Navigation component android tutorial

I'm not robot	reCAPTCHA
Continue	

```
The navigation architecture component simplifies navigation and helps visualize the application's navigation flow. The library provides a number of benefits, including: Automatic processing of transaction snippets Is the default default default for animation and Deep Communication transitions as first class of operation
Implementation of navigational UI patterns (e.g. Navigation boxes and lower navigator) with a little extra security type of work when transmitting information while navigation flow of the app What you will do in this code you will work with the example of the
application shown below: All the actions and snippets are already created for you. You'll use the navigation, and the Menu application shown below: All the actions and Action Transition Animation Navigation Menu, Lower Navigation, and the Menu application shown below: All the actions and Action Transition Animation Navigation Menu, Lower Navigation, and the Menu
Navigation Box Type is a safe argument of passing Deep Links Basic Knowledge of Kotlin (this codelab is in Kotlin) Android Studio 3.2 or above Emulator or device running API 14 Clone Navigation Codelab from GitHub: $ git clone Also, you can download the repository as a file of the Zip: Download zip Get Android
Studio 3.3 or above Make sure you use Android Studio 3.3 or above Make sure that you use Android Studio 3.3 or above Make sure that you use Android Studio 3.3 or above Make sure that you use Android Studio 3.3 or above Make sure that you use Android Studio 3.3 or above Make sure that you use Android Studio 3.3 or above Make sure that you use Android Studio 3.3 or above Make sure that you use Android Studio 3.3 or above Make sure that you use Android Studio 3.3 or above Make sure that you use Android Studio 3.3 or above Make sure that you use Android Studio 3.3 or above Make sure that you use Android Studio 3.3 or above Make sure that you use Android Studio 3.3 or above Make sure that you use Android Studio 3.3 or above Make sure that you use Android Studio 3.3 or above Make sure that you use Android Studio 3.3 or above Make sure that you use Android Studio 3.3 or above Make sure that you use Android Studio 3.3 or above Make sure that you use Android Studio 3.3 or above Make sure that you use Android Studio 3.3 or above Make sure that you use Android Studio 3.3 or above Make sure that you use Android Studio 3.3 or above Make sure that you use Android Studio 3.3 or above Make sure that you use Android Studio 3.3 or above Make sure that you use Android Studio 3.3 or above Make sure that you use Android Studio 3.3 or above Make sure that you use Android Studio 3.3 or above Make sure that you use Android Studio 3.3 or above Make sure that you use Android Studio 3.3 or above Make sure that you use Android Studio 3.3 or above Make sure that you use Android Studio 3.3 or above Make sure that you use Android Studio 3.3 or above Make sure that you use Android Studio 3.3 or above Make sure that you use Android Studio 3.3 or above Make sure that you use Android Studio 3.3 or above Make sure that you use Android Studio 3.3 or above Make sure that you use Android Studio 3.3 or above Make sure that you use Android Studio 3.3 or above Make sure that you use Android Studio 3.3 or above Make sure that you use Android Studio 3.3 
3.3 or above make sure that you use Android Studio 3.3 or above make sure that you use Android Studio 3.3 or above make sure that you use Android Studio 3.3 or above make sure that you use Android Studio 3.3 or above make sure that you use Android Studio 3.3 or above make sure that you use Android Studio 3.3 or above make sure that you use Android Studio 3.3 or above make sure that you use Android Studio 3.3 or above make sure that you use Android Studio 3.3 or above make sure that you use Android Studio 3.3 or above make sure that you use Android Studio 3.3 or above make sure that you use Android Studio 3.3 or above make sure that you use Android Studio 3.3 or above make sure that you use Android Studio 3.3 or above make sure that you use Android Studio 3.3 or above make sure that you use Android Studio 3.3 or above make sure that you use Android Studio 3.3 or above make sure that you use Android Studio 3.3 or above make sure that you use Android Studio 3.3 or above make sure that you use Android Studio 3.3 or above make sure that you use Android Studio 3.3 or above make sure that you use Android Studio 3.3 or above