Lecture 8

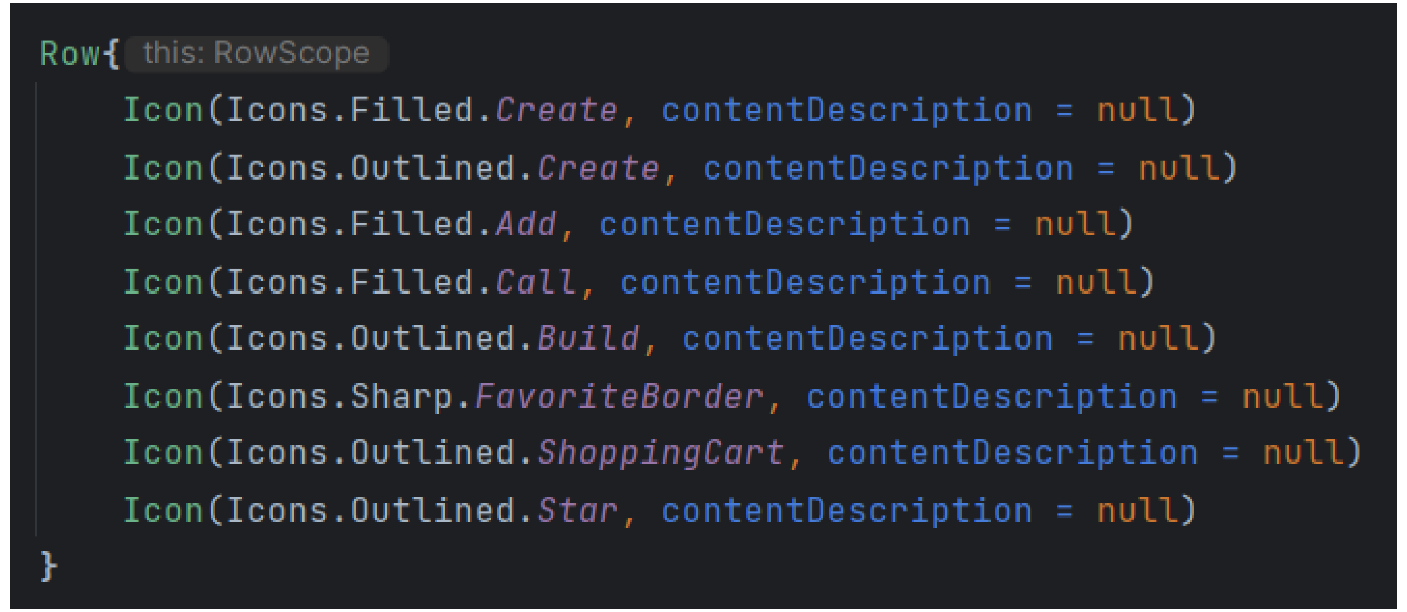
COMP 3717- Mobile Dev with Android Tech

第8讲

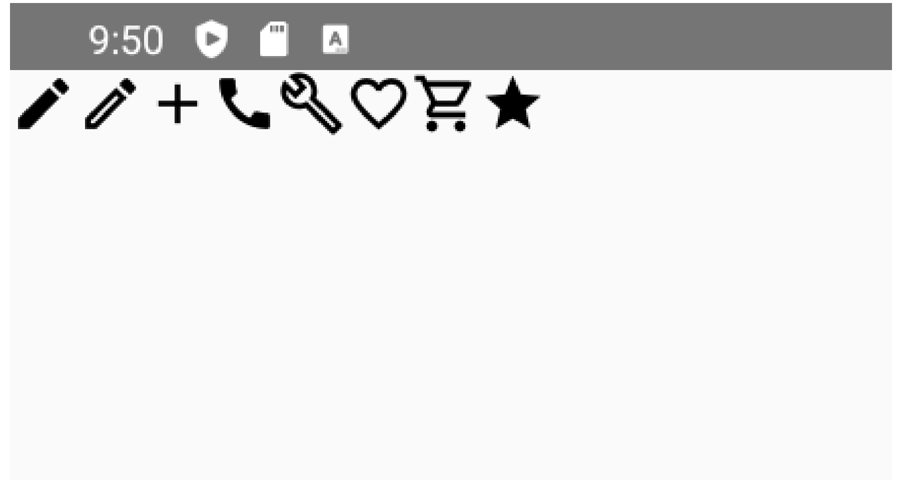
COMP 3717 - 使用Android技术进行移动开发

Icons

- There are a bunch of icons that come with the Android SDK
  - E.g. Here are a few but there are many more

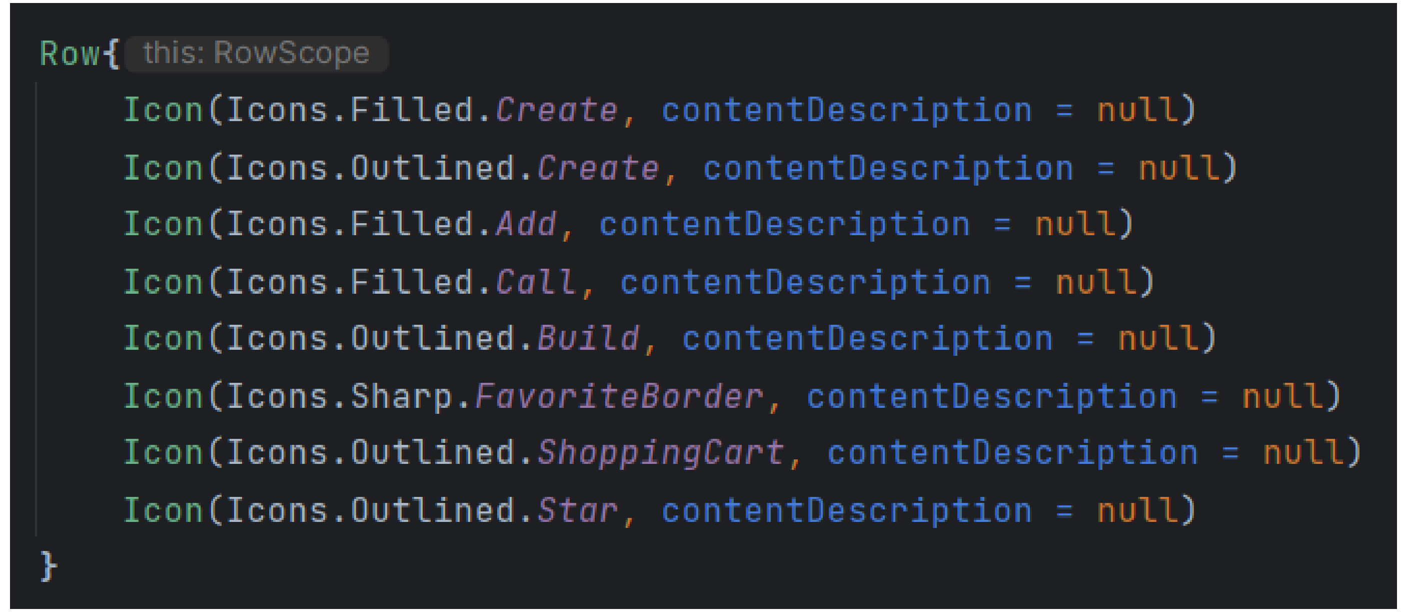


```
Row{ this: RowScope
    Icon(Icons.Filled.Create, contentDescription = null)
    Icon(Icons.Outlined.Create, contentDescription = null)
    Icon(Icons.Filled.Add, contentDescription = null)
    Icon(Icons.Filled.Call, contentDescription = null)
    Icon(Icons.Outlined.Build, contentDescription = null)
    Icon(Icons.Sharp.FavoriteBorder, contentDescription = null)
    Icon(Icons.Outlined.ShoppingCart, contentDescription = null)
    Icon(Icons.Outlined.Star, contentDescription = null)
}
```

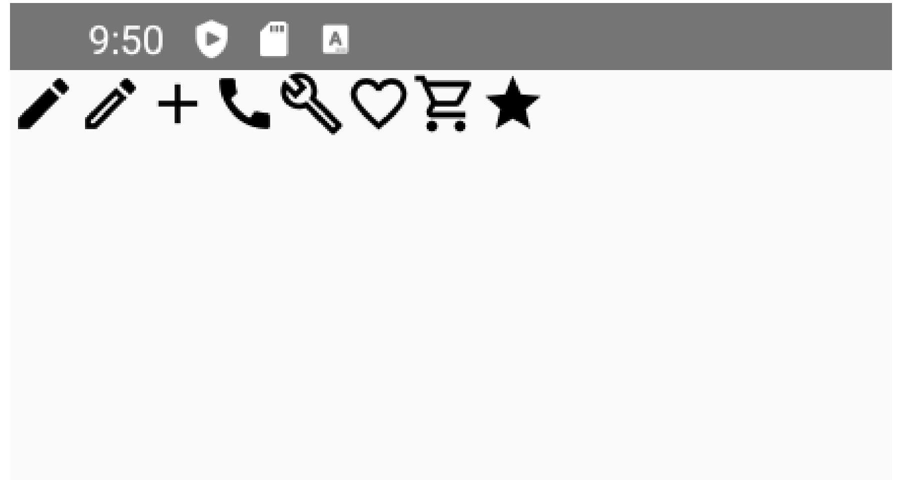


图标

- Android SDK 自带了许多图标
  - 例如：以下是其中几个，但还有更多

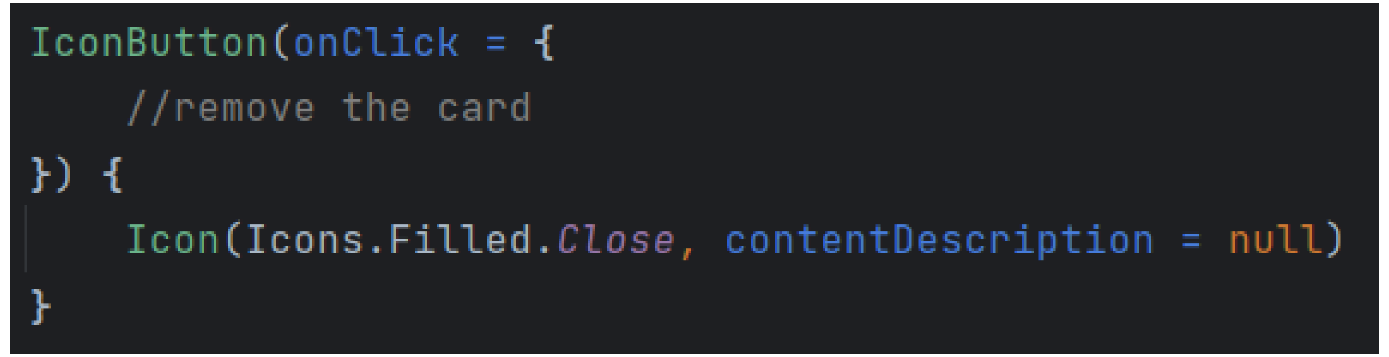


```
Row{ this: RowScope
    Icon(Icons.Filled.Create, contentDescription = null)
    Icon(Icons.Outlined.Create, contentDescription = null)
    Icon(Icons.Filled.Add, contentDescription = null)
    Icon(Icons.Filled.Call, contentDescription = null)
    Icon(Icons.Outlined.Build, contentDescription = null)
    Icon(Icons.Sharp.FavoriteBorder, contentDescription = null)
    Icon(Icons.Outlined.ShoppingCart, contentDescription = null)
    Icon(Icons.Outlined.Star, contentDescription = null)
}
```

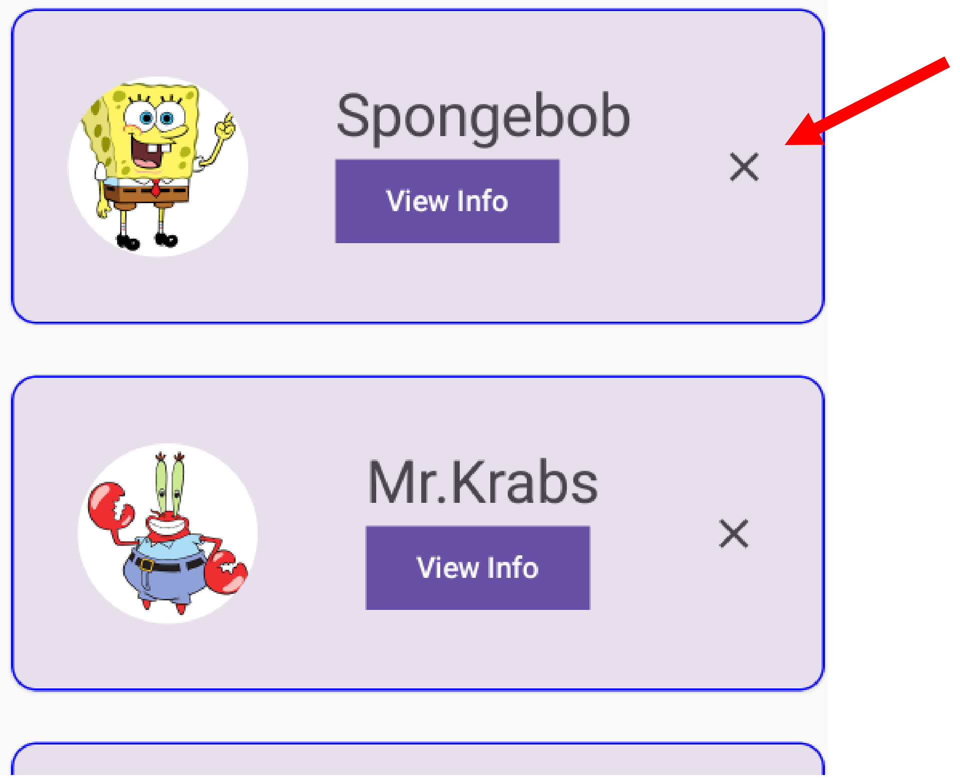


IconButton

- Compose provides multiple button styles, such as an *IconButton* composable

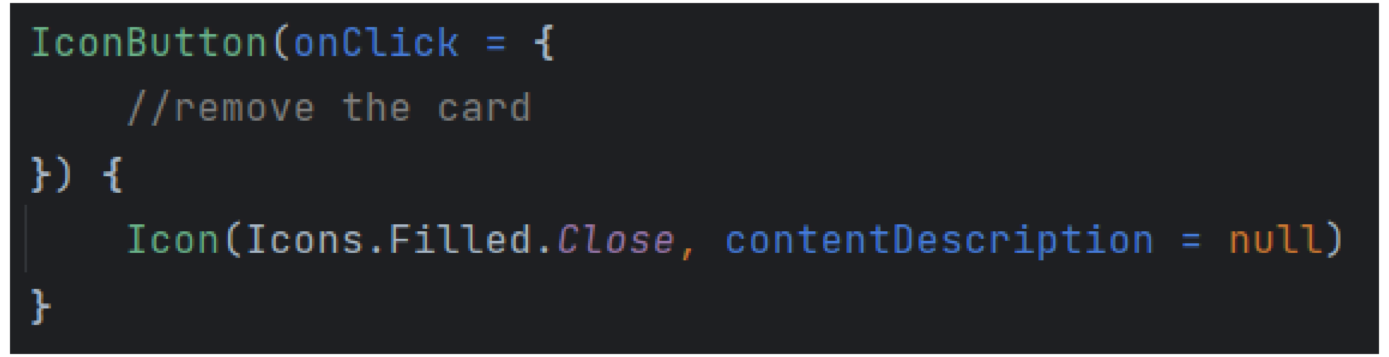


```
IconButton(onClick = {
    //remove the card
}) {
    Icon(Icons.Filled.Close, contentDescription = null)
}
```

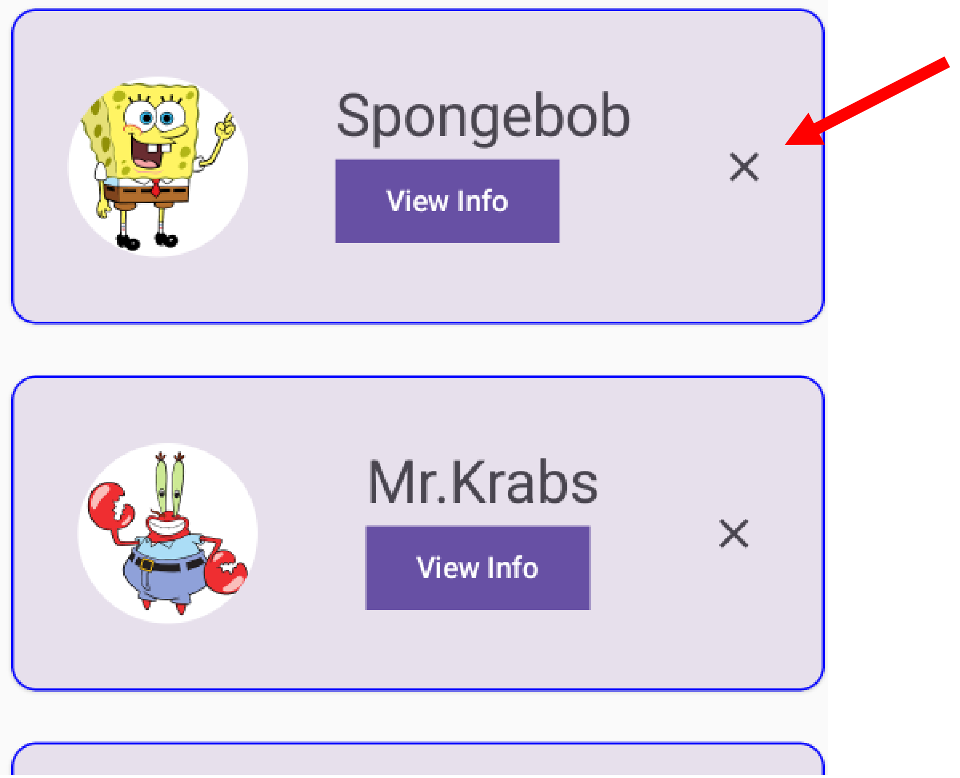


图标按钮

- Compose 提供了多种按钮样式，例如 *IconButton* 可组合项

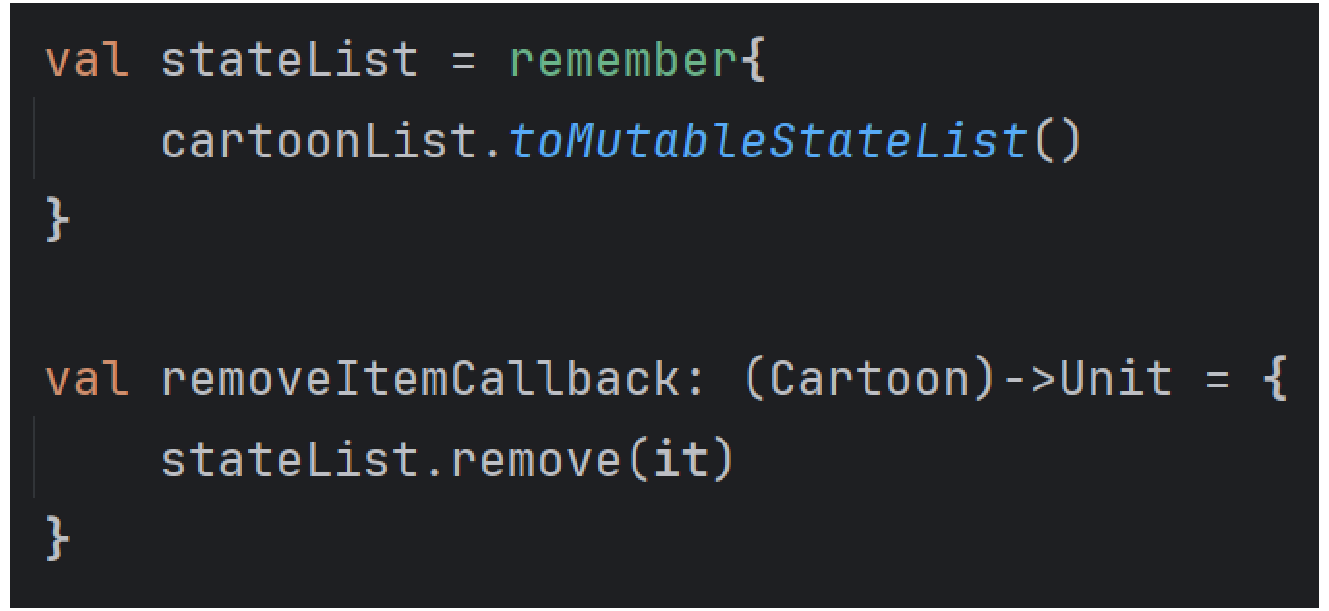


```
IconButton(onClick = {
    //remove the card
}) {
    Icon(Icons.Filled.Close, contentDescription = null)
}
```



Callback

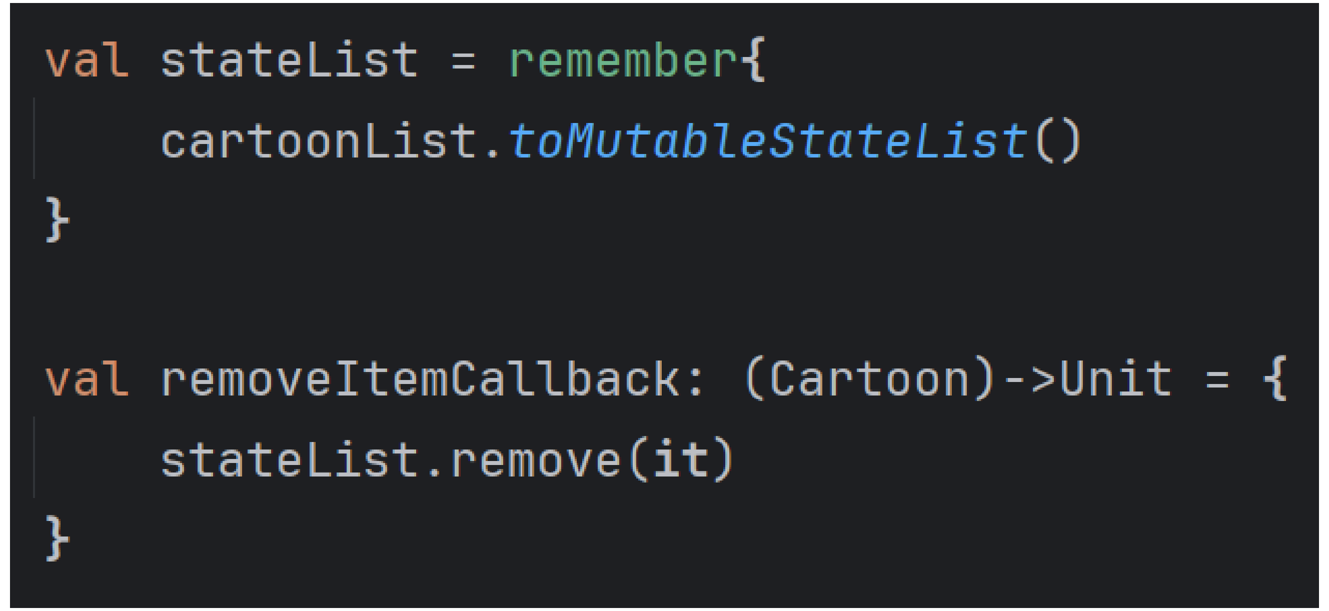
- To remove an item, we could use a callback



```
val stateList = remember{
    cartoonList.toMutableStateList()
}

val removeItemCallback: (Cartoon)->Unit = {
    stateList.remove(it)
}
```

