ANDROID NAVIGATION :

The Navigation component consists of three key parts that are described below:

- Navigation graph: An XML resource that contains all navigation-related information in one centralized location. This includes all of the individual content areas within your app, called *destinations*, as well as the possible paths that a user can take through your app.
- NavHost: An empty container that displays destinations from your navigation graph. The Navigation component contains a default NavHost implementation, NavHostFragment, that displays fragment destinations.
- NavController: An object that manages app navigation within a NavHost. The NavController orchestrates the swapping of destination content in the NavHost as users move throughout your app.

The Navigation component provides a number of other benefits, including the following:

- Handling fragment transactions.
- Handling Up and Back actions correctly by default.
- Providing standardized resources for animations and transitions.
- Implementing and handling deep linking.
- Including Navigation UI patterns, such as navigation drawers and bottom navigation, with minimal additional work.
- Safe Args - a Gradle plugin that provides type safety when navigating and passing data between destinations.
- ViewModel support - you can scope a ViewModel to a navigation graph to share UI-related data between the graph's destinations.

Note:

The <navigation> element is the root element of a navigation graph. As you add destinations and connecting actions to your graph, you can see the corresponding <destination> and <action> elements here as child elements. If you have nested graphs, they appear as child <navigation> elements.

Add a NavHost to an activity

The navigation host is an empty container where destinations are swapped in and out as a user navigates through your app.

The Navigation component's default NavHost implementation, NavHostFragment, handles swapping fragment destinations.

Note: The Navigation component is designed for apps that have one main activity with multiple fragment destinations. The main activity is associated with a navigation graph and contains a **NavHostFragment** that is responsible for swapping destinations as needed. In an app with multiple activity destinations, each activity has its own navigation graph.

FEW IMP POINTS:

- The app:defaultNavHost="true" attribute ensures that your NavHostFragment intercepts the system Back button. Note that only one NavHost can be the default. If you have multiple hosts in the same layout (two-pane layouts, for example), be sure to specify only one default NavHost.

```
<?xml version="1.0" encoding="utf-8"?>
<navigation xmlns:app="http://schemas.android.com/apk/res-auto"
```

```
    xmlns:tools="http://schemas.android.com/tools"
    xmlns:android="http://schemas.android.com/apk/res/android"
    app:startDestination="@id/blankFragment">
    <fragment
        android:id="@+id/blankFragment"

android:name="com.example.cashdog.cashdog.BlankFragment
"
        android:label="@string/label_blank"
        tools:layout="@layout/fragment_blank" />
</navigation>
```

The *start destination* is the first screen users see when opening your app, and it's the last screen users see when exiting your app. The Navigation editor uses a house icon to indicate the start destination.

Connect destinations :

An *action* is a logical connection between destinations. Actions are represented in the navigation graph as arrows. Actions usually connect one destination to another, though you can also create global actions that take you to a specific destination from anywhere in your app.

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

android:name="com.example.cashdog.cashdog.BlankFragment"
        android:label="@string/label_blank"
        tools:layout="@layout/fragment_blank" >
        <action
            android:id="@+id/
action_blankFragment_to_blankFragment2"
            app:destination="@id/blankFragment2" />
    </fragment>
    <fragment
        android:id="@+id/blankFragment2"

android:name="com.example.cashdog.cashdog.BlankFragment2"
        android:label="@string/label_blank_2"
        tools:layout="@layout/fragment_blank_fragment2" />
</navigation>
```

Navigate to a destination :

Navigating to a destination is done using a NavController, an object that manages app navigation within a NavHost.

Each NavHost has its own corresponding NavController.

You can retrieve a NavController by using one of the following methods
Kotlin:

- Fragment.findNavController()
- View.findNavController()
- Activity.findNavController(viewId: Int)

You should retrieve the NavController directly from the NavHostFragment

```
val navHostFragment =

supportFragmentManager.findFragmentById(R.id.nav_host_fragmen
t) as NavHostFragment
val navController = navHostFragment.navController
```

Ensure type-safety by using Safe Args :

The recommended way to navigate between destinations is to use the Safe Args Gradle plugin.

This plugin generates simple object and builder classes that enable type-safe navigation and argument passing between destinations.

EXAMPLE :

As an example, assume we have a navigation graph with a single action that connects the originating
destination, SpecifyAmountFragment, to a receiving
destination, ConfirmationFragment.

Safe Args generates a SpecifyAmountFragmentDirections class with a single
method, actionSpecifyAmountFragmentToConfirmationFragment() t hat returns a NavDirections object. This
returned NavDirections object can then be passed directly
to navigate(), as shown in the following example:

```
override fun onClick(view: View) {
    val action =
        SpecifyAmountFragmentDirections

  .actionSpecifyAmountFragmentToConfirmationFragment()
    view.findNavController().navigate(action)
}
```

AND TO RETRIEVE:
In your receiving destination's code, use
the getArguments() method to retrieve the bundle and use its contents. When using the -ktx dependencies, Kotlin users can also

use the by navArgs() property delegate to access arguments

```
val args: ConfirmationFragmentArgs by navArgs()

override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    val tv: TextView = view.findViewById(R.id.textViewAmount)
    val amount = args.amount
    tv.text = amount.toString()
}
```

Pass data between destinations with Bundle objects :

1). If you aren't using Gradle, you can still pass arguments between destinations by using Bundle objects.
2). Create a Bundle object and pass it to the destination using navigate()

```
val bundle = bundleOf("amount" to amount)
view.findNavController().navigate(R.id.confirmationAction, bundle)
```

In your receiving destination's code, use the getArguments() method to retrieve the Bundle and use its contents:

```
val tv = view.findViewById<TextView>(R.id.textViewAmount)
tv.text = arguments?.getString("amount")
```

Pass data to the start destination

You can pass data to your app's start destination. First, you must explicitly construct a Bundle that holds the data. Next, use one of the following methods to pass the Bundle to the start destination:

- If you're creating your NavHost programmatically, call NavHostFragment.create(R.navigation.graph, args), where *args* is the Bundle that holds your data.
- Otherwise, you can set start destination arguments by calling one of the following overloads of NavController.setGraph():
  - Using the ID of the graph: navController.setGraph(R.navigation.graph, args)
  - Using the graph itself: navController.setGraph(navGraph, args)

To retrieve the data in your start destination, call Fragment.getArguments().

Note: when manually calling **setGraph()** with arguments, you must **not** use the **app:navGraph** attribute when creating the **NavHostFragment** in XML as that will internally call **setGraph()** without any arguments, resulting in your graph and start destination being created twice.