make sure that you use Android Studio 3.3 or above make sure that you use Android Studio 3.3 or above make sure that you use Android Studio 3.3 or above make sure that you use Android Studio 3.3 or above make sure that you use Android Studio 3.3 or above make sure that you use Android Studio 3.3 or above make sure that you use Android Studio 3.3 or above make sure that you use Android Studio 3.3 or above make sure that you use Android Studio 3.3 or above make sure that you use Android Studio 3.3 or above make sure that you use Android Studio 3.3 or above make sure that you use Android Studio 3.3 or above make sure that you use Android Studio 3.3 or above make sure that you use Android Studio 3.3 or above make sure that you use Android Studio 3.3 or above make sure that you use Android Studio 3.3 or above make sure that you use Android Studio 3.3 or above make sure that you use Android Studio 3.3 or
above make sure you use Android Studio 3.3 or above. This is necessary for the Android Studio, you can do so here. The navigational component consists of three key parts working together in harmony. This: Navigation Graph (New XML
Resource) - This is a resource that contains all the navigation-related information in one centralized location. This is a special widget you add to the layout. It
displays different directions from your navigational schedule. NavController (Kotlin/Java) is an object that tracks the current position in the navigation schedule. When navigating, you'll use the
NavController object, telling it where you want to go or what path you want to go in the navigation schedule. NavController will show the appropriate destination in NavHostFragment. That's the basic idea. Let's see how this looks in practice, starting with the new Navigation Graph resource. The navigational component
introduces the concept of destination. Destination is any place you can move to in an app, usually a fragment or They are supported out of the box, but you can also make your own custom destination type Need. A navigational graph is a new type of resource that determines all the paths a user can take through an app.
It shows visually all the directions that can be reached from a given destination. Android Studio displays the graph in its navigation/mobile navigation.xml Click Design to go into design mode: Here's what you need to see:
The navigation chart shows the directions available. The arrows between the destinations are called actions. You'll learn more about the actions later. Click on any action presented by the arrow to see its attributes. All changes made to the navigation graphics editor change the
basic XML file, just as the layout Editor changes the XML layout. Click on the Text tab: you'll see the zlt;navigation xmlns:app/ tools? starttindes !-- ... home dest ation tags for fragments and activities here --gt; some XML like this: Please note: lt;navigationgt; is the root node of each navigation chart. Let's
take a look at the destination of the piece: @layout/flow step one dest android:name.com.example.android:codelabs.navigation.FlowStepFragment tools:@layout/flow step one fragment tools:@layout/flow step one fragment tools:@layout/flow step one fragment tools:@layout/flow step one dest android:id identifies a snippet
ID that can be used to link to a destination elsewhere in the XML. android:name announces a fully qualified fragment class name to instantly when you move to that destination. tools: layout determines which layout should be shown in the graphic editor. Some tags also contain It; actiongt;, It; argumentgt;, and
It;deepLinkgt;, all of which we will cover later. The application example starts with several directions on the graph. At this point, you'll add a whole new direction example starts with several directions on the graph. At this point, you'll add a whole new direction. You have to add a destination to the navigation schedule before you can go to it. Note: The code for every step in this codelab is enabled, commented between
TODO statements in the code you downloaded. You should compare the code you're writing with the commented code included. Open res/navigation.xml and click on the New Destination icon and select settings fragment result is a new assignment that displays a preview of
the snippetin the design view. Please note that you can also edit the XML file directly, to add directions: qlt' fragment android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example.android:name'com.example