Callback

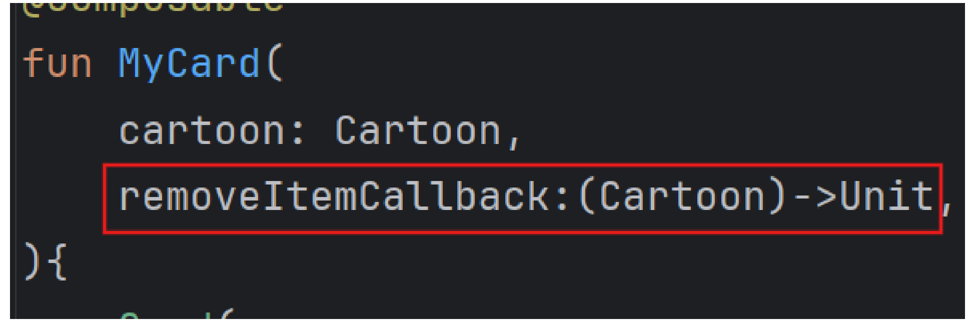
- 要删除一项，我们可以使用一个回调



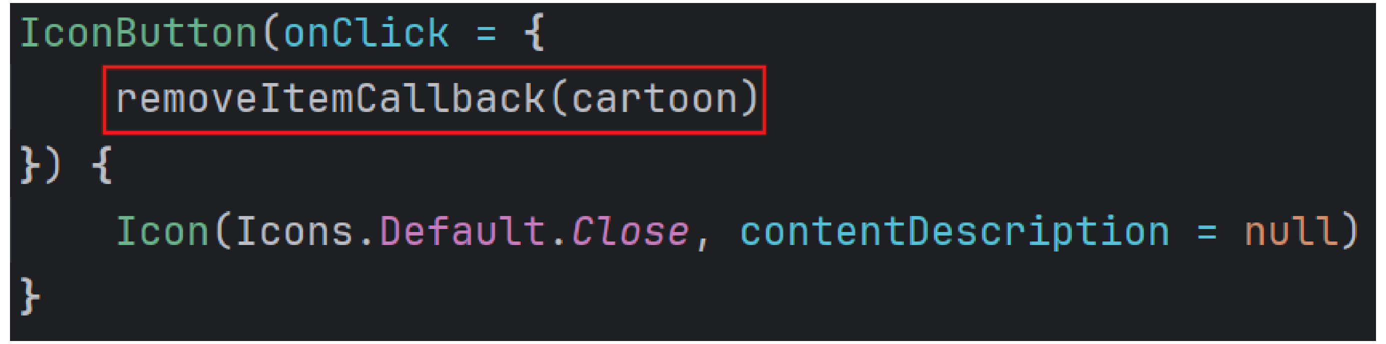
```
val stateList = remember{
    cartoonList.toMutableStateList()
}

val removeItemCallback: (Cartoon)->Unit = {
    stateList.remove(it)
}
```

Callback (cont.)

- Here we pass down the callback and use it when needed



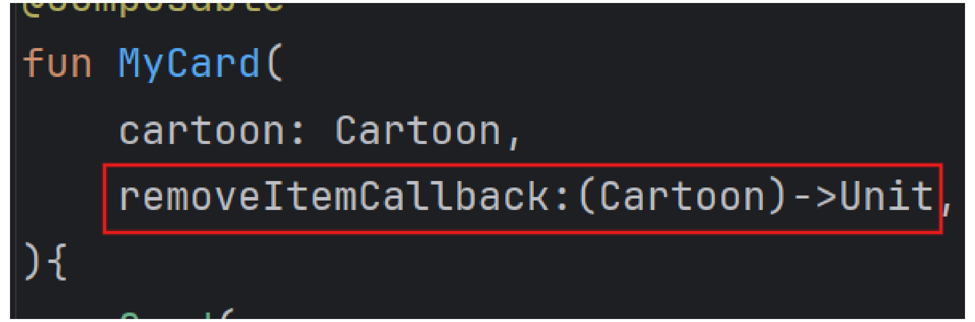
```
@Composable
fun MyCard(
    cartoon: Cartoon,
    removeItemCallback:(Cartoon)->Unit,
){
    Card(
```



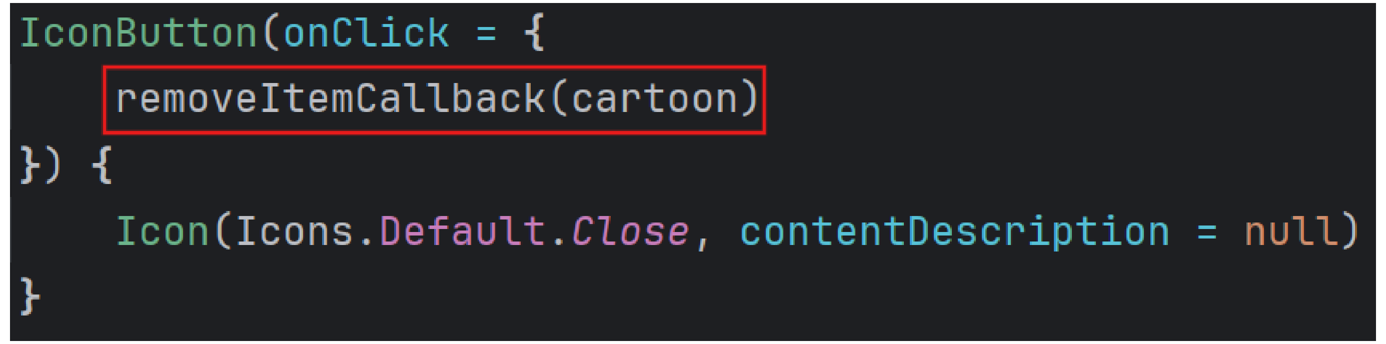
```
IconButton(onClick = {
    removeItemCallback(cartoon)
}) {
    Icon(Icons.Default.Close, contentDescription = null)
}
```

Callback（续）

- 在这里我们传递回调函数，并在需要时使用它



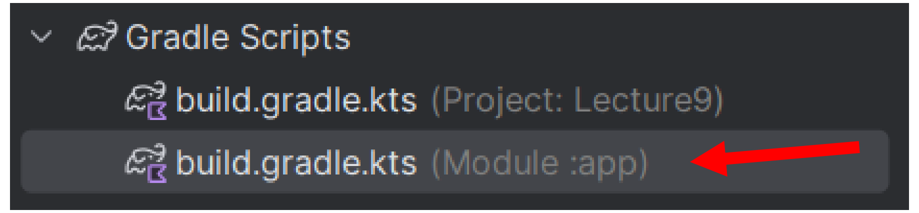
```
@Composable
fun MyCard(
    cartoon: Cartoon,
    removeItemCallback:(Cartoon)->Unit,
){
    Card(
```



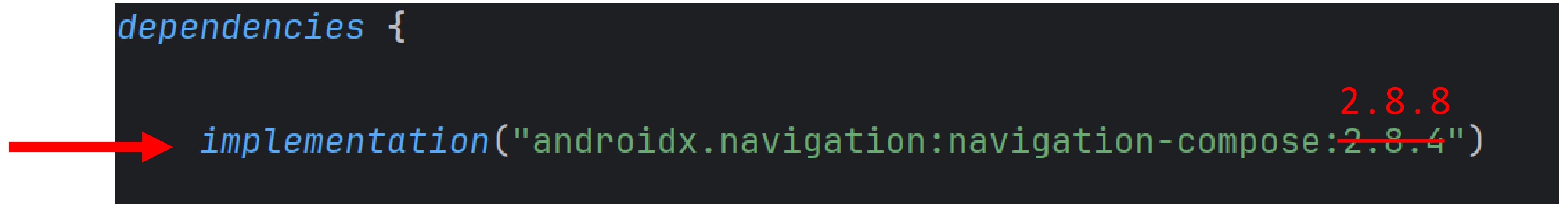
```
IconButton(onClick = {
    removeItemCallback(cartoon)
}) {
    Icon(Icons.Default.Close, contentDescription = null)
}
```

Navigation

- To navigate between two screens in our app we need to use navigation architecture
- To use navigation, we need to add the dependency to our project
  - *Double check you don’t have the dependency already*



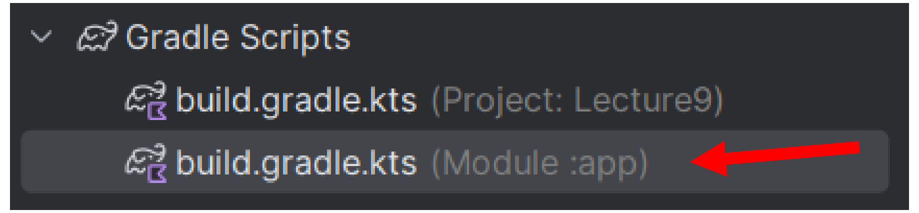
Gradle Scripts
build.gradle.kts (Project: Lecture9)
build.gradle.kts (Module :app)



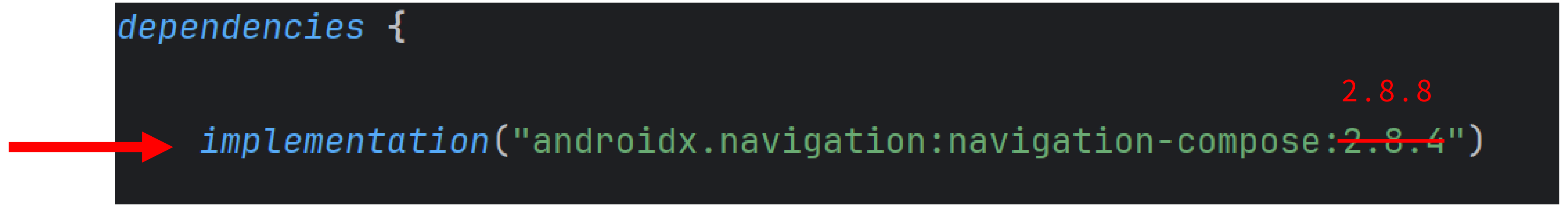
```
dependencies {
                                                             2.8.8
    implementation("androidx.navigation:navigation-compose:~~2.8.4~~")
```

导航

- 要在我们应用的两个屏幕之间进行导航，需要使用导航架构
- 要使用导航，我们需要将依赖项添加到项目中
  - *请仔细检查是否已包含该依赖项*



Gradle Scripts
build.gradle.kts (Project: Lecture9)
build.gradle.kts (Module :app)



```
dependencies {
                                                             2.8.8
    implementation("androidx.navigation:navigation-compose:~~2.8.4~~")
```

Navigation (cont.)

- Advantages
  - The biggest advantage is that our composables are managed on the back stack
  - Provides structure when navigating between composables
  - Easily add transitions between composables

- Three important components needed
  - Navigation controller
  - Navigation host
  - Navigation graph

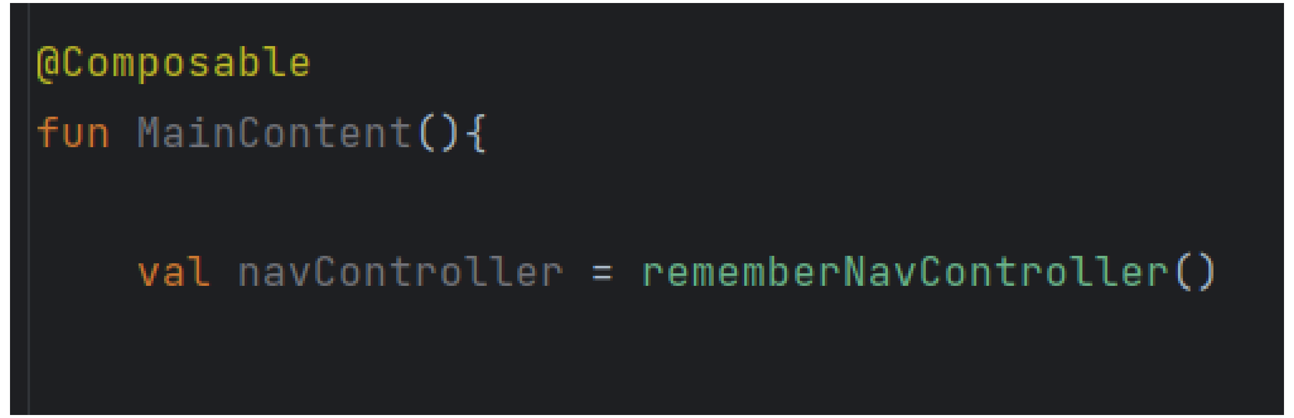
导航（续）

- 优点
  - 最大的优点是我们的可组合项由返回栈进行管理
  - 在可组合项之间导航时提供结构
  - 可轻松在可组合项之间添加过渡效果

- 需要三个重要组件
  - 导航控制器
  - 导航宿主
  - 导航图

Navigation controller

- To navigate to a destination, we use a *NavController*
  - Destinations: different content areas in your app

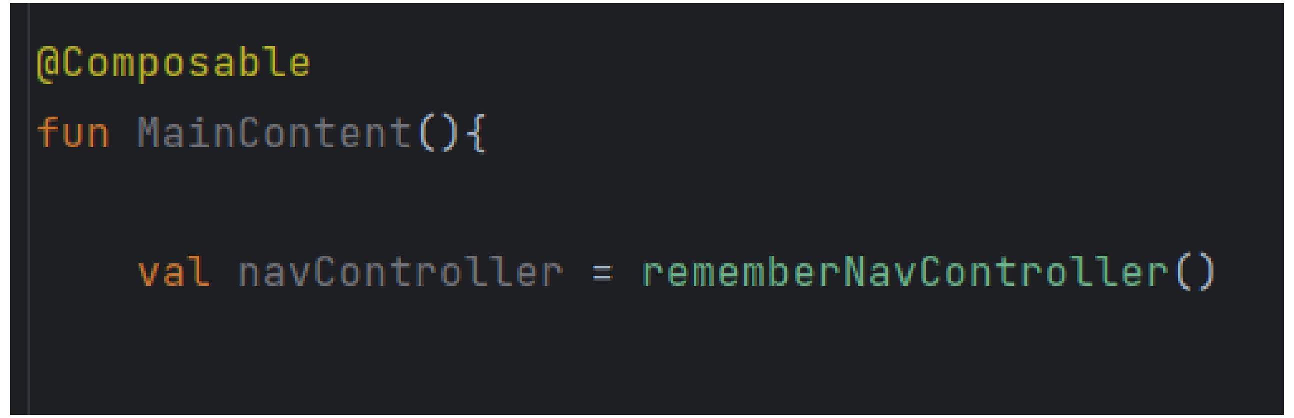


```
@Composable
fun MainContent(){

    val navController = rememberNavController()
```

- *rememberNavController* is a composable that allows us to create a *NavController* with navigation functionality

导航控制器

- 要导航到目标位置，我们使用 *NavController*
  - 目标位置：应用中不同的内容区域



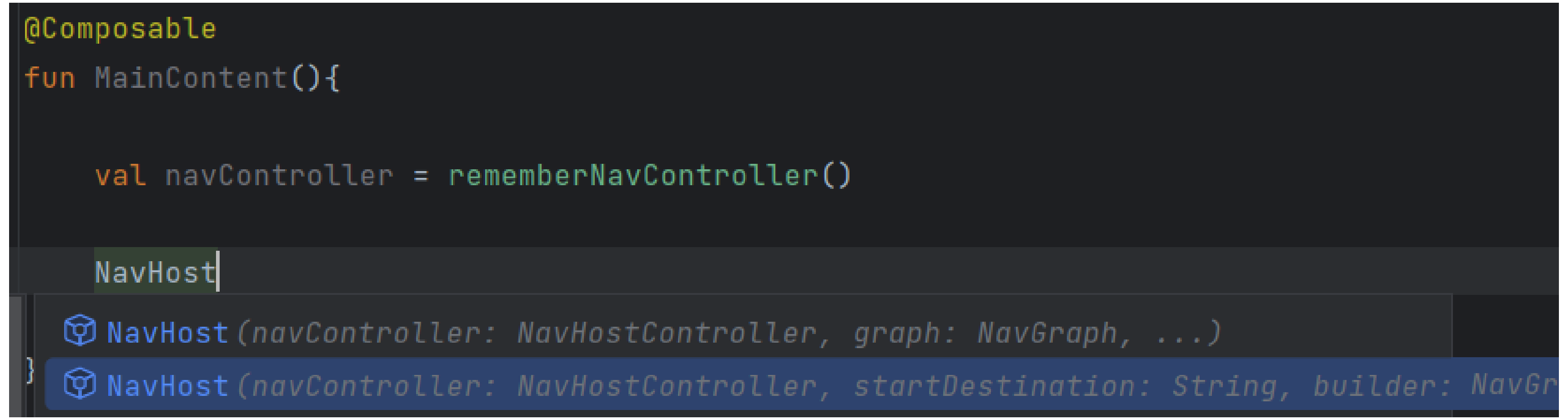
```
@Composable
fun MainContent(){

    val navController = rememberNavController()
```

- *rememberNavController* 是一个可组合项，允许我们创建具有导航功能的 *NavController*

Navigation host

- A navigation host is a single container where destinations are swapped in and out

- A navigation host is derived from a *NavHost* composable

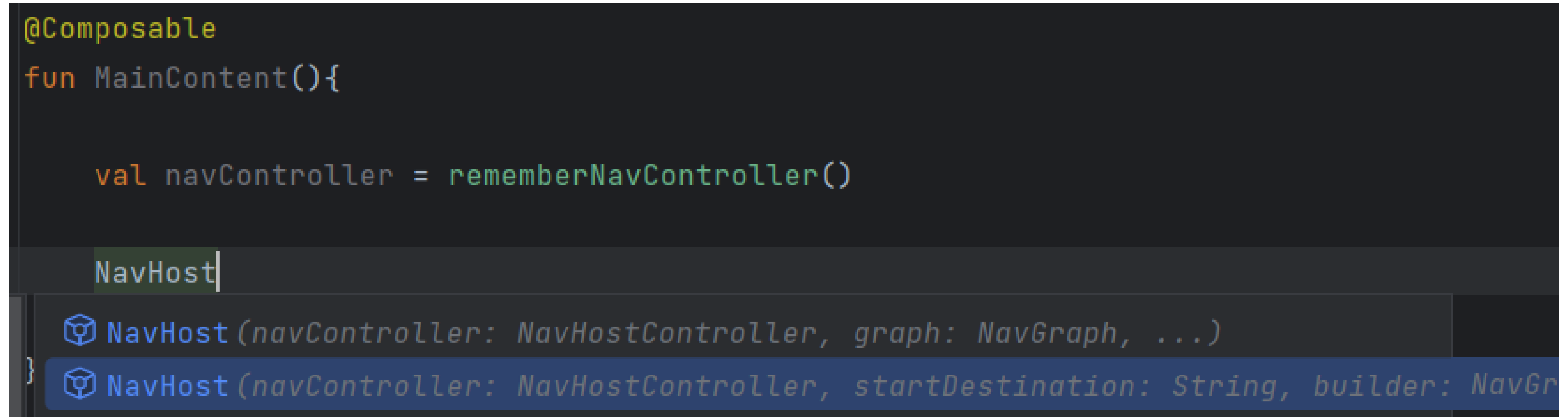


```
@Composable
fun MainContent(){

    val navController = rememberNavController()

    NavHost
   NavHost(navController: NavHostController, graph: NavGraph, ...)
}  NavHost(navController: NavHostController, startDestination: String, builder: NavGr
```

导航宿主

- 导航宿主是一个单一容器，用于切换目的地

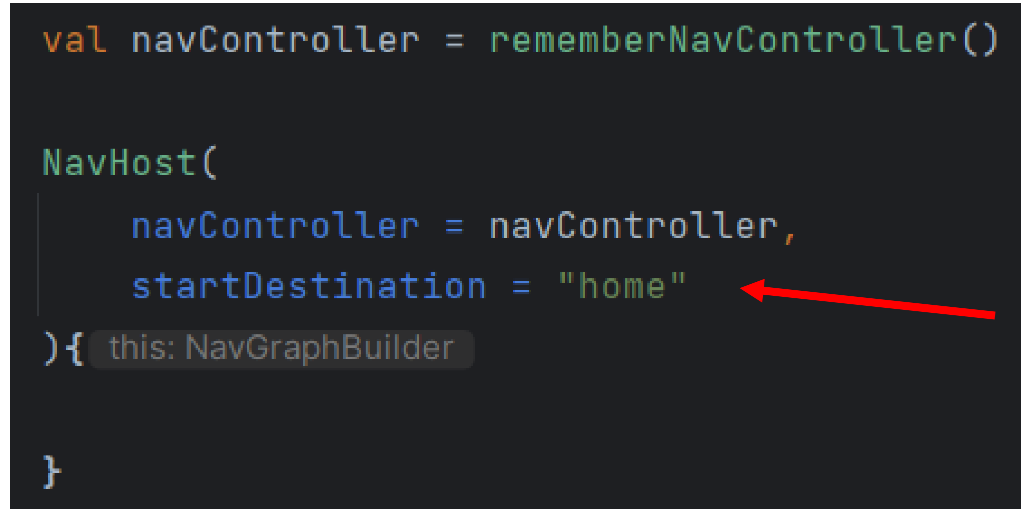
- 导航宿主派生自 *NavHost* 可组合项



```
@Composable
fun MainContent(){

    val navController = rememberNavController()

    NavHost
   NavHost(navController: NavHostController, graph: NavGraph, ...)
}  NavHost(navController: NavHostController, startDestination: String, builder: NavGr
```

Navigation host (cont.)

