add the following

```
drawerLayout = findViewById(R.id.drawerlayout)
navController = findNavController(R.id.fragment)

navigationView.setupWithNavController(navController)

appBarConfiguration = AppBarConfiguration(navController.graph, drawerLayout)
setupActionBarWithNavController(navController, appBarConfiguration)
```

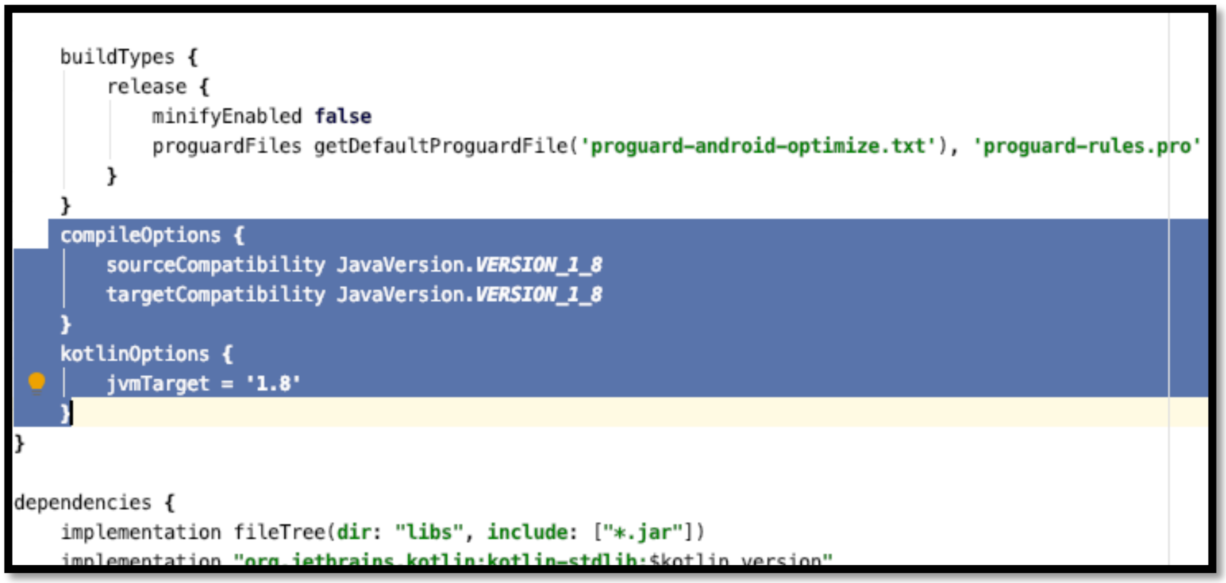
If you see the error shown below



```
appBarConfiguration = AppBarConfiguration(navController.graph, drawerLayout)
```

Cannot inline bytecode built with JVM target 1.8 into bytecode that is being built with JVM target 1.6. Please specify proper '-jvm-target' option

then you can fix it by adding the following code into your gradle app module and sync



```
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
        }
    }
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
    kotlinOptions {
        jvmTarget = '1.8'
    }
}

dependencies {
    implementation fileTree(dir: "libs", include: ["*.jar"])
    implementation "org.jetbrains.kotlin:kotlin-stdlib:$kotlin_version"
```

6. Test the app as you make progress.
7. Implement the **NavController.OnDestinationChangedListener** and Override the **onDestinationChanged.** This method should allow you to load different context menus depending on the destination. But for now, we will only toast the name of the destination
8. Override the **onResume** and **onPause** methods to attach and detach the listener. Otherwise, it will not work.

```
override fun onResume() {
    super.onResume()
    navController.addOnDestinationChangedListener(::onDestinationChanged)
}

override fun onPause() {
    super.onPause()
    navController.removeOnDestinationChangedListener(::onDestinationChanged)
}
```