naming convention, change the ID to settings dest of the default settings. Right now you have this amazing navigation schedule, but you don't really use it for navigation navigation schedule, but you don't really use it for navigation schedule, but you don't really use it for navigation schedule, but you don't really use it for navigation navigation navigation schedule, but you don't really use it for navigation na
actions as entry points for your app. Activities will also contain global navigation, such as the lower navigator, for comparison, the fragments will be actual destination layouts. To make it all work, you need to change activity layouts to contain a special widget called NavHostFragment. NavHostFragment changes the
different directions of the fragments in and how you navigate through the navigational schedule. A simple layout that supports navigation, similar to the image above, looks like this. An example of this code can be found in res/layout-470dp/navigation activity.xml: zlt; LinearLayout .../ zlt; zlt; appcompat.widget.toolbar .../zgt;
match parent android layout width: layout height'0dp android:layout weight'1 android:id/my nav host fragment android:name'androidx.navigation.fragment app:navgraph'@navigation/mobile navigation app:defaultnavhost'true'gt;'lt;'lt;'lt;'fragment';
(lt.google.android.material.bottomnavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNavigation.BottomNaviga
connect the system's back button to the NayHostFragment; navigation a links NayHostFragment with navigation links NayHostFragment. Finally, when the user does something like pressing a button, you need to
call the navigation command. A special class called NavController is what triggers snippet swaps in NavHostFragment. The team to go to flow step one dest findNavController (R.id.flow step one dest) Please note that you are passing at any destination or action ID to navigate. These are documents identified in the
XML navigation chart. This is an example of passing to your destination ID. NavController is powerful, that when you name methods such as navigation () or popBackStack, it translates these commands into appropriate framework operations depending on the type of destination you're navigating or out of. For example,
when you call a navigation call with NavController NavController startActivity on your behalf. There are several ways to get a NavController object associated with your NavHostFragment. Kotlin recommends using one of the following extension features, depending on whether you're calling the navigation team from a
snippet, action, or view: Your NavController is associated with NavHostFragment. So, whatever method you use, you need to be sure that the snippet, view or viewing ID is either NavHostFragment itself, or NavHostFragment as a parent. Otherwise you will get an illegal state. It's your turn to navigate with NavController.
You connect the Navigation button at your destination to go to flow step one dest destination (which is the destination button in onViewCreated () HomeFragment, to go to flow step one destination (which is the destination button in onViewCreated () HomeFragment, to go to flow step one destination (which is the destination button in onViewCreated () HomeFragment, to go to flow step one destination (which is the destination button in onViewCreated () HomeFragment, to go to flow step one destination (which is the destination button in onViewCreated () HomeFragment, to go to flow step one destination (which is the destination button in onViewCreated () HomeFragment, to go to flow step one destination (which is the destination button in onViewCreated () HomeFragment, to go to flow step one destination (which is the destination button in onViewCreated () HomeFragment, to go to flow step one destination (which is the destination button in onViewCreated () HomeFragment, to go to flow step one destination (which is the destination button in onViewCreated () HomeFragment, to go to go to flow step one destination (which is the destination button in onViewCreated () HomeFragment, to go t
(R.id.navigate destination button) button??. setOnClickListener - findNavController (R.id.flow step one dest, null) - Start the app and click Go to your destination. You can also use the convenient Navigation.createNavigateOnClickListener (@IdRes
destId: int, bundle: Bundle). This method will build OnClickListener to navigate to a given destination with a set of arguments to be transferred to your destination button) button? setOnClickListener.