- The NavHost first links your navigation controller with a starting destination

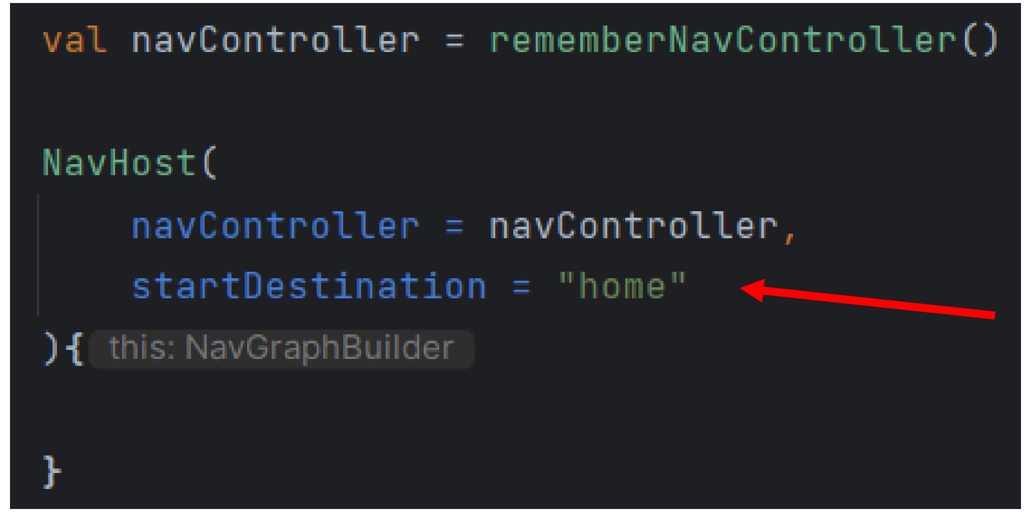


```
val navController = rememberNavController()

NavHost(
    navController = navController,
    startDestination = "home"
){ this: NavGraphBuilder

}
```

- Each destination is associated with a *route*
  - *route:* a string that defines the path to your composable

导航宿主（续）

- NavHost 首先将您的导航控制器与起始目的地关联起来



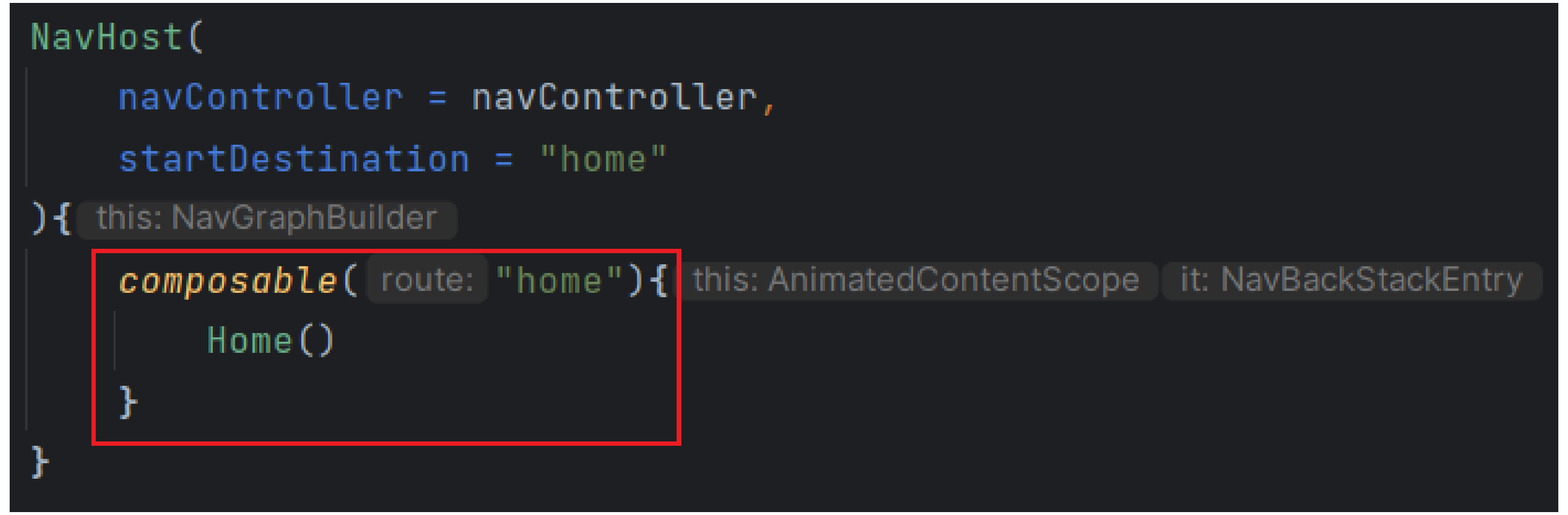
```
val navController = rememberNavController()

NavHost(
    navController = navController,
    startDestination = "home"
){ this: NavGraphBuilder

}
```

- 每个目的地都与一个 路由
  - 路由： 一个定义可组合项路径的字符串

Navigation Graph

- The NavHost also links your navigation controller to your navigation graph

- A navigation graph specifies the destinations that you can navigate between

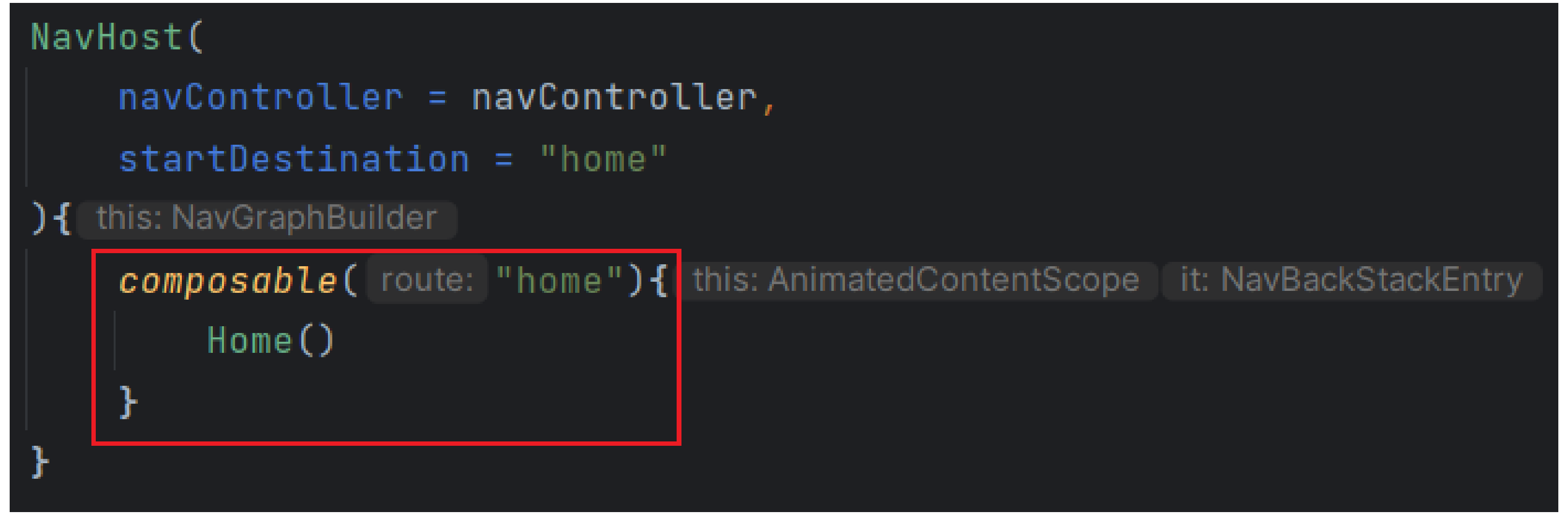


```
NavHost(
    navController = navController,
    startDestination = "home"
){ this: NavGraphBuilder
    composable( route: "home"){ this: AnimatedContentScope  it: NavBackStackEntry
        Home()
    }
}
```

导航图

- NavHost 还将您的导航控制器链接到导航图

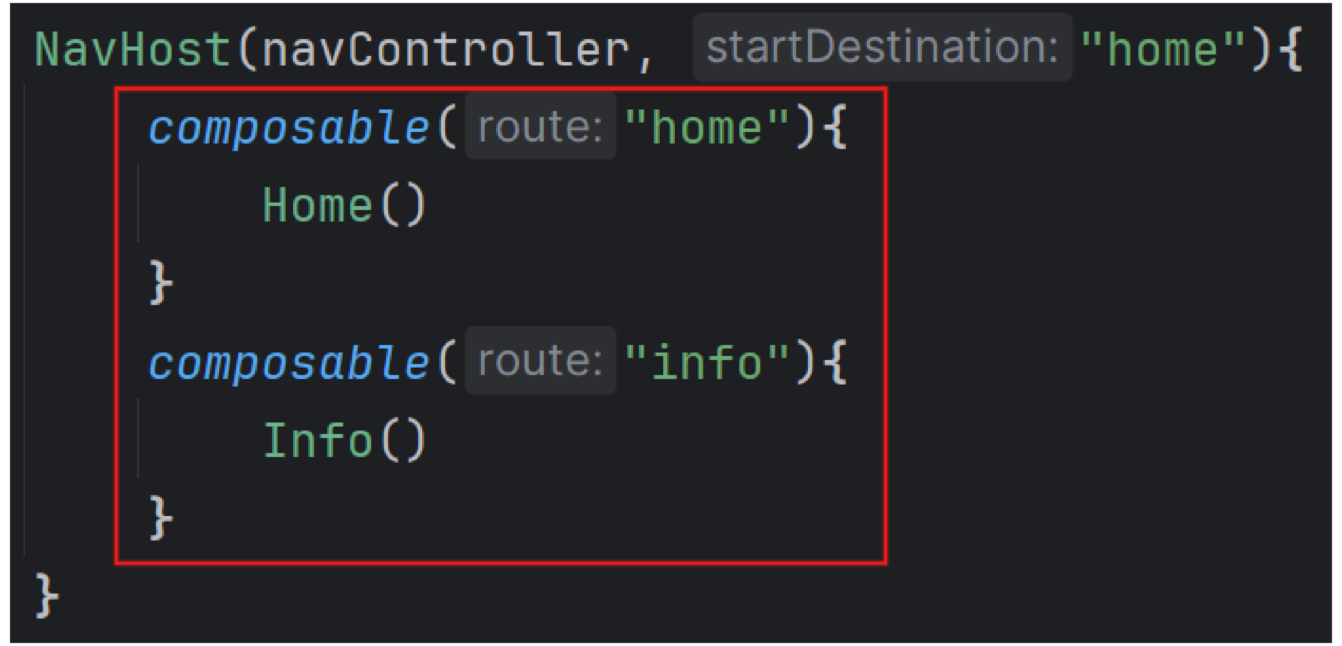
- 导航图指定您可以进行导航的 目标位置。



```
NavHost(
    navController = navController,
    startDestination = "home"
){ this: NavGraphBuilder
    composable( route: "home"){ this: AnimatedContentScope  it: NavBackStackEntry
        Home()
    }
}
```

Navigation Graph (cont.)

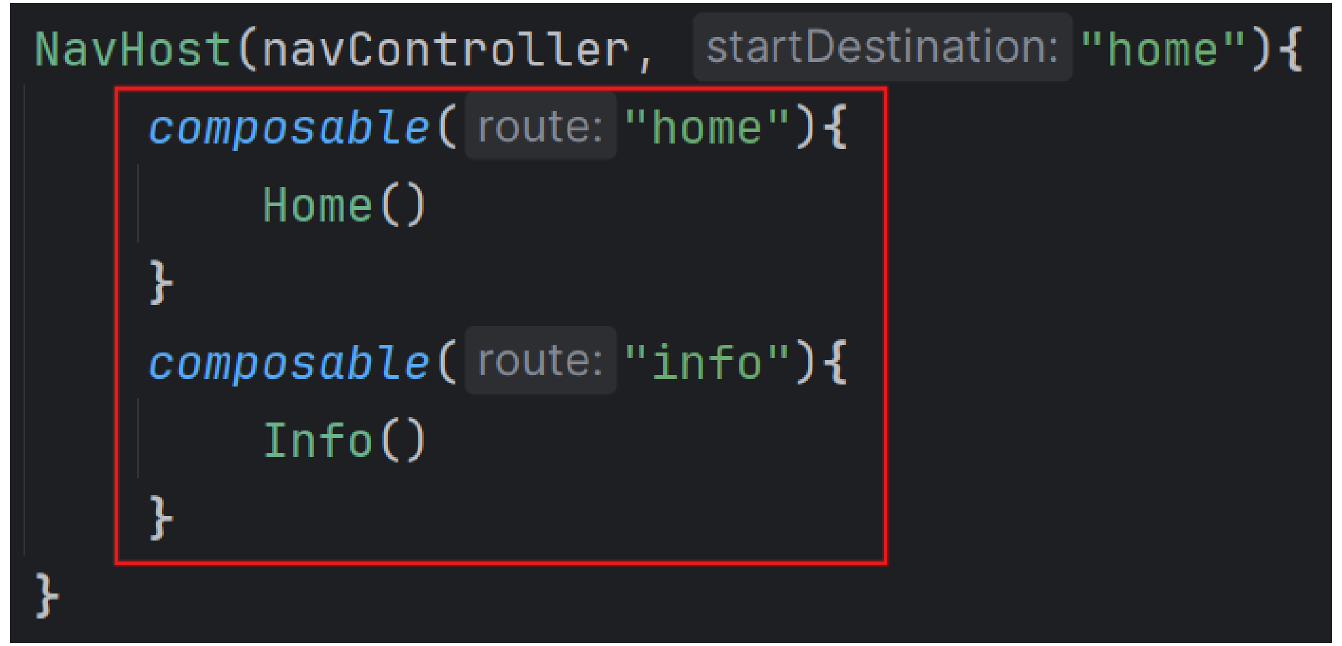
- We should add all the screens we want to navigate between here



```
NavHost(navController, startDestination: "home"){
    composable( route: "home"){
        Home()
    }
    composable( route: "info"){
        Info()
    }
}
```

导航图（续）

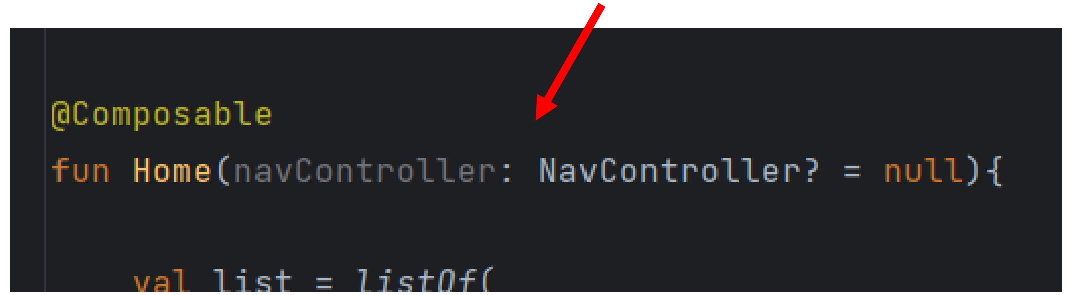
- 我们应该将需要相互导航的所有屏幕都添加到 此处



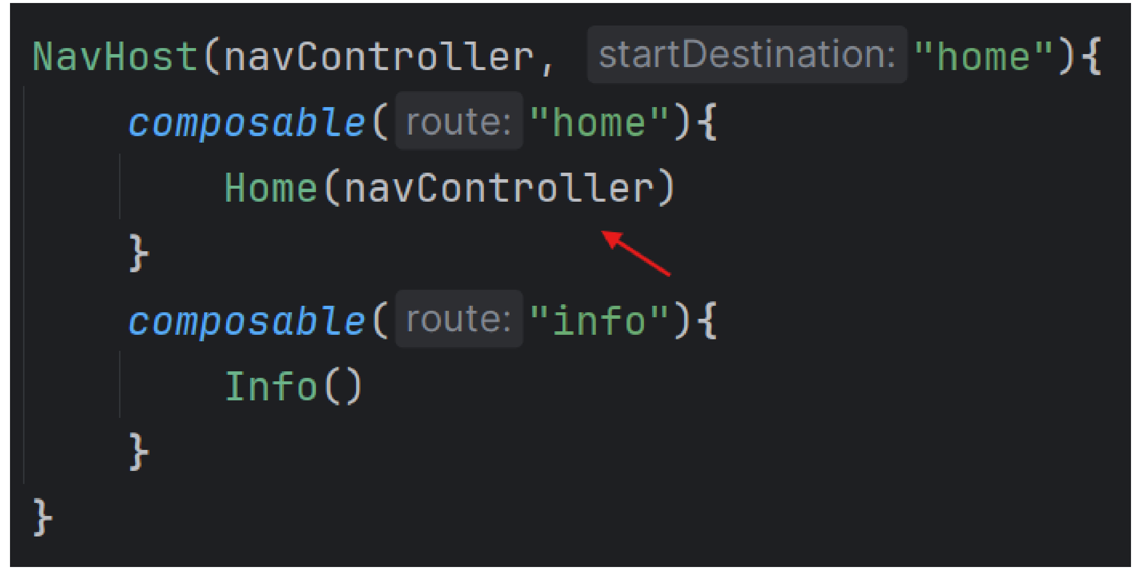
```
NavHost(navController, startDestination: "home"){
    composable( route: "home"){
        Home()
    }
    composable( route: "info"){
        Info()
    }
}
```

Navigation controller (cont.)

- To navigate from one screen to another we need a reference to the navigation controller



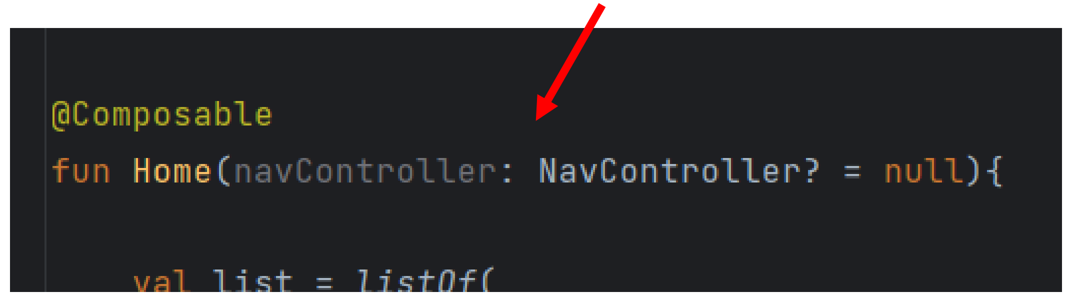
```
@Composable
fun Home(navController: NavController? = null){

    val list = listOf(
```



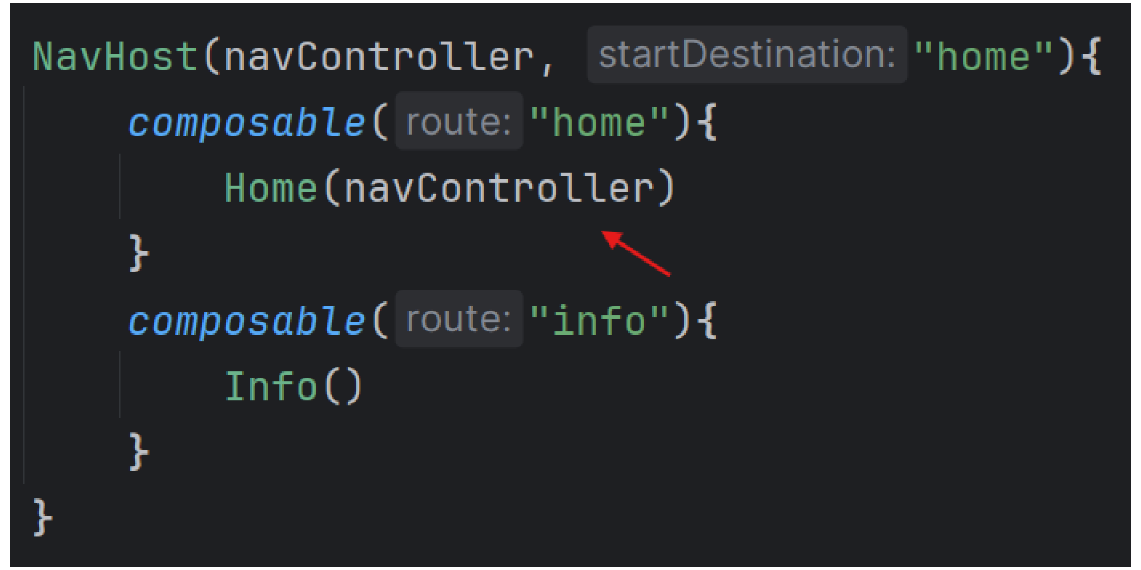
```
NavHost(navController, startDestination: "home"){
    composable( route: "home"){
        Home(navController)
    }
    composable( route: "info"){
        Info()
    }
}
```

导航控制器（续）

- 要从一个屏幕导航到另一个屏幕，我们需要一个指向导航控制器的引用



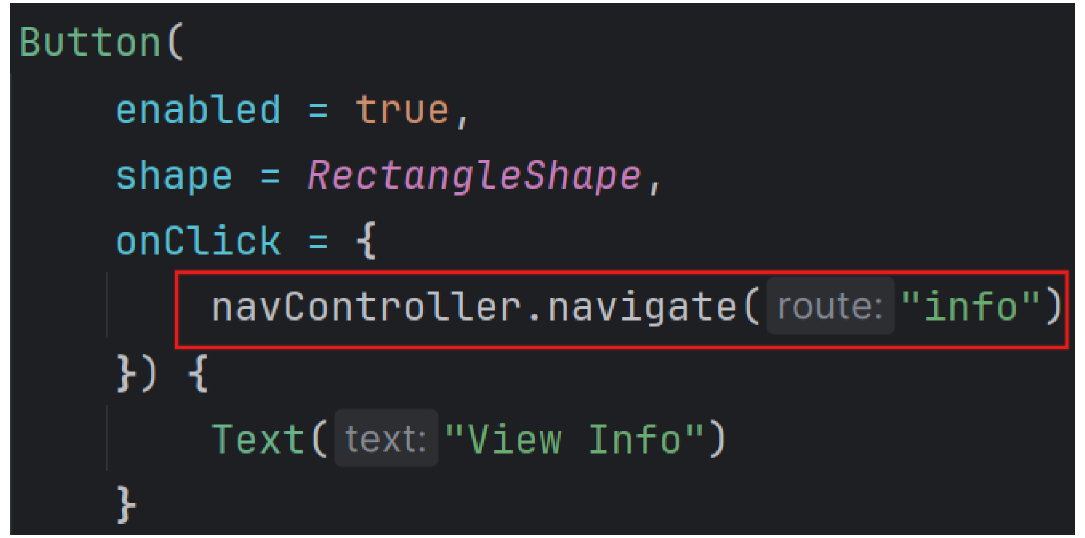
```
@Composable
fun Home(navController: NavController? = null){

    val list = listOf(
```



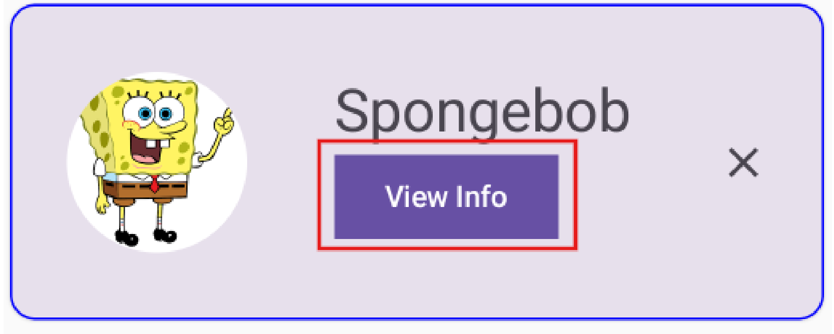
```
NavHost(navController, startDestination: "home"){
    composable( route: "home"){
        Home(navController)
    }
    composable( route: "info"){
        Info()
    }
}
```

Navigation controller (cont.)

- We then can invoke navigate on our *navController* through an event and pass in the route we want to navigate to

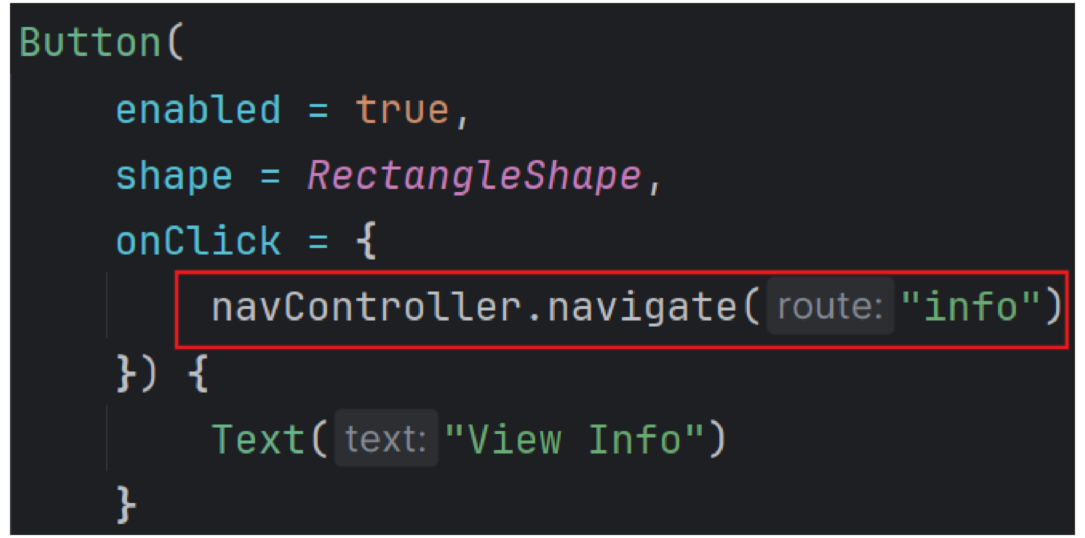


```
Button(
    enabled = true,
    shape = RectangleShape,
    onClick = {
        navController.navigate( route: "info")
    }) {
        Text( text: "View Info")
    }
```

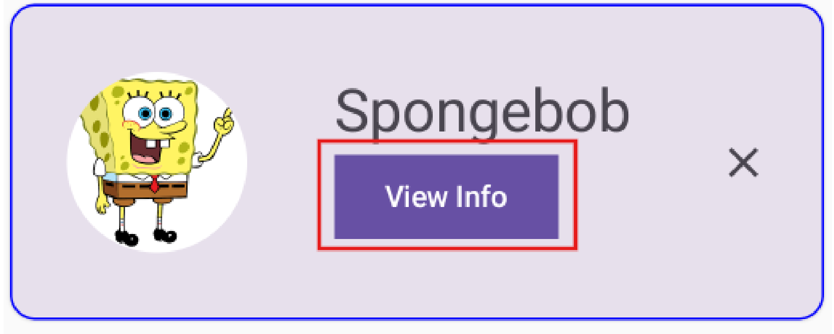


导航控制器（续）

- 然后我们可以通过一个事件调用 navigate 方法，该方法作用于我们的 *navController* 上并传入我们想要导航到的路由

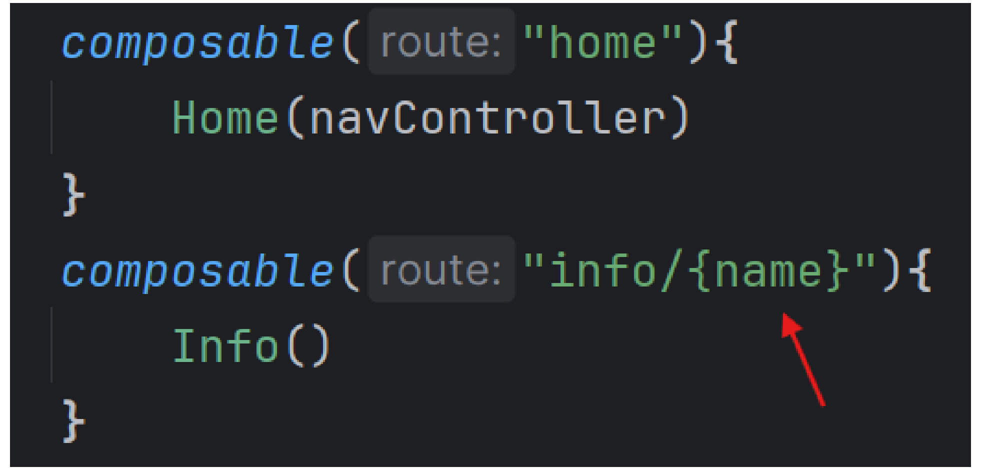


```
Button(
    enabled = true,
    shape = RectangleShape,
    onClick = {
        navController.navigate( route: "info")
    }) {
        Text( text: "View Info")
    }
```



Pass argument with navigation

- To pass an argument between destinations we need to change the *route*

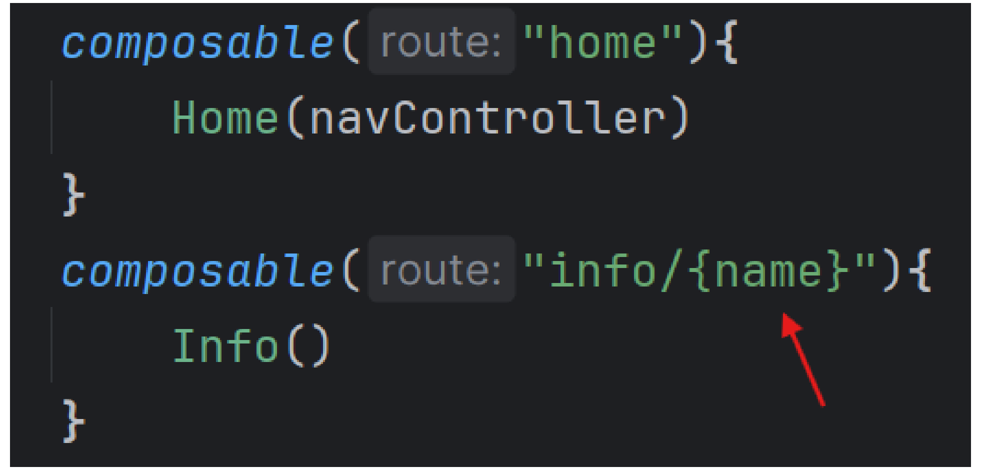


```
composable( route: "home"){
    Home(navController)
}
composable( route: "info/{name}"){
    Info()
}
```

- We add */{key}* after the destination name

传递 带导航的参数

- 要在目的地之间传递参数，我们需要更改 路由

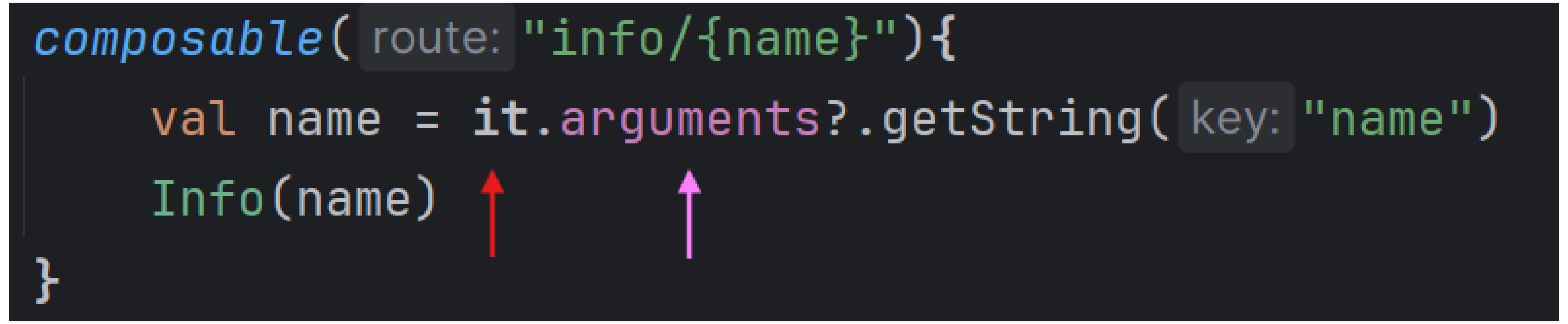


```
composable( route: "home"){
    Home(navController)
}
composable( route: "info/{name}"){
    Info()
}
```

- 我们在目的地名称后添加 */{key}* 即可

Pass argument with navigation (cont.)

- We access the *arguments* as a *Bundle* through the *NavBackStackEntry*

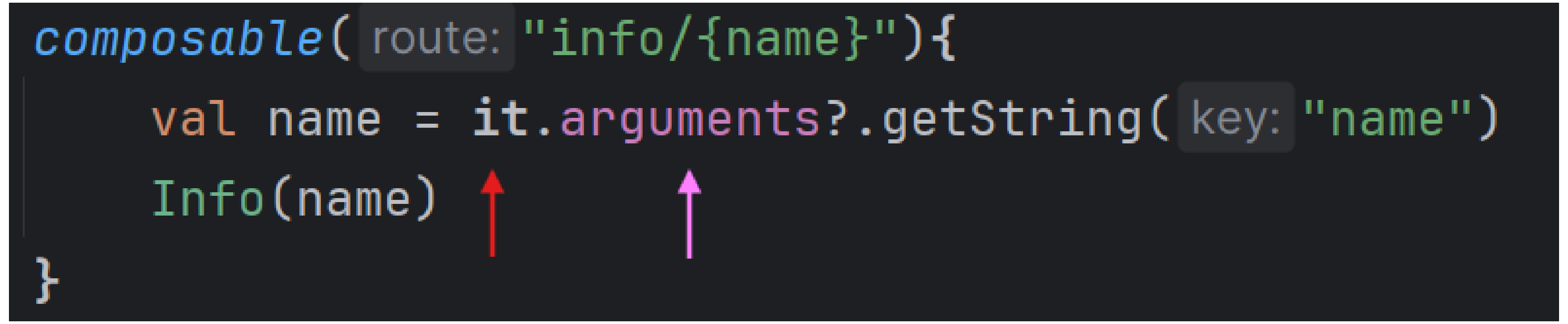


```
composable( route: "info/{name}"){
    val name = it.arguments?.getString( key: "name")
    Info(name)
}
```

- The *NavBackStackEntry* represents the previous destination
  - In our example it will be *Home*

传递 带导航的参数（续）

- 我们通过 *参数* 以 *Bundle* 的形式从 *NavBackStackEntry* 访问

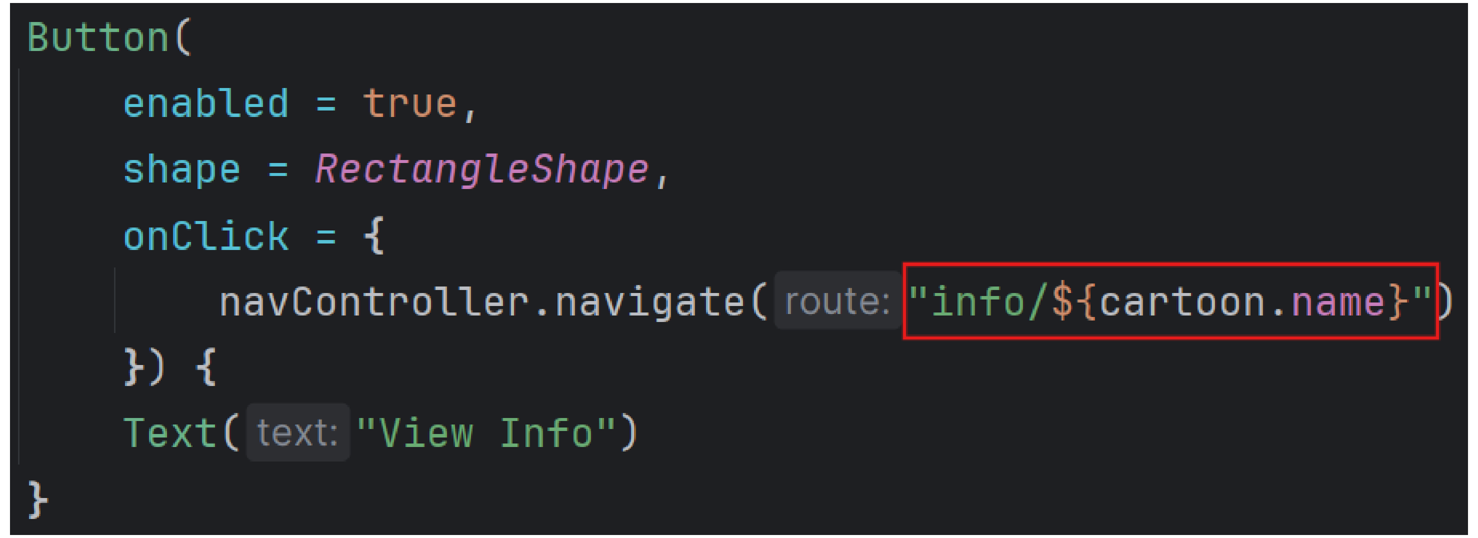


```
composable( route: "info/{name}"){
    val name = it.arguments?.getString( key: "name")
    Info(name)
}
```

- *NavBackStackEntry* 表示前一个目的地
  - 在我们的示例中，它将是 首页

Pass argument with navigation (cont.)

- Then we provide the argument we want to pass when we navigate

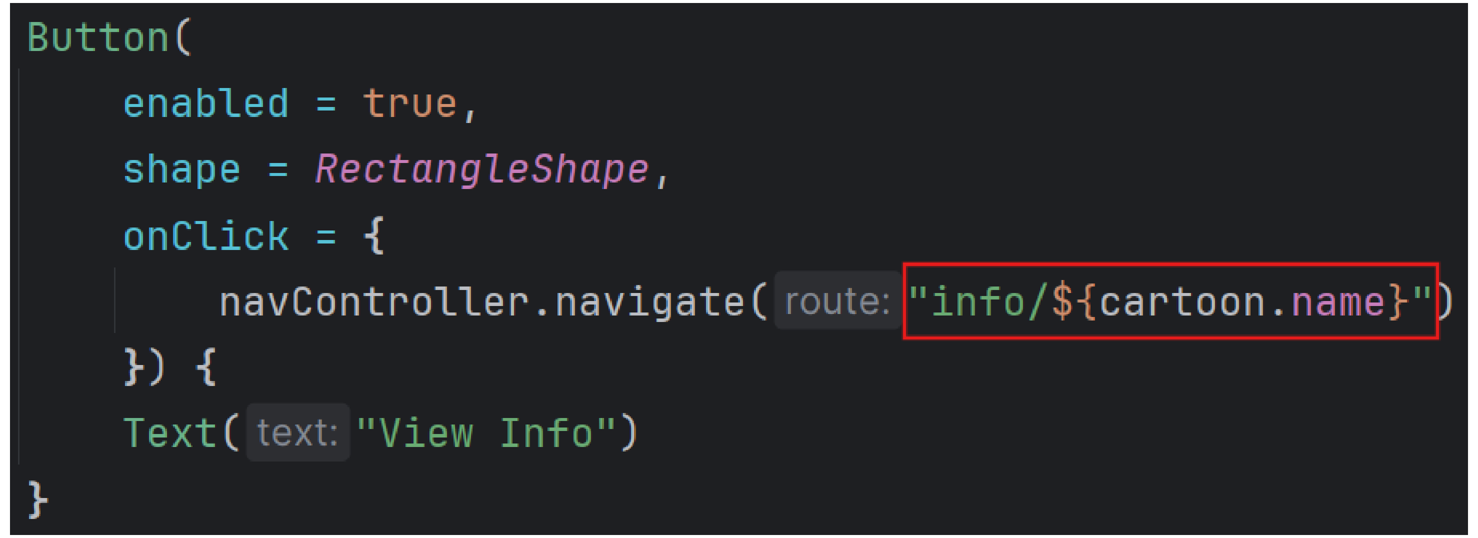


```
Button(
    enabled = true,
    shape = RectangleShape,
    onClick = {
        navController.navigate( route: "info/${cartoon.name}")
    }) {
    Text( text: "View Info")
}
```

- It should match the route, but using the actual argument

传递 通过导航传递参数（续）

- 然后，在导航时提供我们想要传递的参数

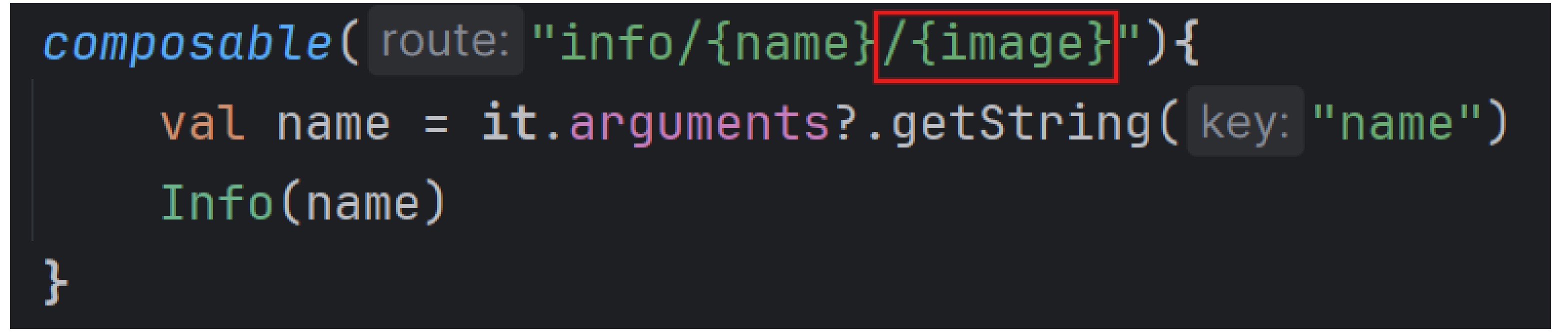


```
Button(
    enabled = true,
    shape = RectangleShape,
    onClick = {
        navController.navigate( route: "info/${cartoon.name}")
    }) {
    Text( text: "View Info")
}
```

- 它应与路由匹配，但需使用 实际参数

Pass multiple arguments with navigation

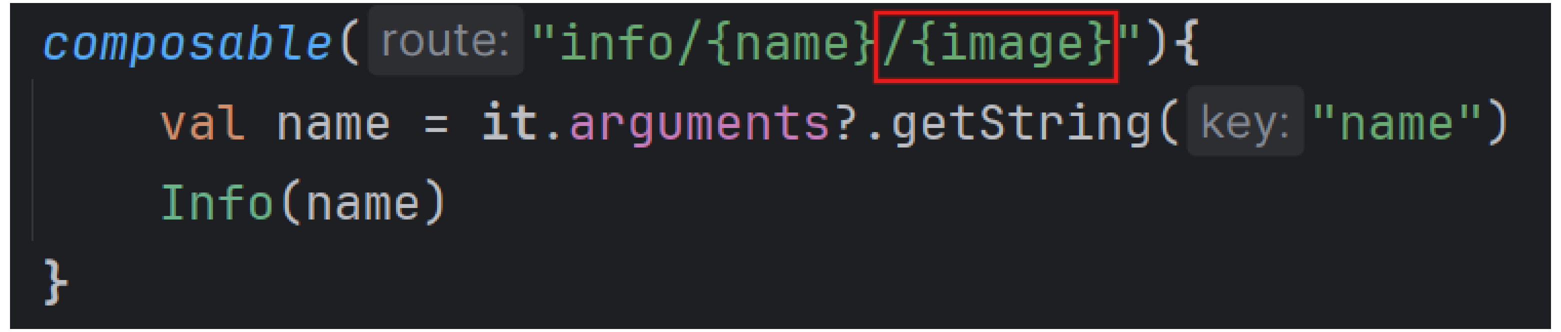
- To pass a second argument we just add another /{key} to the route



```
composable( route: "info/{name}/{image}"){
    val name = it.arguments?.getString( key: "name")
    Info(name)
}
```

传递 通过导航传递多个参数

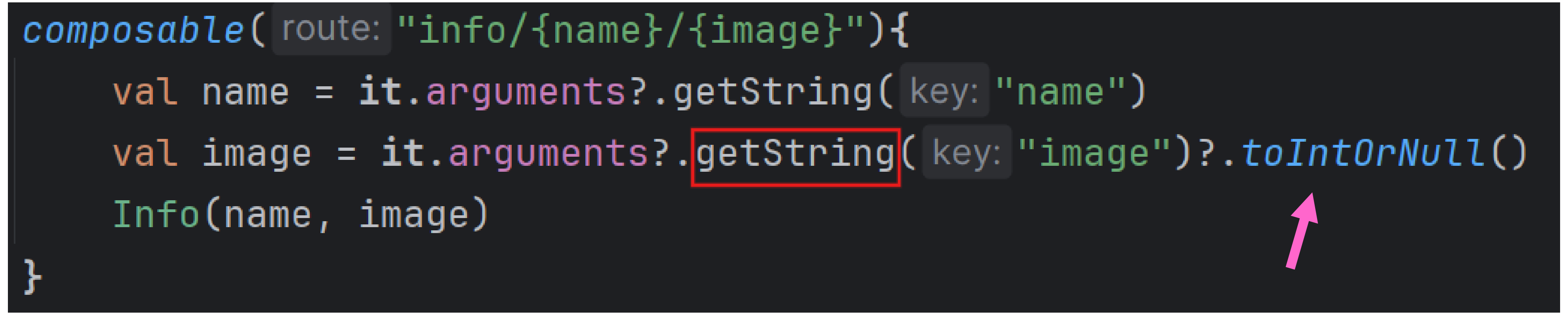
- 要传递第二个参数，我们只需在路由中添加另一个 /{key} 即可



```
composable( route: "info/{name}/{image}"){
    val name = it.arguments?.getString( key: "name")
    Info(name)
}
```

Pass multiple arguments with navigation (cont.)