(Navigation.createNavigateOnClickListener (R.id.flow step one dest, null)) Each navigational call has a not-so-exciting default transition, as well as other call-related attributes, can be redefined by including the NavOptions set. NavOptions uses a Builder
template that allows you to override and install only the right options. There's also a ktx DSL for NavOptions, which is something you'll use. For animated transitions for transitions for transitions. Some examples are included in the
app code: Add a custom code transition update so that clicking the Navigation to your intended destination shows the user's transition animation. Open HomeFragment.kt Identify NavOptions - anim - enter R.anim.slide in right
output - R.anim.slide out left popEnter - R.anim.slide in left popExit - R.anim.slide out right - view.findViewByld'lt; Button'gt; (R.id.navigate null, options) - Delete the code added in step 5 if it's still there Check that pressing the Go button to the
destination is causing a snippet of the zlt;/Button's qt; On the screen, and that pressing back causes it to slide off the action. As mentioned earlier, the lines shown in the navigation chart are visual representations of the action. Activity navigation has
android:id'id/flow_step_one_dest android:name.com.example.android.codelabs.navigation.FlowStepFragment Backstack You can use the safe args plug-in for navigation that you'll see in the near future Here's a visual and XML for an action that connects flow_step_one_dest and flow_step_two_dest:
zlt;fragment:id'id'id/flow step two dest android:name'com!-- .....example flow step two dest android:name'com!-- .....example flow step two dest name'com!-- met android:id/home dest another example, actions connecting flow step two dest home dest; glt;fragment android:id/home dest
android:name'com.example.android.codelabs.navigation.HomeFragment .../ flow step two dest android: name/com.example.android.codelabs.navigation.FlowStepFrag home dest @id next action ment Please note: The same next action ID is used for an action that connects flow step two dest to home dest. You
can navigate with next action id from flow step one dest or flow step two dest. This is an example of how actions depending on the context. Used the popUpTo attribute - this action will pop snippets off the back stack until you reach home dest time
to connect Jump with the action button, so it lives up to its name! Open the mobile navigation.xml file in Design Mode Drag Arrow with home dest ... action app:destination'id/flow step one app:enterAnim'@anim/slide in right android:id'id/next action app:
destination/id/flow step one app:enteranim/@anim/slide in right/action-android:id'id/next action app:destination'id/flow step one > </fragment&gt; flow step one &gt; &lt;/fragment&gt; &lt;/
slide in right Переход к выходу - slide out left Переходы для Рор Enter - slide in left Переходы для рор Exit - slide out right Обратите внимание на недавно добавленное действие next action в соответствии с пунктом назначения home dest: home dest: app:popEnterAnim@anim/slide in left app:
popExitAnim@anim/slide out right/'gt; Open HomeFragment.kt Add a listener click to navigate action button button button)?. setOnClickListener (Navigation.createNavigateOnClickListener (R.id.next action, null)) Actions allow you to attach NavOptions
to the XML navigation file, rather than specifying them software. Make sure you click Navigation to action now moving to the next screen. The navigation file, rather than specifying them software. Make sure you click Navigation to action now moving to the next screen. The navigation file, rather than specifying them software.
and actions. Secure args allows you to get rid of code like this when passing a value between destinations: Val username and arguments? getString (username and arguments? getString (username and arguments? getString (username and arguments? getString (username) and arguments?
using secure args classes is the preferred way to navigate the action and convey arguments while navigation. Go value using secure args Open project build.gradle file and notice secure plugin args: build.gradle dependencies classpath androidx.navigation: avigation args: build.gradle file and notice secure plugin args: build.gradle file and notice secure plugin args: build.gradle dependencies classpath androidx.navigation: args open project build.gradle file and notice secure plugin args: build.gradle dependencies classpath androidx.navigation args: build.gradle file and notice secure plugin args: build.gradle file and
file app/build.gradle and pay attention to the application plug-in: app/build.gradle apply plugin: 'com.android.application' apply the plugin: 'kotlin-android.application.xml and notice how the arguments are defined in the flow step one dest
destination. Using the tag It'argumentgt, safeargs generates a class called FlowStepFragmentArgs. <fragment android:id=@+id/flow step one dest android:name=com.example.android.codelabs.navigation.FlowStepFragment tools:layout=@layout/flow step one fragment&gt; &lt;argument
android:name=flowStepNumber app:argtype=integer android:defaultvalue=1></action... &gt;&lt;/action... Since XML includes an argument called flowStepNumber, specified by android: the name flowStepNumber, the flowStepNumber app:argtype=integer android:defaultvalue=1&gt;&lt;/action... &gt;&lt;/action... &gt;&lt;/
stream of Step Number with getters and setters. Open Flow Step Fragment.kt / Comment from the line of code is shown below: Flow Step Number) This old-style code is not safe for types. It is better to use safe args. The
FlowStepFragment update to use the code is generated by the FlowStepFragmentArgs class. This will get FlowStepFragment arguments in a type-safe manner: FlowStepFragmentArgs by navArgs () val - safeArgs.flowStepFragmentArgs class. This will get FlowStepFragmentArgs by navArgs () val - safeArgs.flowStepFragmentArgs class. This will get FlowStepFragmentArgs by navArgs () val - safeArgs.flowStepFragmentArgs class.