- By default, all arguments are managed as strings

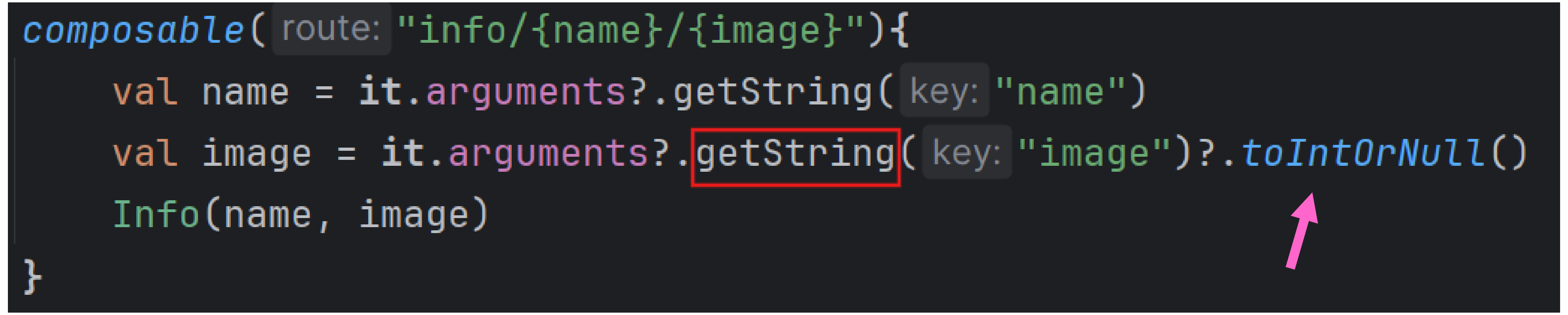


```
composable( route: "info/{name}/{image}"){
    val name = it.arguments?.getString( key: "name")
    val image = it.arguments?.getString( key: "image")?.toIntOrNull()
    Info(name, image)
}
```

- If we want to pass a different type, we need to handle it appropriately

传递 带导航的多个参数（续）

- 默认情况下，所有参数都作为 字符串 进行管理

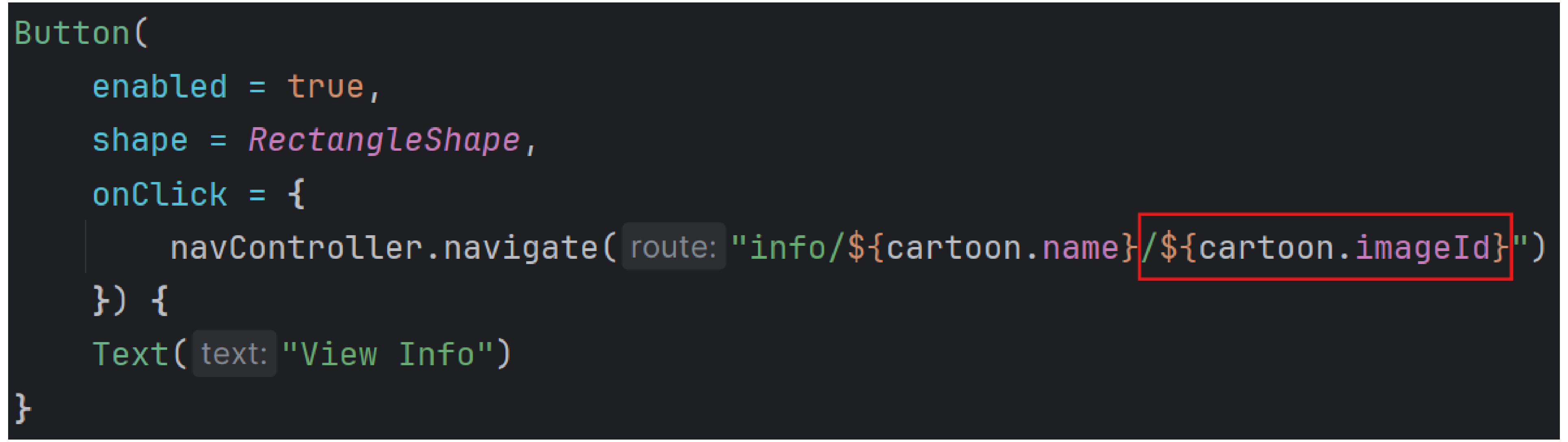


```
composable( route: "info/{name}/{image}"){
    val name = it.arguments?.getString( key: "name")
    val image = it.arguments?.getString( key: "image")?.toIntOrNull()
    Info(name, image)
}
```

- 如果想要传递不同的类型，我们需要 适当地处理它

Pass multiple arguments with navigation (cont.)

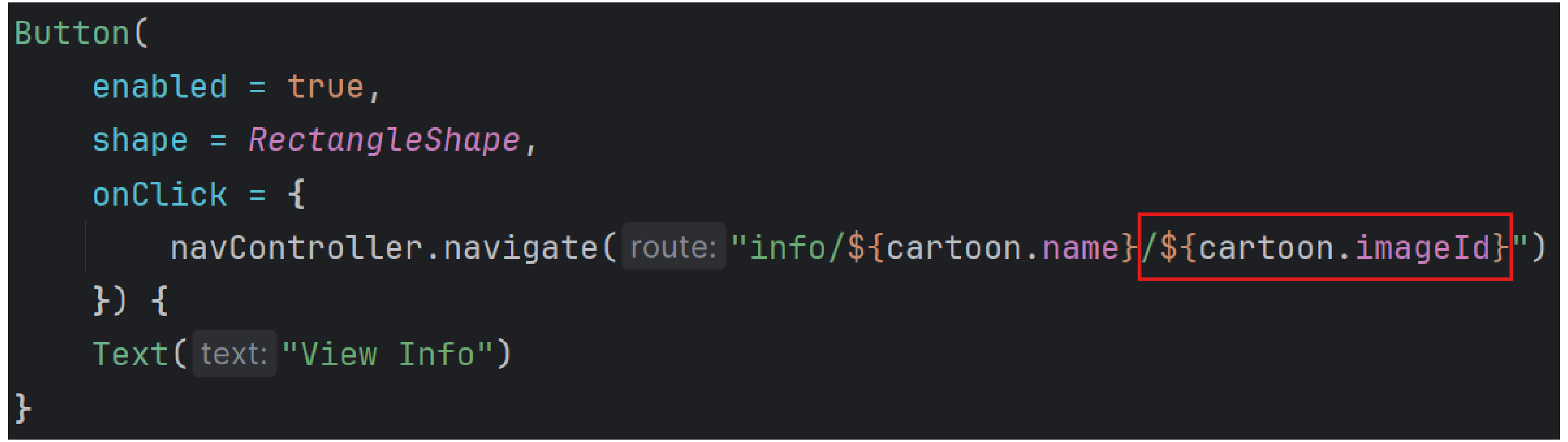
- We would again need to provide the arguments that we want to pass



```
Button(
    enabled = true,
    shape = RectangleShape,
    onClick = {
        navController.navigate( route: "info/${cartoon.name}/${cartoon.imageId}")
    }) {
    Text( text: "View Info")
}
```

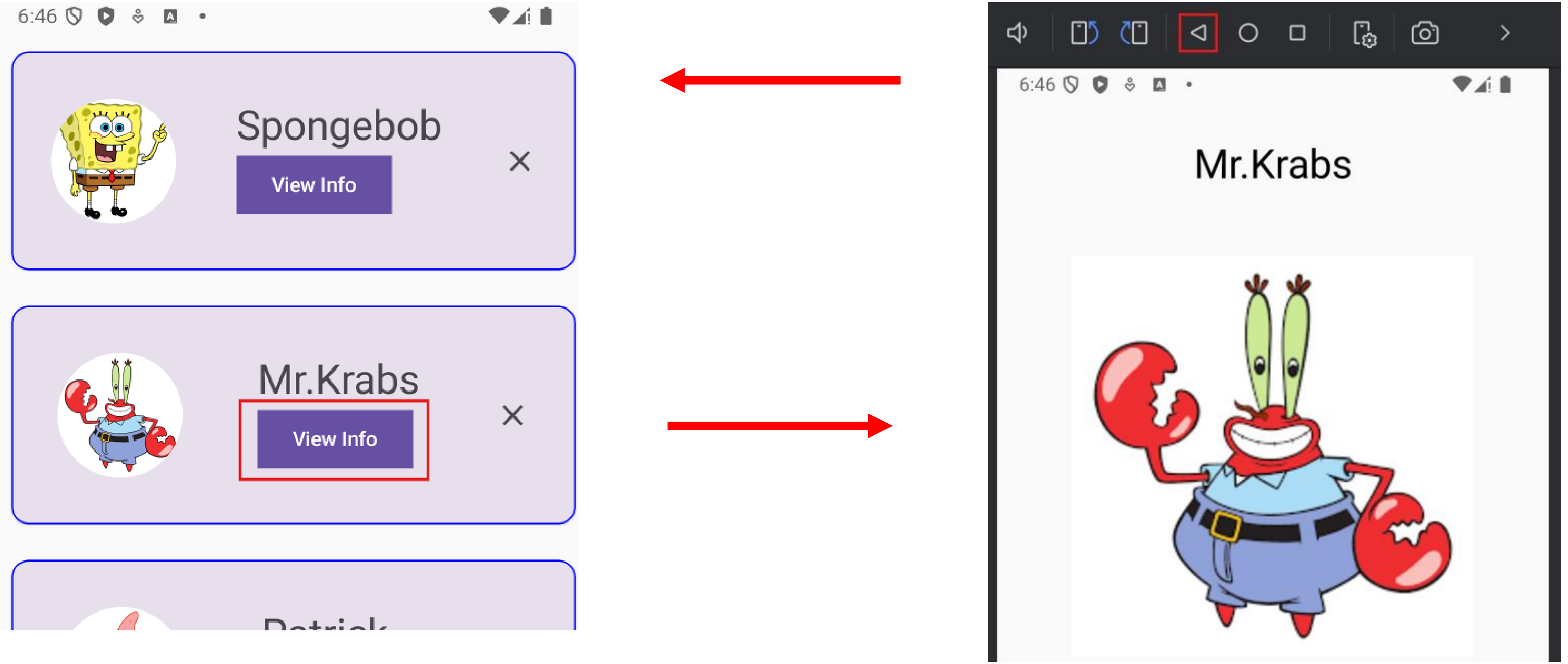
传递 使用导航传递多个参数（续）

- 我们再次需要提供想要传递的参数

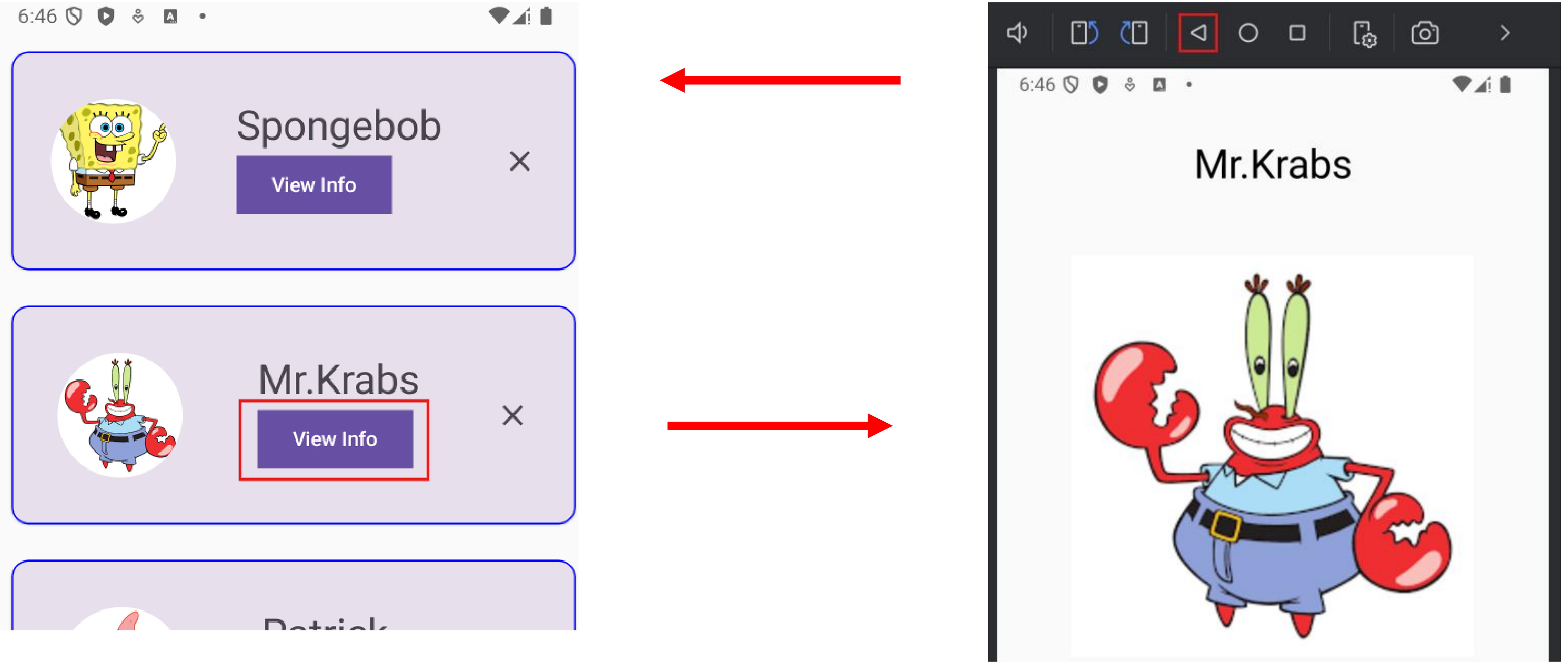


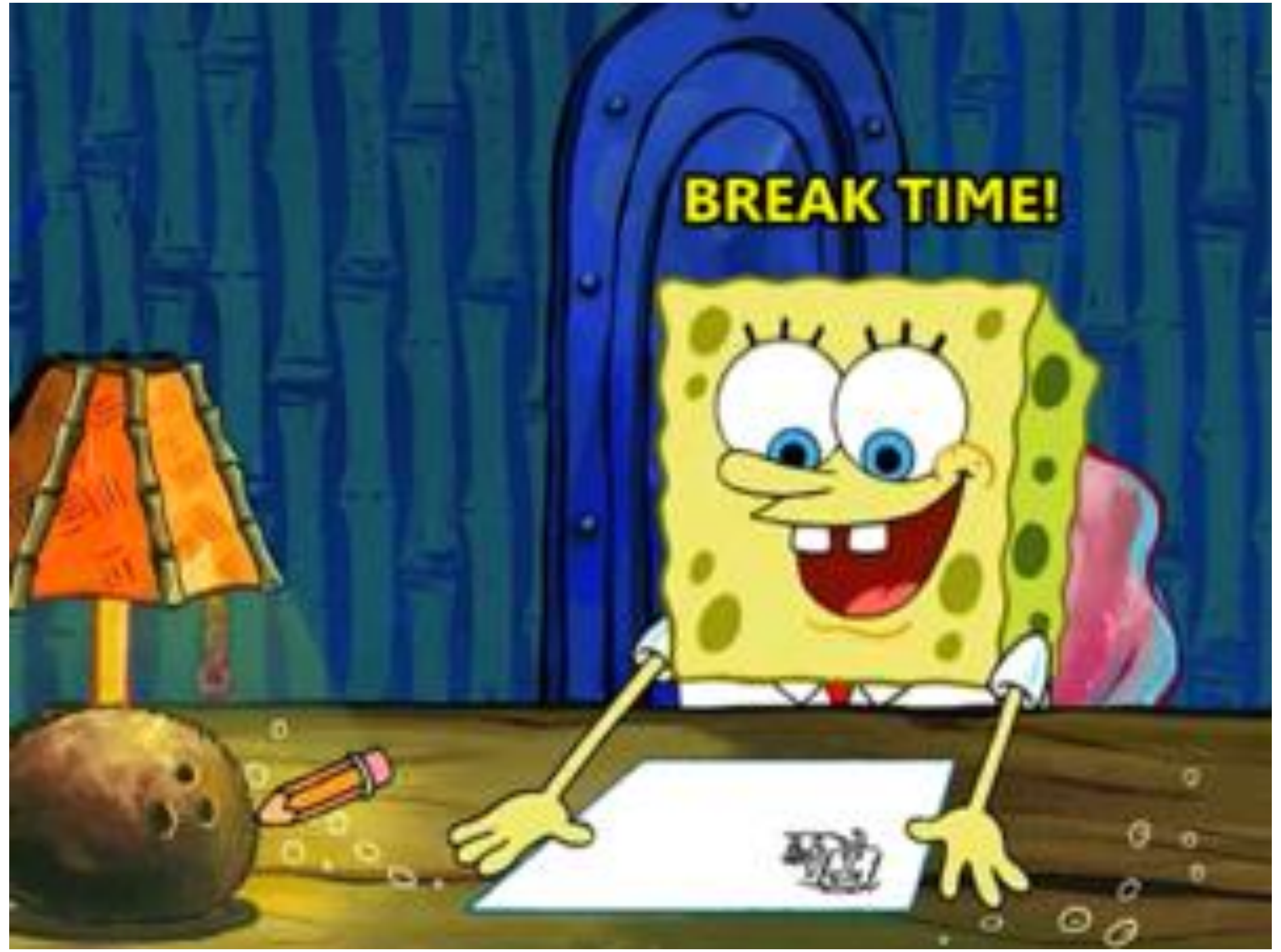
```
Button(
    enabled = true,
    shape = RectangleShape,
    onClick = {
        navController.navigate( route: "info/${cartoon.name}/${cartoon.imageId}")
    }) {
    Text( text: "View Info")
}
```

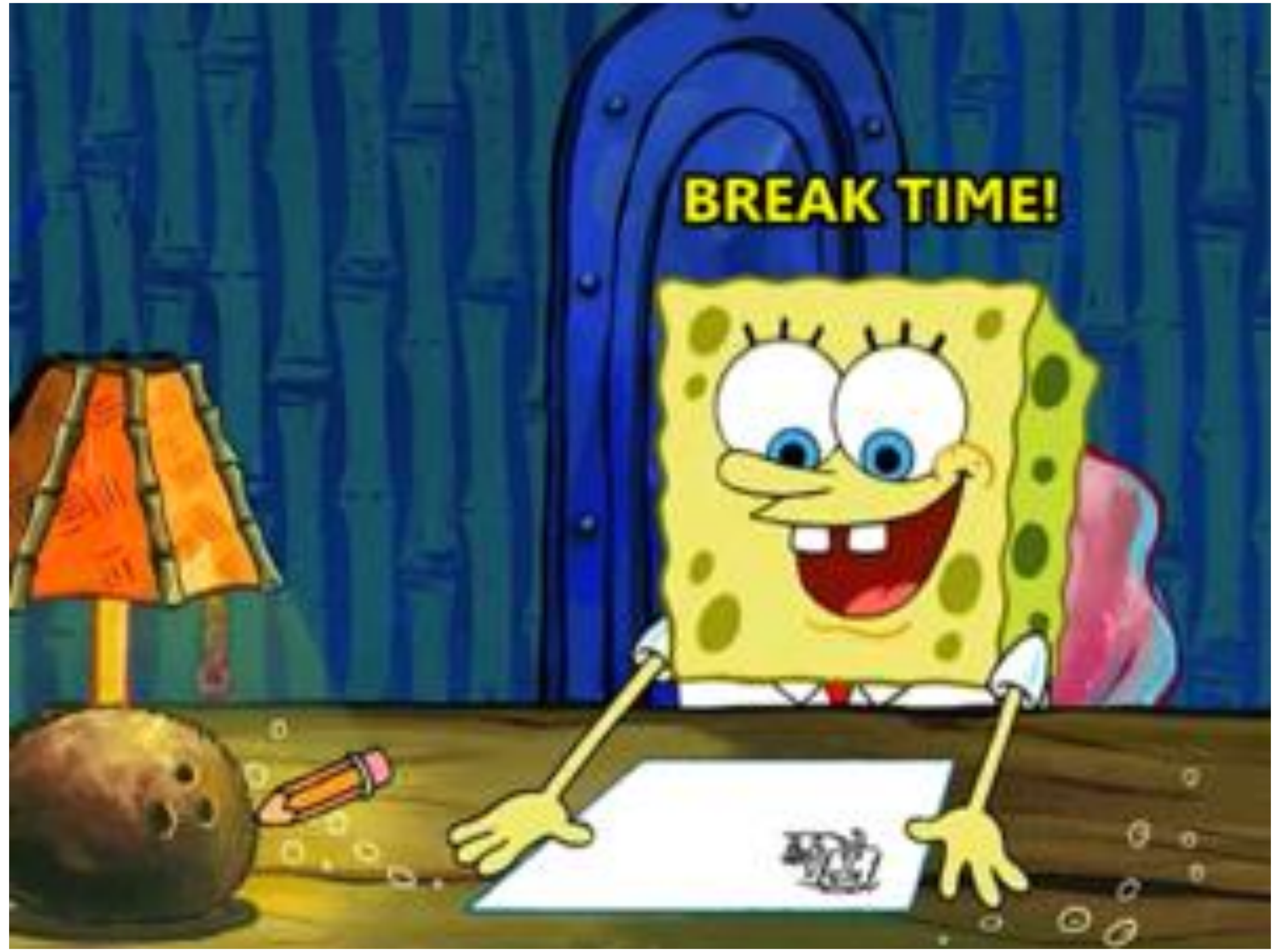
Pass multiple arguments with navigation (cont.)