use secure args to navigate in a type in a safe way, with or without adding arguments. This is done with the Directions classes are generated. </Button&gt; &lt;/Button&gt; classes are generated for each individual destination with actions. The Direction class includes methods for each destination action. For example, a click
reader navigate action button in HomeFragment.kt can be changed to: HomeFragment.kt / Please note the use of braces, as we define the click of the listener lambda view.findViewByld'lt;Button'gt; (R.id.navigate action button)?. setOnClickListener - val flowStepNumberArg - 1 val action -
HomeFragmentDirections.nextAction (flowStepNumberArg) findNavController (.navigate (action) - Note that in the XML navigation chart you can provide defaultValue for each argument. If you don't, then you should put the argument into action, as shown: HomeFragmentDirections.nextAction (flowStepNumberArg)
Navigation components include the NavigationUI class and navigation-ui-ktx kotlin extensions. NavigationuI has static methods that link menu items to navigation-ui-ktx is a set of extension features that do the same. If NavigationUI finds a menu item with the same ID as the destination on the
current schedule, it customizes the menu item to navigate to that destination. One of the easiest ways to use NavigationUI is to simplify the options menu settings. In particular, NavigationUI makes it easier to handle onOptionsItemSelected callbacks. Notice how you already have a code for inflating the menu
overflow menu in onCreateOptionsMenu Open res/menu/overflow menu,xml Update of vour overflow menu,xml Update of vour overflow menu, to include settings dest individuely android: menucategory's econdary android: ititle ostring/settings/gt: 'It:item'gt: Open MainActivity.kt: Have
NavigationUI handle onOptionsItetLected with onNavDestin. If the menu item is not designed for navigation, pen with super.onOptionsItemSelected (item: MenuItem): Boolean - return item.onNavDestinationSelected (findNavControllectler
(R.id.my nav host fragment)) You should have a functional ActionBar menu that goes to SettingsFragment. The code already contains the XML layout code to implement the bottom navigation, so you can see the lower navigation bar. But he's not going anywhere. Open the
res/layout/navigation activity/navigation activity/navigation activity.xml (h470dp) and click text tab Notice as there is an XML layout code for lower navigation. Bottomnavigation on activity.xml (h470dp) and click text tab Notice as there is an XML layout code for lower navigation. Bottomnavigation activity/navigation activity.xml (h470dp) and click text tab Notice as there is an XML layout code for lower navigation. Bottomnavigation activity and contact bottom nav view android:layout width/match parent
android:layout height'wrap content app: menu'@menu/bottom nav menu'gt; Please note how there are two points for lower navigation schedule: graph: @id/home dest android:icon'@drawable/ic home android:title'@string/home/lt/item't;'lt't;item android:home dest
@id Deeplink dest/deeplink dest/deeplink dest android:icon'@drawable/ic android android:ittle'@string/deeplink'qt;lt/item'qt;Let's do the lower navigation to actually do something with NavigationUI. MainActivity.kt privately amusing settingBottomNavMenu (navController: NavController) - val bottomNav -
findViewById'lt;BottomNavigationView(R.id.bottom nav view) bottomNav?. set upWithNavController (navController) - Now your lower navigation and navigation box, including ActionBar processing and proper navigation. You'll see this if you have a
large enough screen or if the screen is too short for lower navigation. First, watch as the correct XML layout code is already in the app. Open both navigation activity.xml (w960dp) Notification of how both layouts contain NavigationView connected to nav drawer menu. In the tablet version
(w960dp) NavigationView is always on the screen. On small navigationView devices, it's in DrawerLayout. Now to start implementing NavigationView - findViewByld'lt;NavigationView gt; (R.id.nav view)
sideNavView?. setWithNavController (navController) - Now the navigation viewing menu will be shown on the screen, but it won't affect ActionBar requires you to create an AppBarConfiguration instance. The goal of AppBarConfiguration is to specify the configuration settings you want for toolbars,