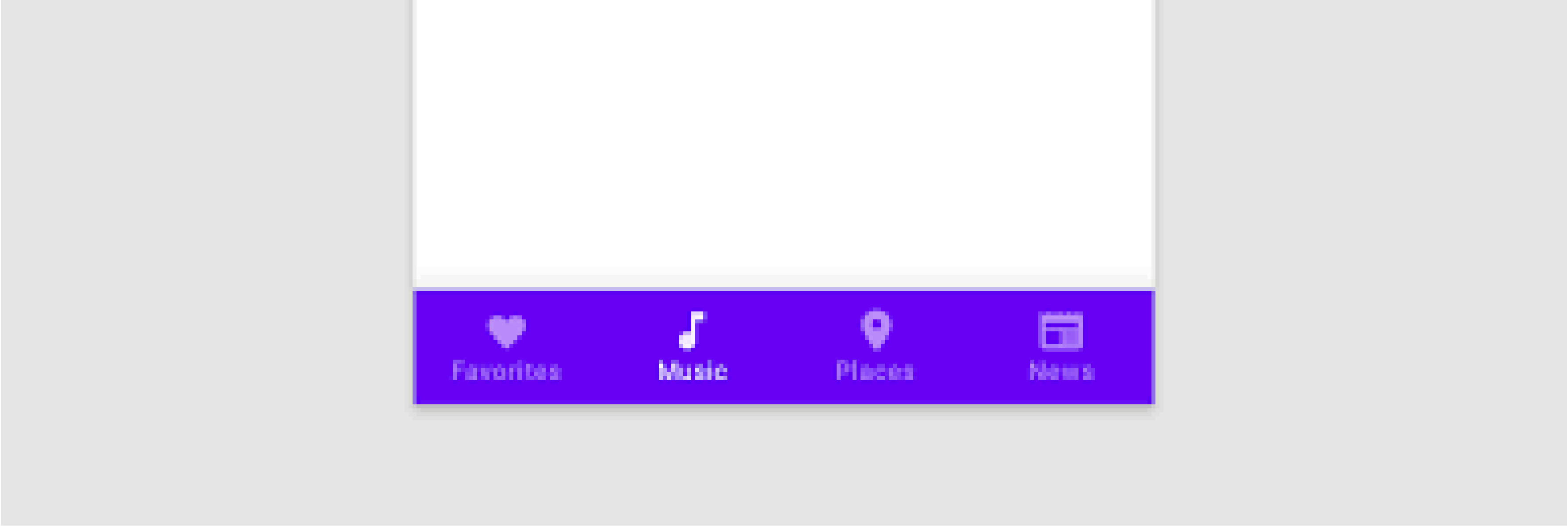
通过导航传递 多个参数（续）



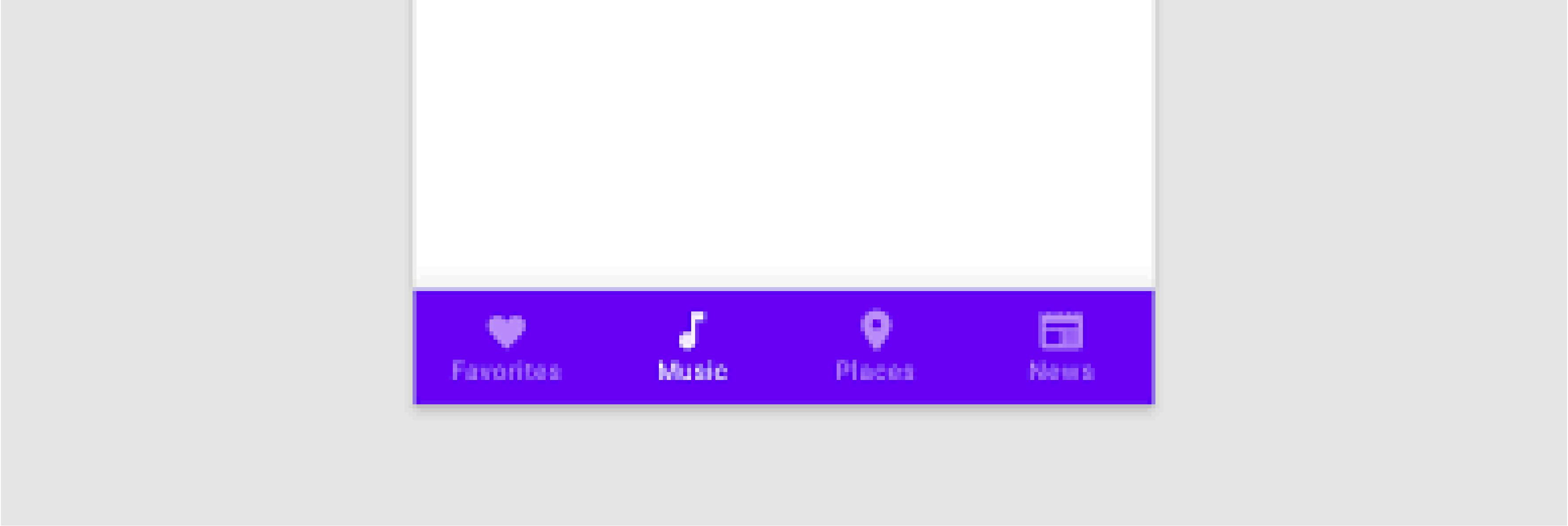

BREAK TIME!


BREAK TIME!

Bottom navigation bar

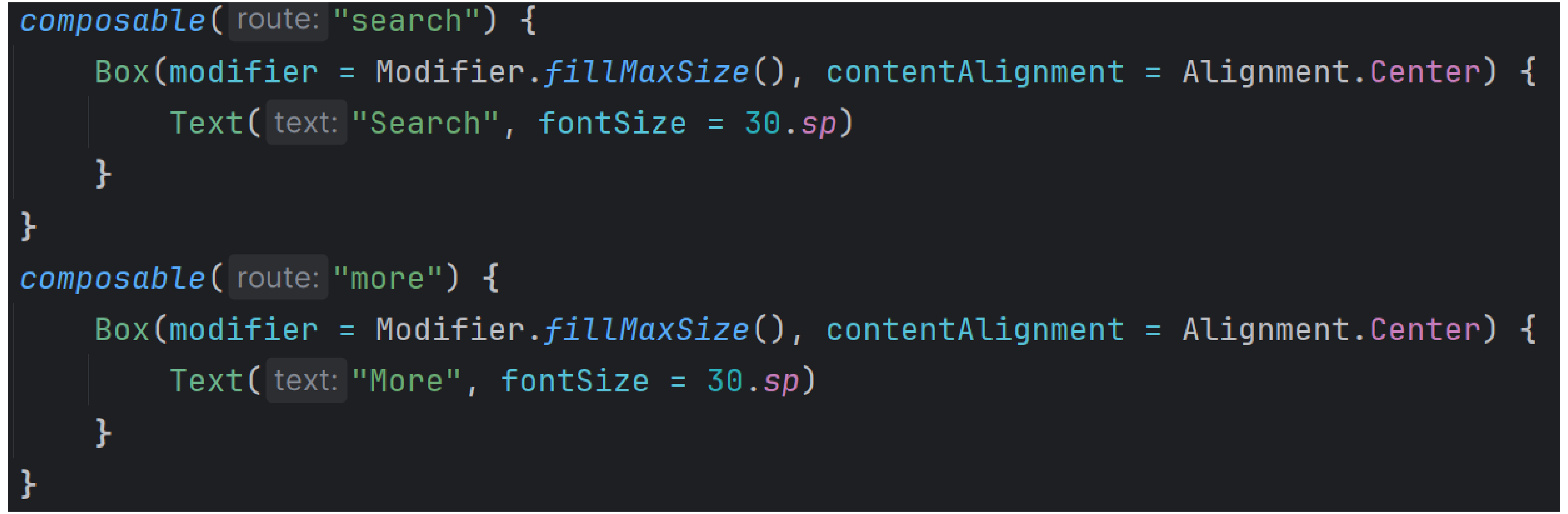


底部导航栏



Bottom navigation bar (cont.)

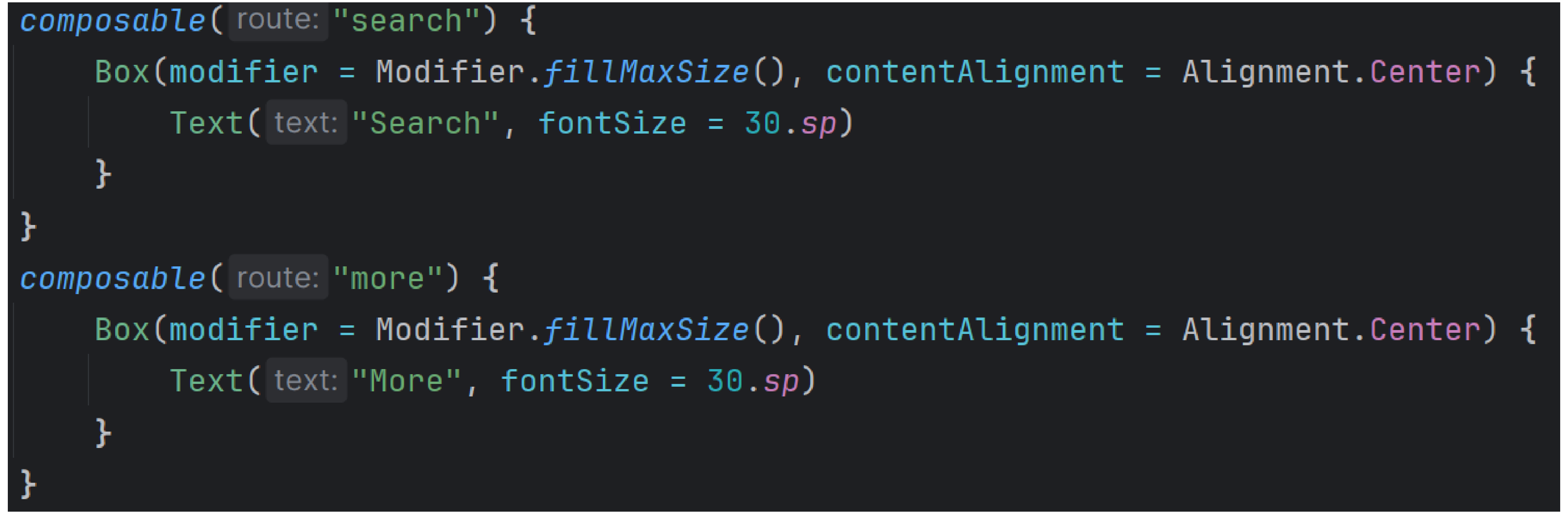
- Let's first create a couple more destinations for to use with a bottom nav bar



```
composable( route: "search") {
    Box(modifier = Modifier.fillMaxSize(), contentAlignment = Alignment.Center) {
        Text( text: "Search", fontSize = 30.sp)
    }
}
composable( route: "more") {
    Box(modifier = Modifier.fillMaxSize(), contentAlignment = Alignment.Center) {
        Text( text: "More", fontSize = 30.sp)
    }
}
```

底部导航栏（续）

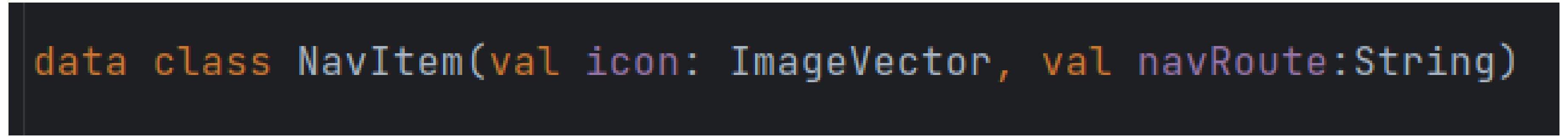
- 让我们先创建更多一些目的地，用于底部导航栏



```
composable( route: "search") {
    Box(modifier = Modifier.fillMaxSize(), contentAlignment = Alignment.Center) {
        Text( text: "Search", fontSize = 30.sp)
    }
}
composable( route: "more") {
    Box(modifier = Modifier.fillMaxSize(), contentAlignment = Alignment.Center) {
        Text( text: "More", fontSize = 30.sp)
    }
}
```

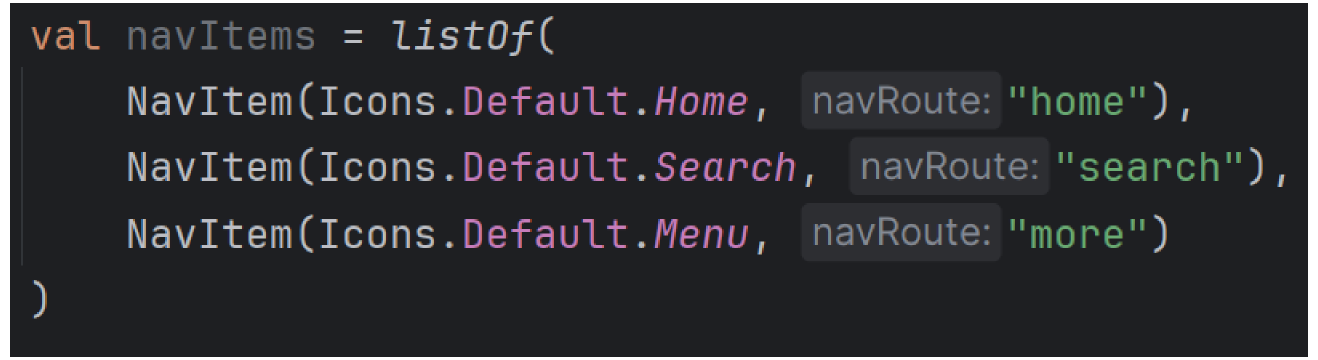
Bottom navigation bar (cont.)

- We then need to create the data for our navigation items
  - You could also add a title for the items if you want



```
data class NavItem(val icon: ImageVector, val navRoute:String)
```

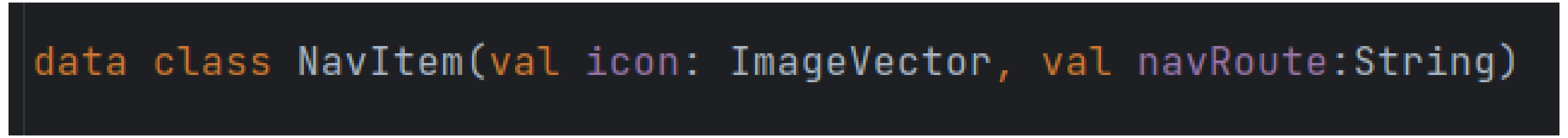
- Next, how many items do we want in our navigation bar?



```
val navItems = listOf(
    NavItem(Icons.Default.Home, navRoute: "home"),
    NavItem(Icons.Default.Search, navRoute: "search"),
    NavItem(Icons.Default.Menu, navRoute: "more")
)
```

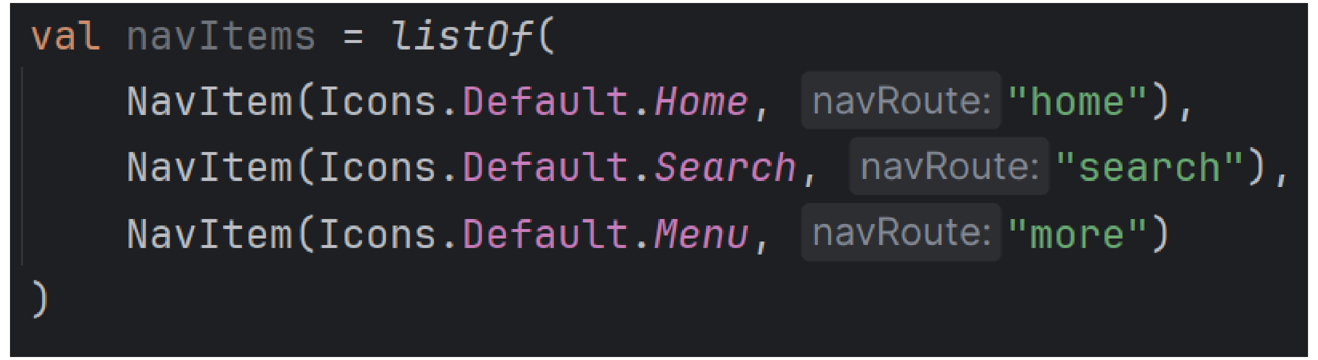
底部导航栏（续）

- 接下来我们需要为导航项创建数据
  - 如果需要，你还可以为这些项添加标题



```
data class NavItem(val icon: ImageVector, val navRoute:String)
```

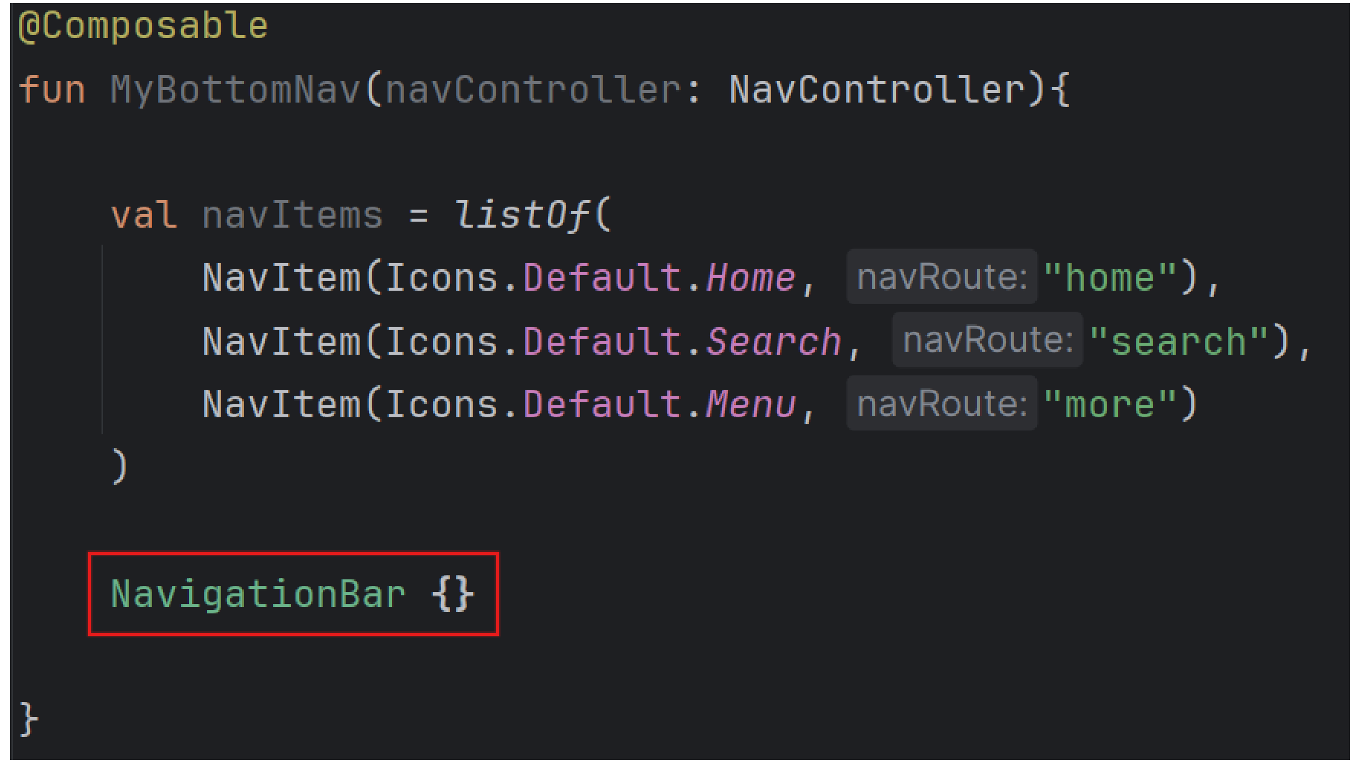
- 接下来，我们希望在导航栏中放置多少个项?



```
val navItems = listOf(
    NavItem(Icons.Default.Home, navRoute: "home"),
    NavItem(Icons.Default.Search, navRoute: "search"),
    NavItem(Icons.Default.Menu, navRoute: "more")
)
```

Bottom navigation bar (cont.)

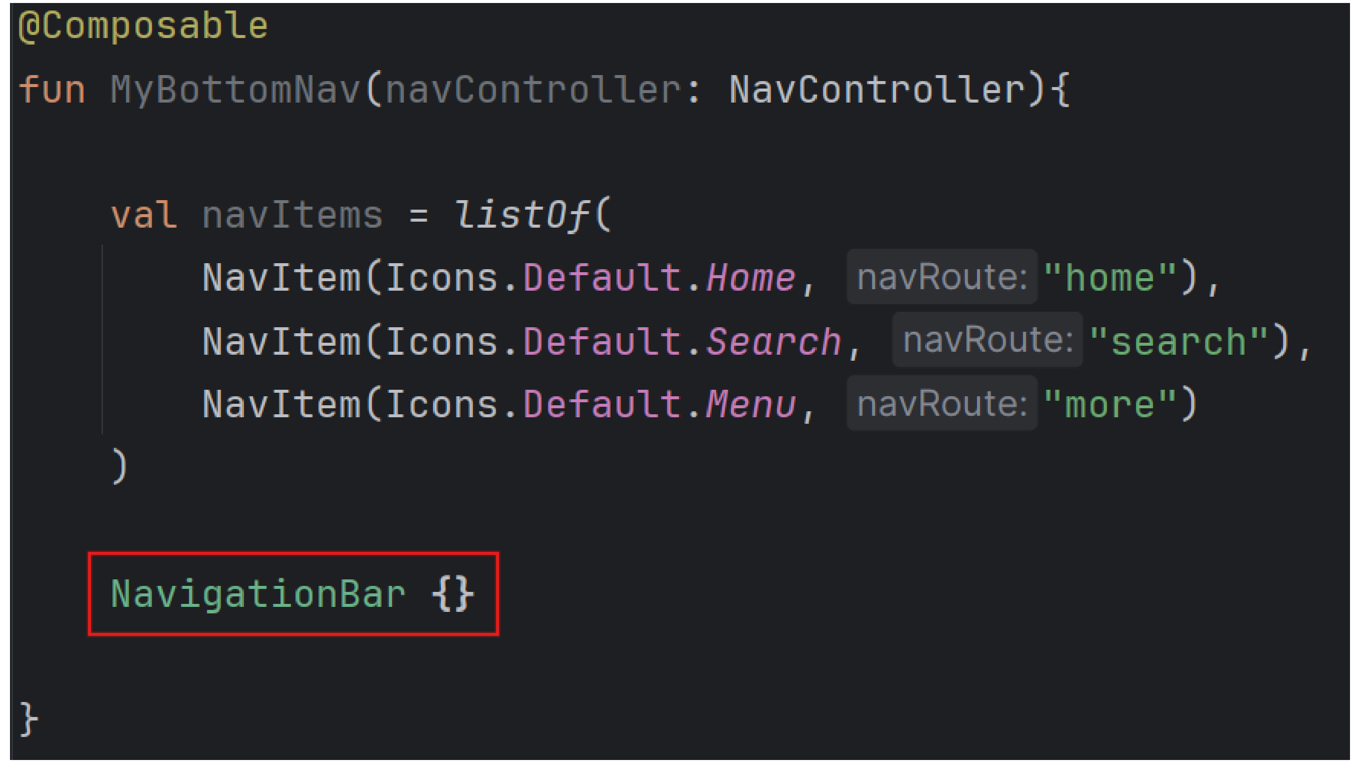
- We then create our *NavigationBar*



```
@Composable
fun MyBottomNav(navController: NavController){

    val navItems = listOf(
        NavItem(Icons.Default.Home, navRoute: "home"),
        NavItem(Icons.Default.Search, navRoute: "search"),
        NavItem(Icons.Default.Menu, navRoute: "more")
    )

    NavigationBar {}

}
```

底部导航栏（续）

- 然后我们创建我们的 *NavigationBar*

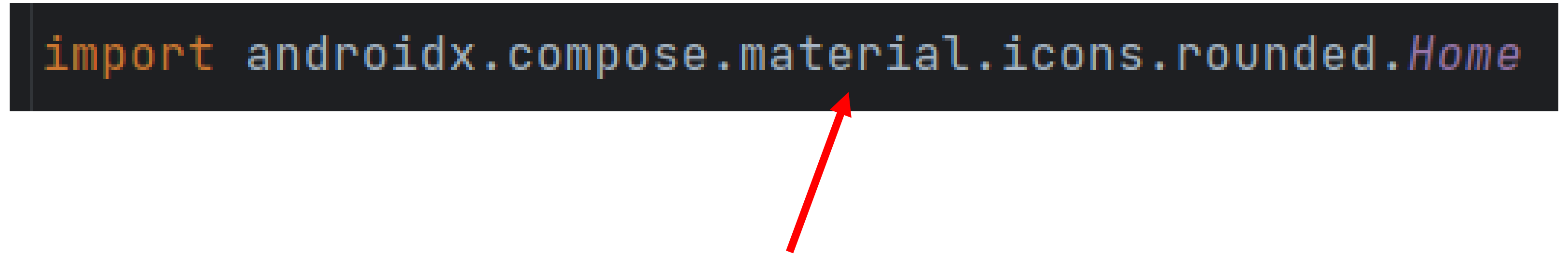


```
@Composable
fun MyBottomNav(navController: NavController){

    val navItems = listOf(
        NavItem(Icons.Default.Home, navRoute: "home"),
        NavItem(Icons.Default.Search, navRoute: "search"),
        NavItem(Icons.Default.Menu, navRoute: "more")
    )

    NavigationBar {}

}
```

Material Design

- Material Design provides us with all the components and styles you normally see when creating a google based mobile application
  - https://m3.material.io/

- There are also older versions of Material that you may be using without knowing
  - https://m2.material.io/
  - Ex.



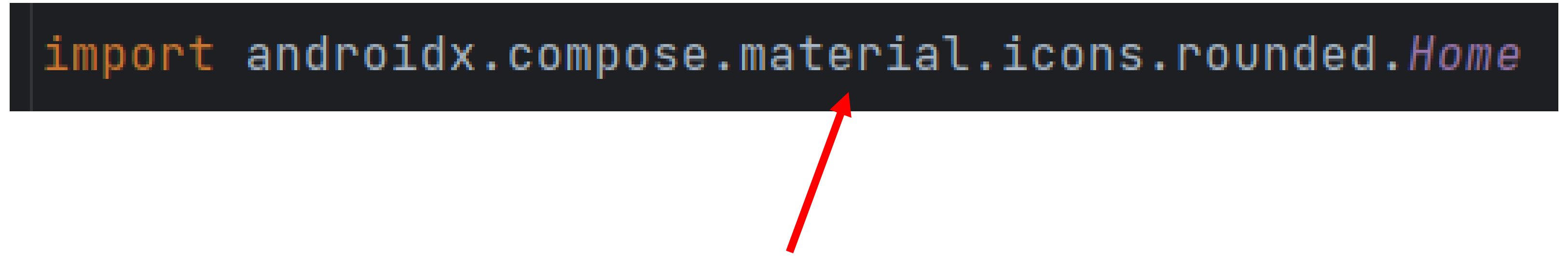
```
import androidx.compose.material.icons.rounded.Home
```

材料设计

- 材料设计为我们提供了创建基于谷歌的移动应用程序时通常会看到的所有组件和样式

  当你创建基于谷歌的移动应用程序时通常会看到这些
  - https://m3.material.io/

- 你可能正在使用一些较早版本的材料设计
  却 unaware
  - https://m2.material.io/
  - Ex.

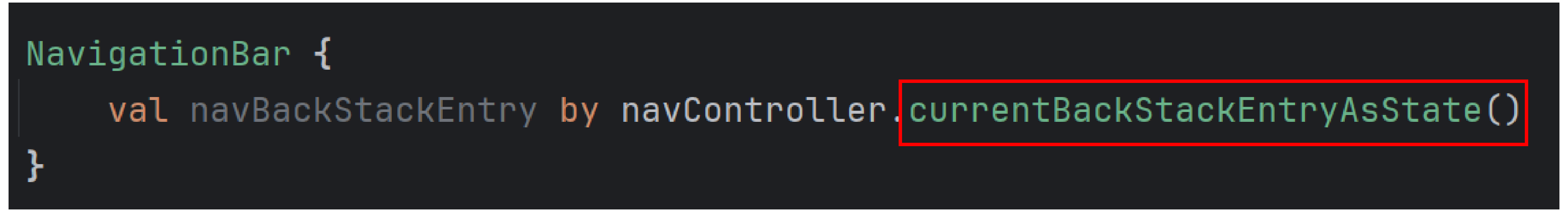


```
import androidx.compose.material.icons.rounded.Home
```

Bottom navigation bar (cont.)

- We then need to get a reference to the current back stack entry

- We reference it as mutable state, because we want to trigger a recomposition when navigating

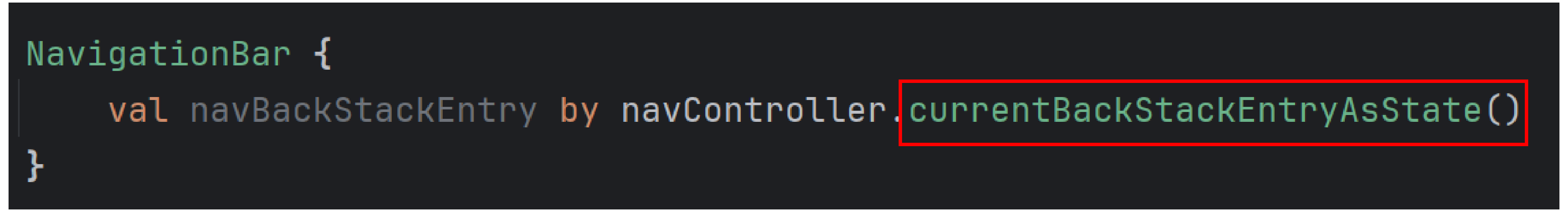


```
NavigationBar {
    val navBackStackEntry by navController.currentBackStackEntryAsState()
}
```

底部导航栏（续）

- 然后我们需要获取对当前返回栈条目的引用

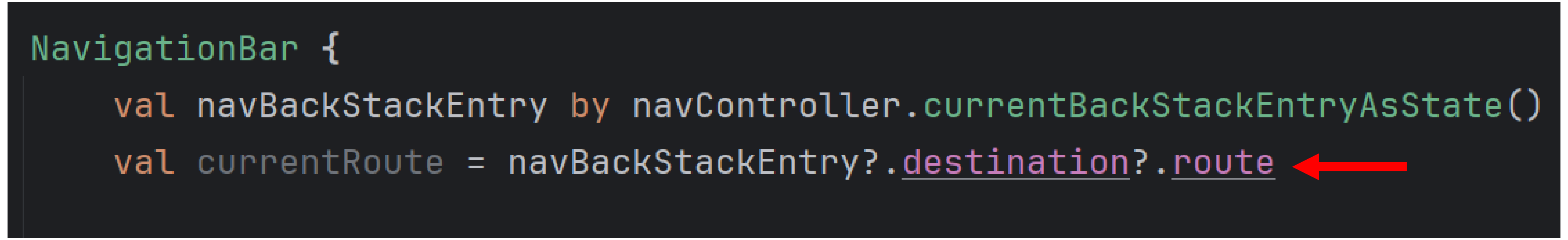
- 我们将其称为 可变状态，因为我们在导航时需要触发一次重组



```
NavigationBar {
    val navBackStackEntry by navController.currentBackStackEntryAsState()
}
```

Bottom navigation bar (cont.)

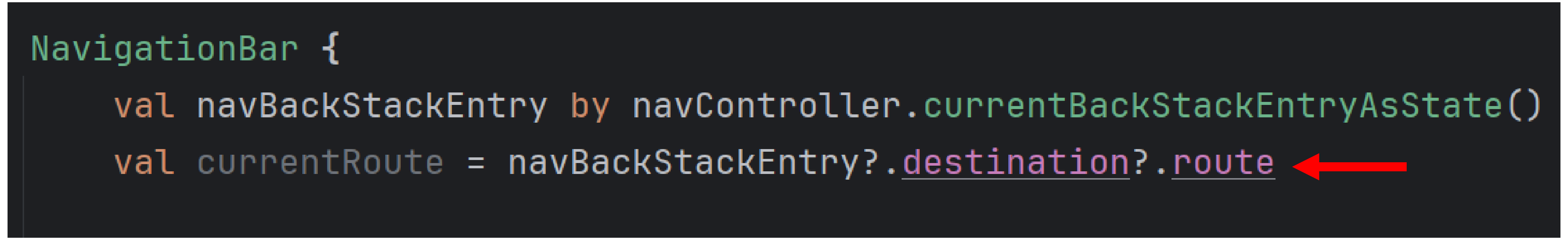
- Now we can get access to the current route



```
NavigationBar {
    val navBackStackEntry by navController.currentBackStackEntryAsState()
    val currentRoute = navBackStackEntry?.destination?.route
```

底部导航栏（续）

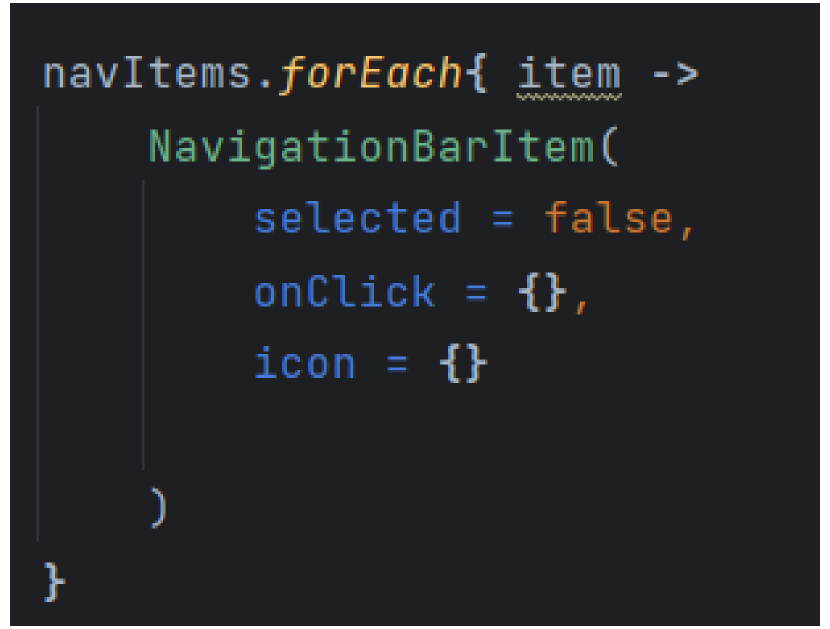
- 现在我们可以访问 当前路由



```
NavigationBar {
    val navBackStackEntry by navController.currentBackStackEntryAsState()
    val currentRoute = navBackStackEntry?.destination?.route
```

Bottom navigation bar (cont.)

- For each item in our nav bar we want to create a *NavigationBarItem*

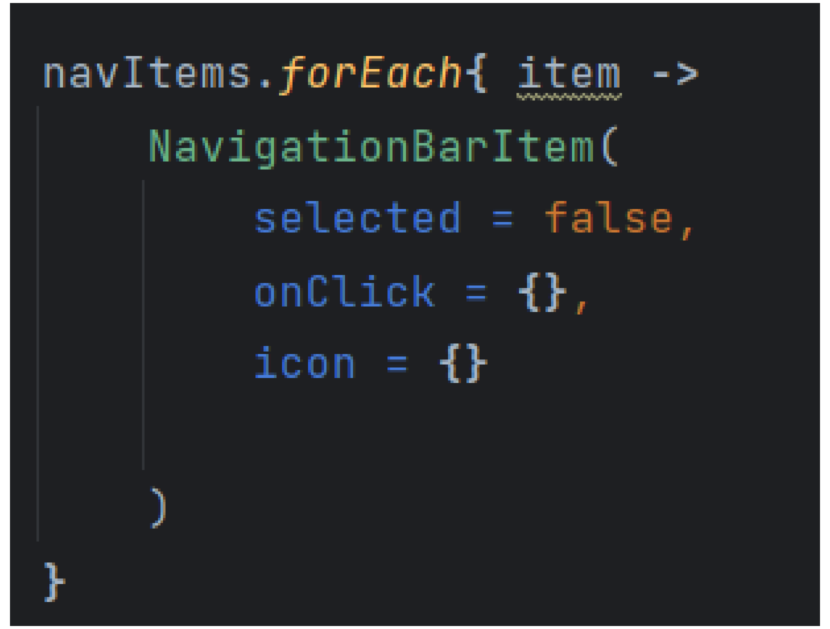


```
navItems.forEach{ item ->
    NavigationBarItem(
        selected = false,
        onClick = {},
        icon = {}

    )
}
```

- The three params we care about are selected, onClick and icon

底部导航栏（续）

- 对于导航栏中的每个项目，我们希望创建一个*NavigationBarItem*

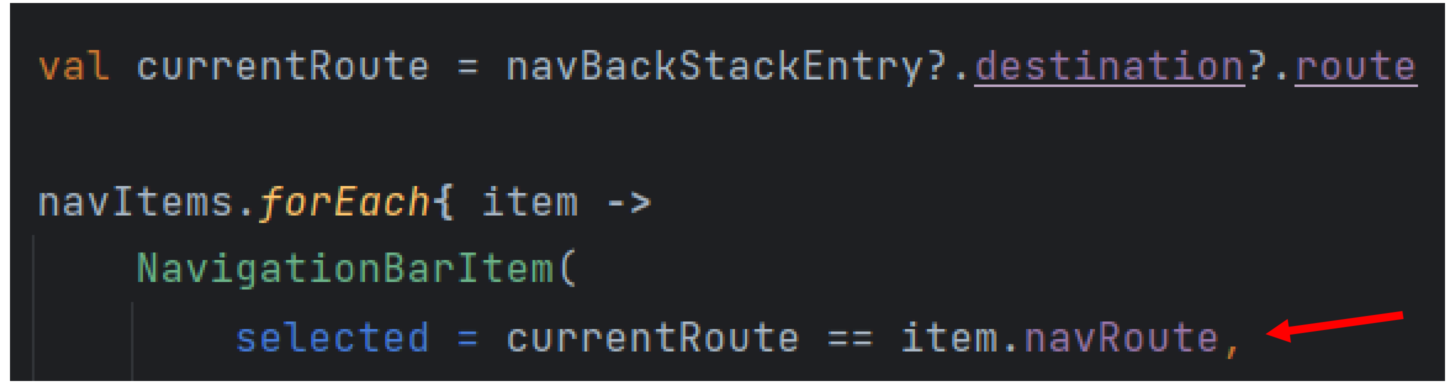


```
navItems.forEach{ item ->
    NavigationBarItem(
        selected = false,
        onClick = {},
        icon = {}

    )
}
```

- 我们关心的三个参数是 selected、onClick 和 icon

Bottom navigation bar (cont.)

- The *selected* param returns a *Boolean*

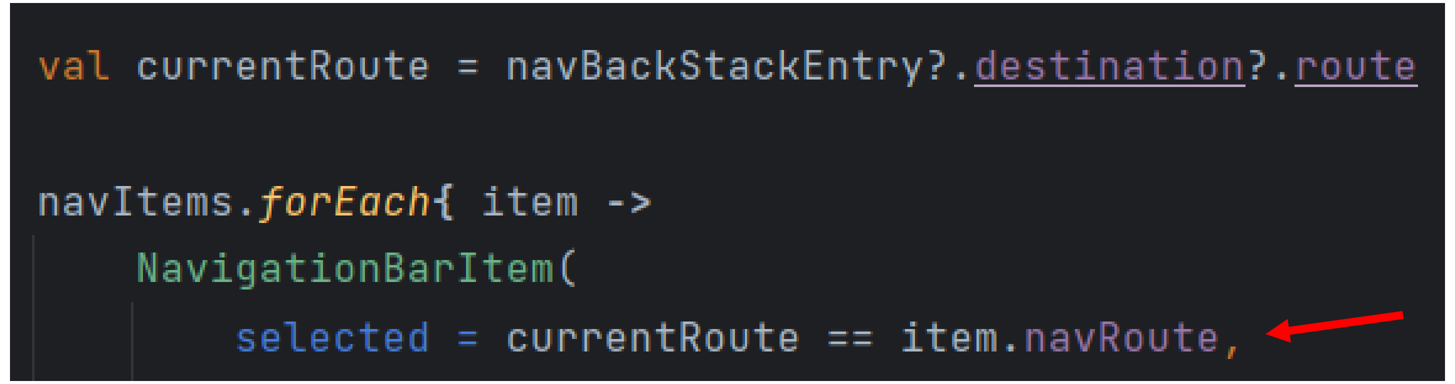


```
val currentRoute = navBackStackEntry?.destination?.route

navItems.forEach{ item ->
    NavigationBarItem(
        selected = currentRoute == item.navRoute,
```

- If the current route == item route, then we want to return true

底部导航栏（续）

- *selected* 参数返回一个 *Boolean*

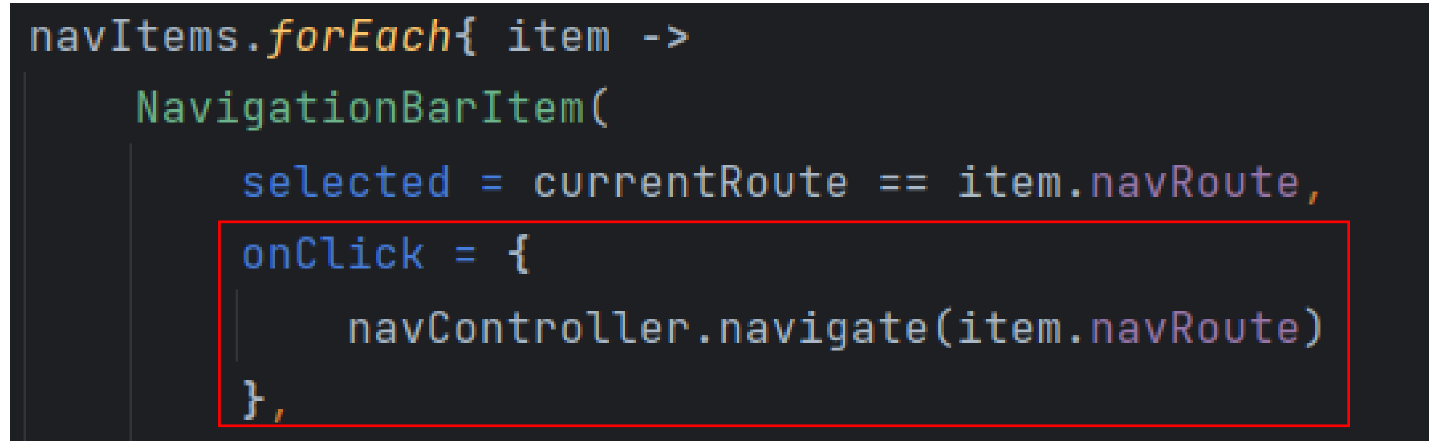


```
val currentRoute = navBackStackEntry?.destination?.route

navItems.forEach{ item ->
    NavigationBarItem(
        selected = currentRoute == item.navRoute,
```

- 如果 当前路由 == 与项的路由相同， 则我们希望返回 true

Bottom navigation bar (cont.)

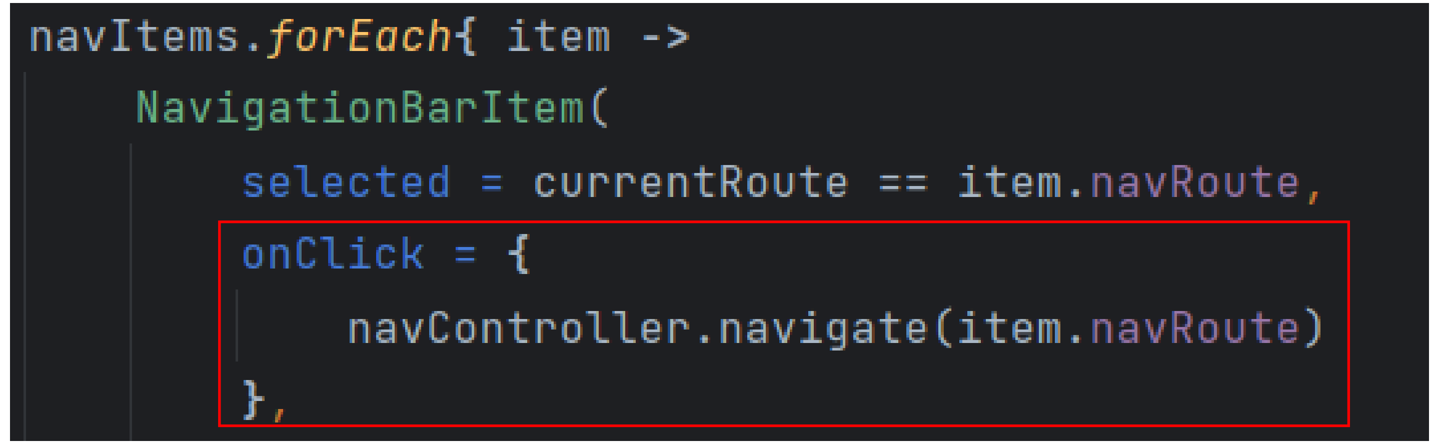
- When we click a nav bar item we want to navigate to that items route



```
navItems.forEach{ item ->
    NavigationBarItem(
        selected = currentRoute == item.navRoute,
        onClick = {
            navController.navigate(item.navRoute)
        },
```

底部导航栏（续）

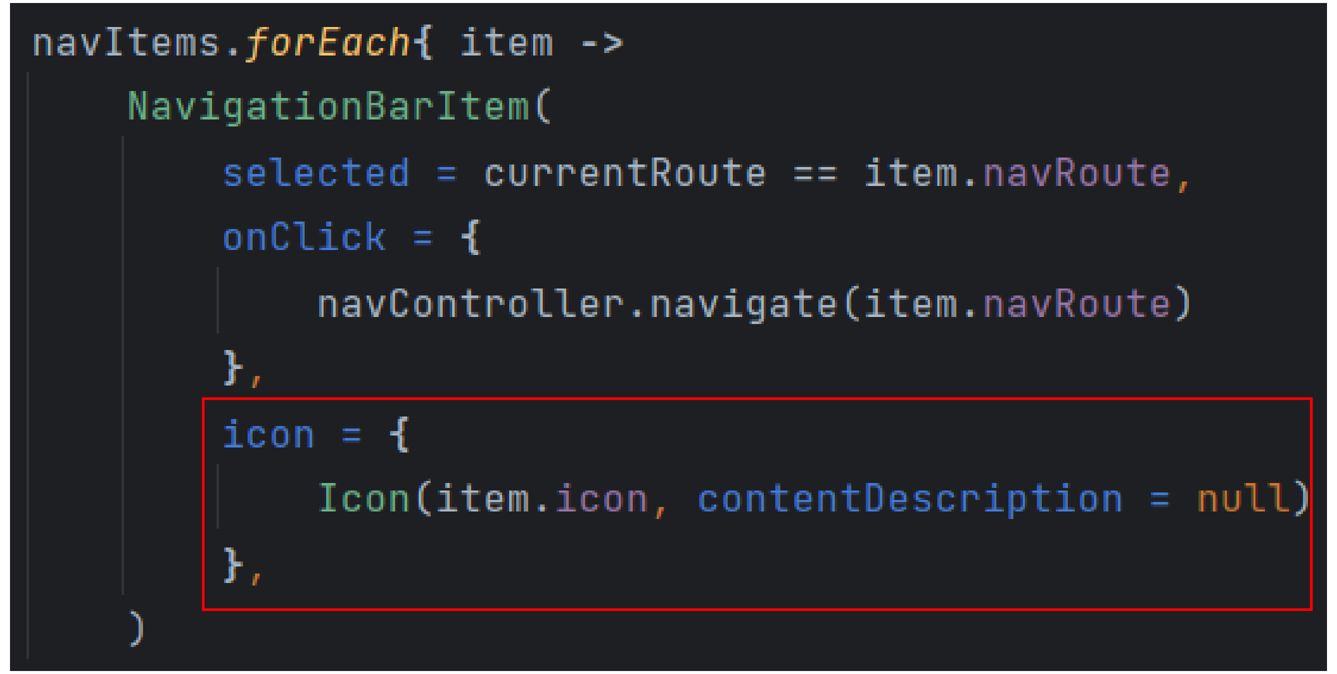
- 当我们点击导航栏项时，我们希望导航到该项的路由



```
navItems.forEach{ item ->
    NavigationBarItem(
        selected = currentRoute == item.navRoute,
        onClick = {
            navController.navigate(item.navRoute)
        },
```

Bottom navigation bar (cont.)