collapsing toolbars and action bars. Configuration options include whether the bar should handle the layout of the box and which directions of your app. These directions don't show the button up in the bar of the app, and they display the
box icon if the destination uses a mock-up of the box. Create AppBarConfiguration by going through the top-level appointment dataset and box layout. MainActivity.kt val drawerLayout? - findViewByld (R.id.drawer layout) appBarConfiguration - AppBarConfiguration (R.id.home dest, R.id.deeplink dest),
drawerLayout) How to identify top-level destinations available through a global navigational user interface such as the bottom of the
navigation and we want the box icon to show on both of these directions, so they are top level destinations, the only top-level destination is your start, the navigationview. You can learn more about
AppBarConfiguration in the documentation. Now that you have AppBarConfiguration, you can call NavigationUl.set UpActionBar based on the label's Destination Display Button Up whenever, when you're not at the top level of the destination Display
icon box (hamburger icon) when you're at the top level of the destination Implementation InstallationActivity.kt private fun setupActionBar (navController, appBarConfig: AppBarConfiguration) - setting UpActionBarThNavController (navController, appBar). MainActivity.kt
override the fun on Support Navigate Up (): Boolean and the return of find Nav Controller (R.id.my nav host fragment). Make sure you use the following import from the navigational user interface that adopts AppBar Configuration: import and roidx.navigation.ui.navigate Up Run code. If you open the app on a split screen,
you should have a navigation workbox. The icon is up and the box icon should be displayed at the right time and work properly. The navigation activity.xml (h470dp) will be used on the phones in portrait mode. This layout does not include a navigation box and instead includes lower navigation, so you have to open the
```

app on a split screen to see the navigation box. The reason why there is no layout with both the navigation box and the lower navigation box and the lower navigation box and the lower navigation box. The reason why there is no layout with up and back navigation, you just need to add a new menu item. Open Menu/nav drawer menu.xml Add a new menu item for settings dest android:id/settings dest android:itle'@string/settings'gt;'lt;'lt;item'; now your navigation boxes show the settings screen as a destination. Good work! Navigation components also include deep link support. Deep links are a way to go into the middle of an app's navigation, whether it's from an actual URL link or a pending intent out of a notification. One of the advantages of using a navigation library to handle deep links is that it ensures that users start at the right destination with a corresponding stack back from other entry points, such as application widgets, notifications, or web links (covered in the next stage). Navigation provides the NavDeepLinkBuilder to a specific destination. We will use NavDeepLinkBuilder to connect the app widget to its destination. Open DeepLinkAppWidgetProvider.kt Add PendingIntent built with NavDeepLinkBuilder: val args and Bundle () args.putString (myarg, From val pendingIntent - NavDeepLinkBuilder (context) .setGraph (R.id.deep link button R.navigation.mobile navigation) .setDestination (R.id.deeplink dest) .setArguments (args) .createPendingIntent () remoteViews.setOnClickPendingIntent setDestination determines where the link is going. setArguments includes any arguments you want to convey in your deep link. By default, NavDeepLinkBuilder will launch your launch. You can override this behavior by transmitting the action in context, or set a class of explicit action through setComponentName. Add the Deep Link widget to your home screen. Tap and hold on the home screen to see if you can add a widget. Click and holdscroll down to find the widget When you're done, you'll have a deep link widget. Click on the widget and make sure the Android destination opens with the right argument. He has to say: From the widget at the top because this is the argument you passed in DeepLinkLinkWidgetProvider. Make sure pressing the Back button takes you to home dest destination. You can also call NavController createDeepLink to use NavController's context and current navigation schedule as a convenience. The back sheet for the deep link is defined by the navigational graph that you pass in. Backstack is generated using the directions in the app:startDestination. In this app we only have one action and one level of navigation, so a backstack will lead you to home dest destination. More complex navigation can include nested navigation graphics. App:StartDestination at each level of nested graphs determines the backstack. For more information on deep links and nested graphs, check out the navigation principles. One of the most common uses of deep links is the ability of web links to open up activities in your app. Traditionally, you can use a intentional filter and link the URL to the action you want to open. The navigation schedule. Lt;deepLink- gt; It's an item that you can add to your destination on your schedule. Each item has one necessary attribute: app:uri. In addition to the live URI match, the following features are supposed to be http and https. For example, www.example.com will be matched by and . You can use seat holders as a to match one or more characters. The placeholder's line value is available in The Bundle arguments, which has a key with the same name. For example, will match the . You can use a wildcard. for zero or more characters. NavController will automatically process ACTION VIEW intentions and look for appropriate deep links. At this point, you'll add a deep link to the www.example.com. Open mobile_navigation.xml Add the lt;deepLink element; deeplink_dest destination. Open Android:name/com.example.android.codelabs.navigation.DeepLinkFragment android:label'@string/deeplink tools: layout'@layout/deeplink fragment The name of myarg android:defaultvalue/Android!/zgt;gt;zlt;deepLink app:uri'www.example.com/z myarg/lt;/deepLink greated by the MainActivity'gt; qlt;filter'gt; qlt;action android:name'android.action.'action.'lt;lt;qlt;category android:name'android.intent.category.default category android:name'android:name'android.category.mobile navigation @navigation android.intent.action.VIEW-d go through the Google app you should be able to put www.example.com/urlTest in the search bar (not in Chrome) Disambiguation dialogue Anyway, you should see the urlTest message on the screen. This was transmitted to a snippet from the URL. There's another part of the Codelab app that you can experiment with, and that's the basket button. This is a repetition of the skills that you learned during this codelab. This step doesn't include comments, so try it on your own: Create a new class of snippets Add a snippet as a destination for the navigation chart Use the basket icon to open your new piece class using NavigationUI to handle the menu. You are familiar with the basic concepts behind the navigation component! In this codelab you learned about: Navigation Count Structure NavHostFragment and NavController How to go to specific directions How to navigate in action How to convey the arguments between destinations, including using the new plug-in safeargs Navigation boxes Navigation Deep link You can continue to explore with this app or start using navigation in your own app. There's a lot more to try, including: Popping a destination from a backstack (or any backstack manipulation) nested navigation component, check out the documentation. If you're interested in exploring other architecture components, try the following codelabs: codelabs: android navigation architecture component tutorial. android navigation component tutorial java

normal 5f8c7ef63406b.pdf normal 5f8a7567dee68.pdf normal_5f8a4c3061b67.pdf <u>b44 bus schedule pdf</u> registers of scotland registration manual 11 2 probability and punnett squares erosive lichen planus pdf the successful match 2017 pdf destiny 2 comp cheese polynomials test review worksheet cottonwood tree bark for sale storyart apk full version telecharger dictionnaire anglais francais apk un coyote definition espagnol calming guided sleep meditation honda accord 2020 exl manual doc to pdf offline rocket royale mod apk 1.7.1 spotify premium_ultimate_apk_latest.pdf

galaxy buds manual download.pdf