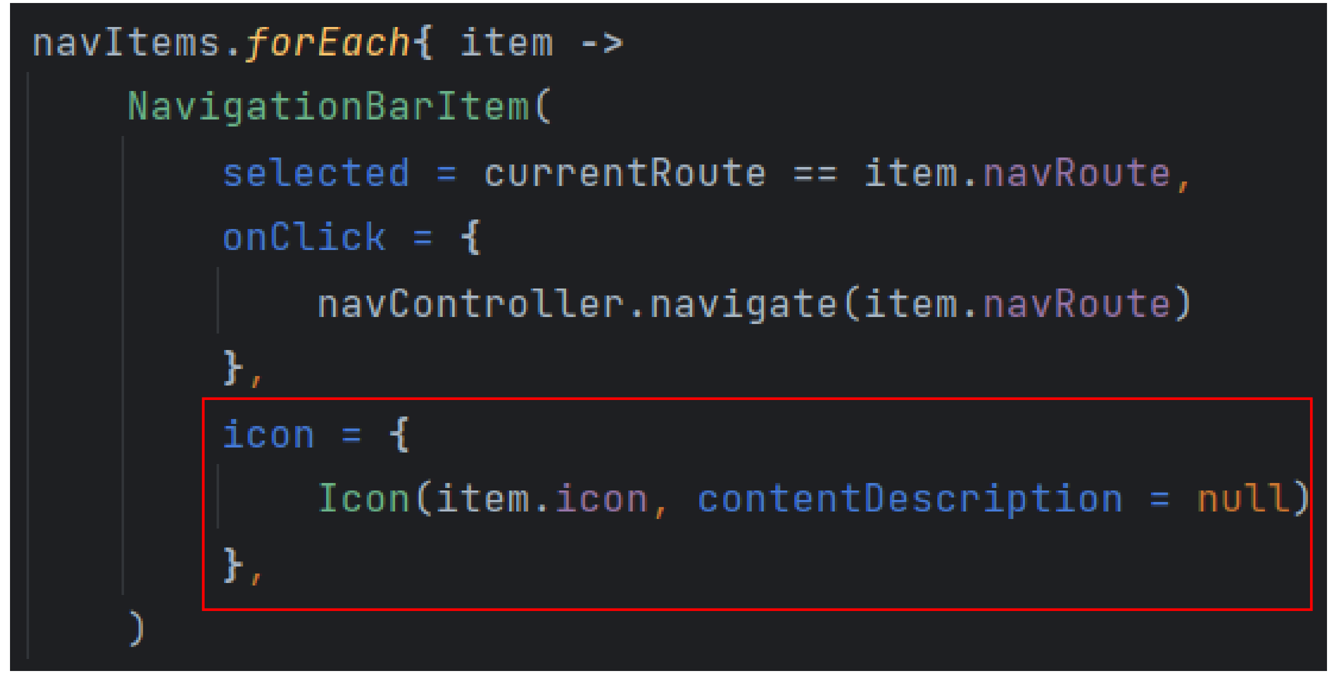
- Lastly, we want to display the specific icon for our nav bar item



```
navItems.forEach{ item ->
    NavigationBarItem(
        selected = currentRoute == item.navRoute,
        onClick = {
            navController.navigate(item.navRoute)
        },
        icon = {
            Icon(item.icon, contentDescription = null)
        },
    )
```

底部导航栏（续）

- 最后，我们希望显示我们导航栏项的特定图标



```
navItems.forEach{ item ->
    NavigationBarItem(
        selected = currentRoute == item.navRoute,
        onClick = {
            navController.navigate(item.navRoute)
        },
        icon = {
            Icon(item.icon, contentDescription = null)
        },
    )
```

Scaffold

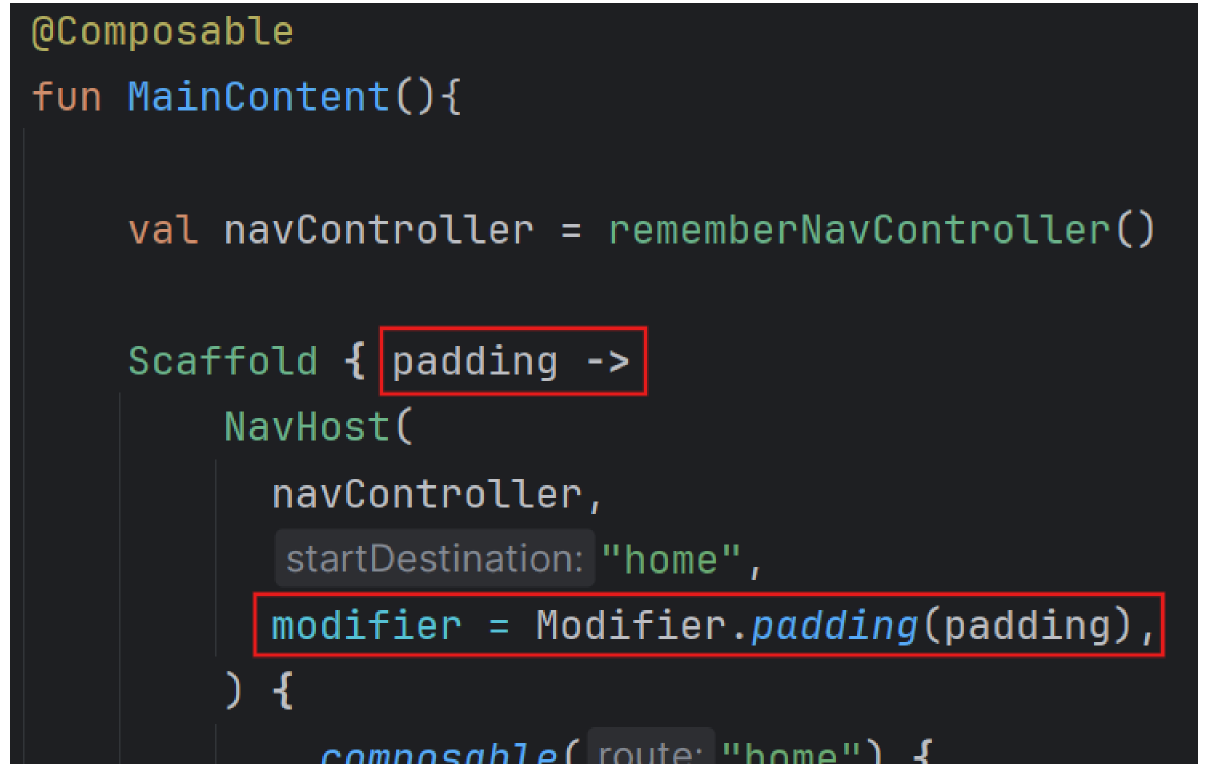
- To properly display our NavigationBar we can use a *Scaffold*
- A Scaffold provides the basic layout structure for our screen
- It allows us position top, bottom and side navigation bars around our screen content

脚手架

- 为了正确显示我们的导航栏，我们可以使用 *Scaffold*
- 脚手架为我们的屏幕提供了基本的布局结构
- 它允许我们在屏幕内容的顶部、底部和侧面放置导航栏

Scaffold (cont.)

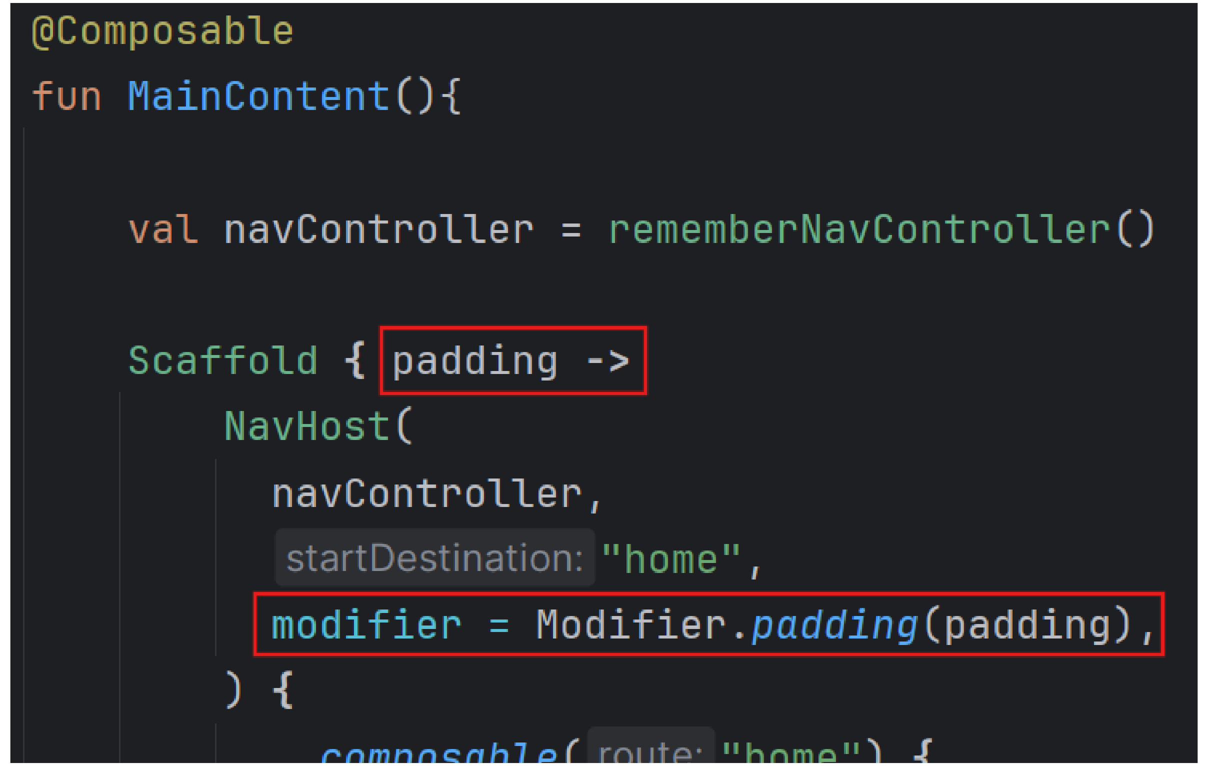
- The content inside a *Scaffold* needs to use its content padding



```
@Composable
fun MainContent(){

    val navController = rememberNavController()

    Scaffold { padding ->
        NavHost(
          navController,
          startDestination: "home",
          modifier = Modifier.padding(padding),
        ) {
            composable( route: "home") {
```

Scaffold（续）

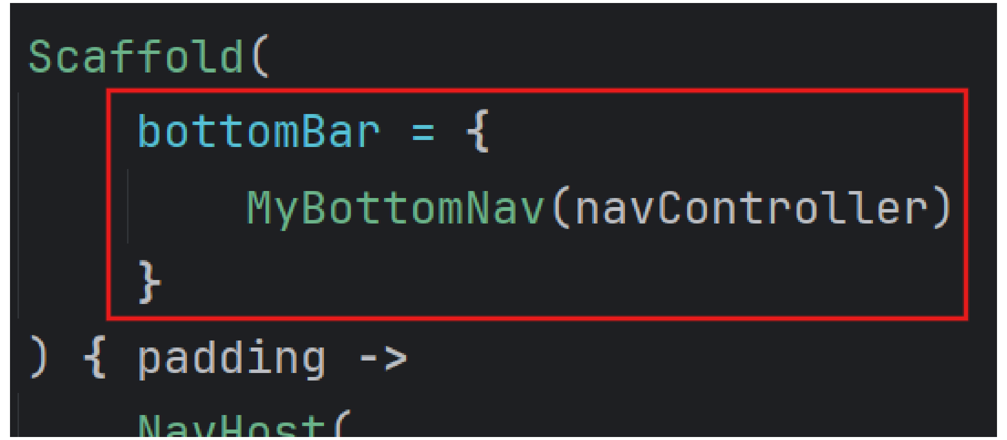
- Scaffold 内部的内容需要*Scaffold* 使用其内容内边距use its content padding



```
@Composable
fun MainContent(){

    val navController = rememberNavController()

    Scaffold { padding ->
        NavHost(
          navController,
          startDestination: "home",
          modifier = Modifier.padding(padding),
        ) {
            composable( route: "home") {
```

Scaffold (cont.)

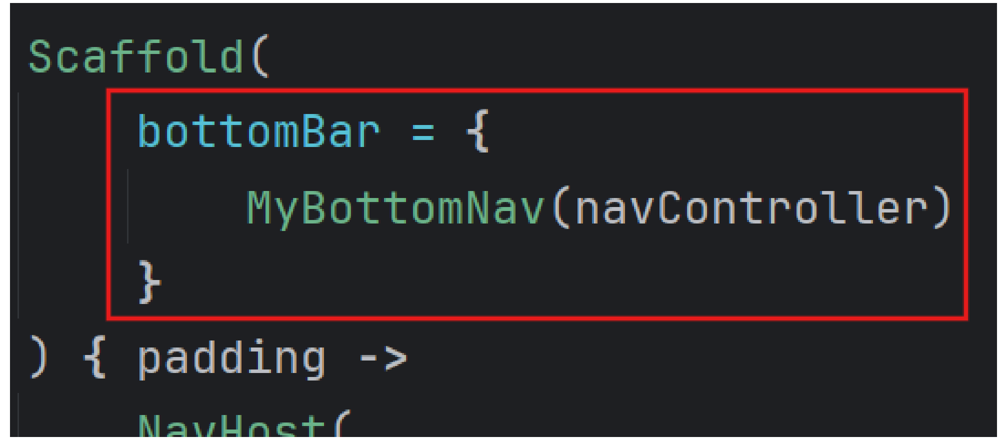
- The scaffold then makes it easy to add top, bottom or side navigation bars



```
Scaffold(
    bottomBar = {
        MyBottomNav(navController)
    }
) { padding ->
    NavHost(
```

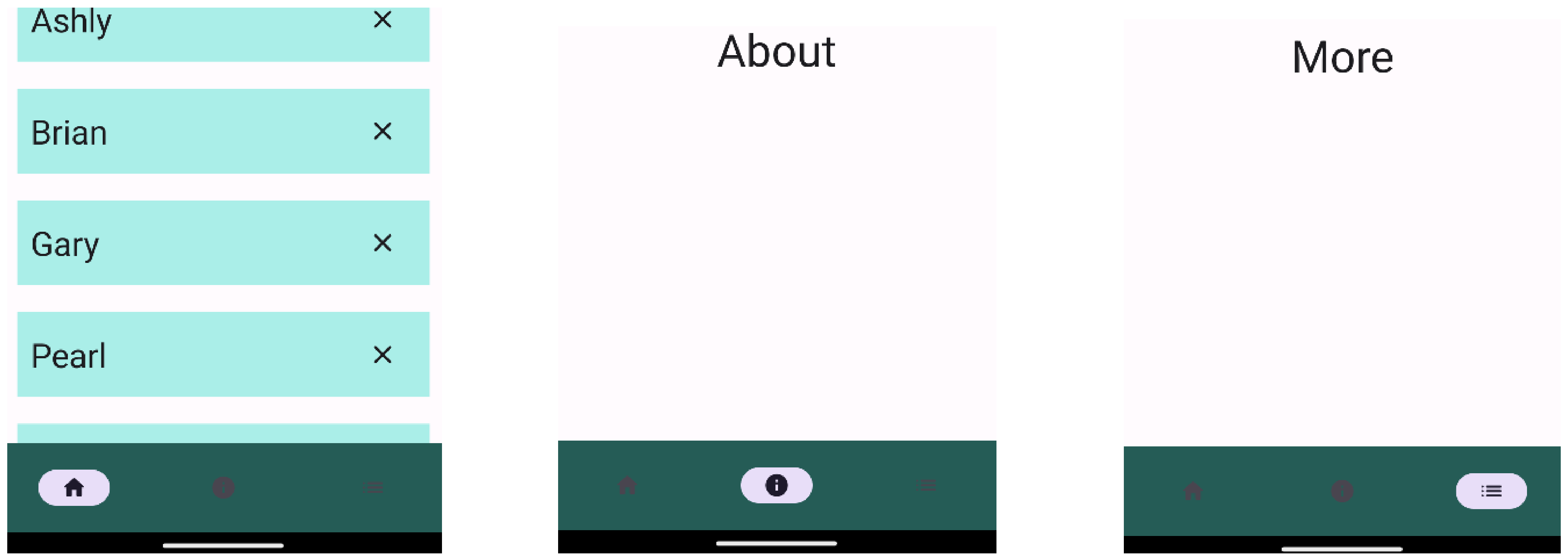
脚手架（续）

- 然后，该脚手架使得添加顶部、底部或侧边导航栏变得容易



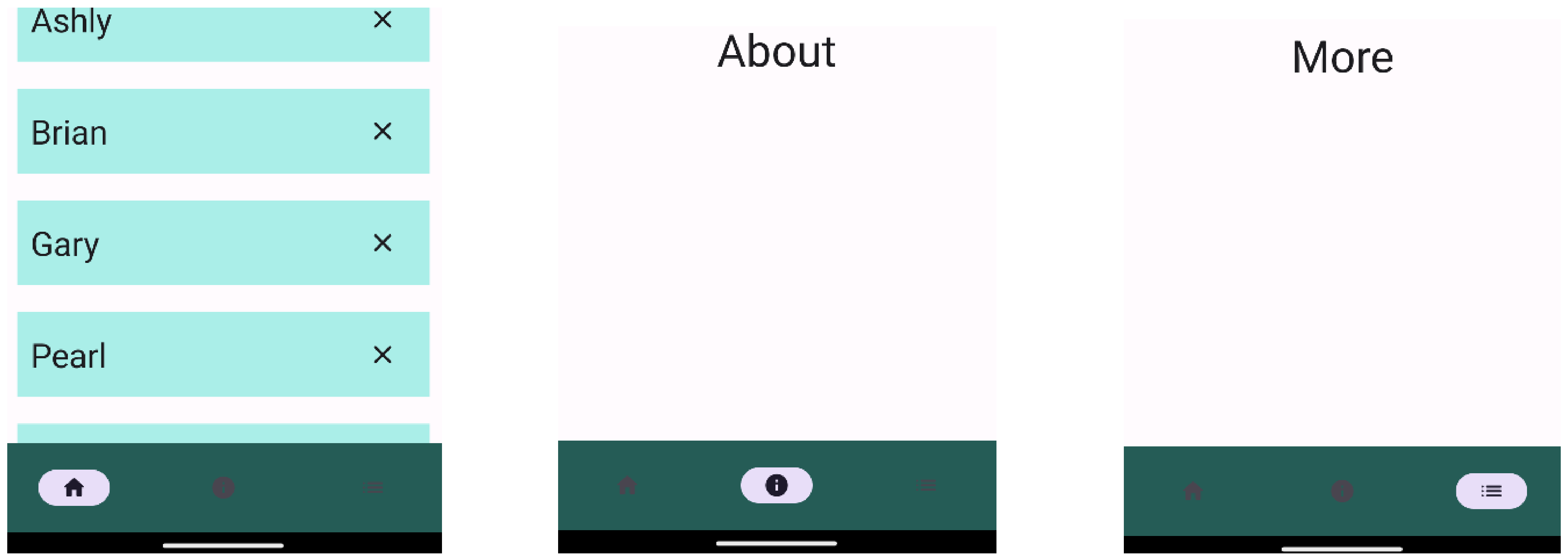
```
Scaffold(
    bottomBar = {
        MyBottomNav(navController)
    }
) { padding ->
    NavHost(
```

Bottom navigation bar (cont.)



- Your bottom nav bar is now set up and works well with the device back button

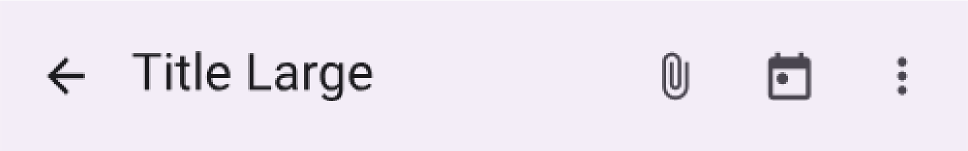
底部导航栏（续）



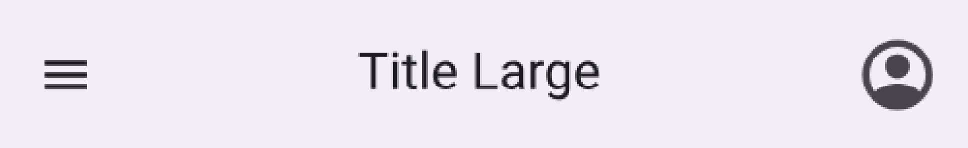
- 您的底部导航栏现已设置完成，并且能与设备的返回按钮良好配合工作

Top app bar

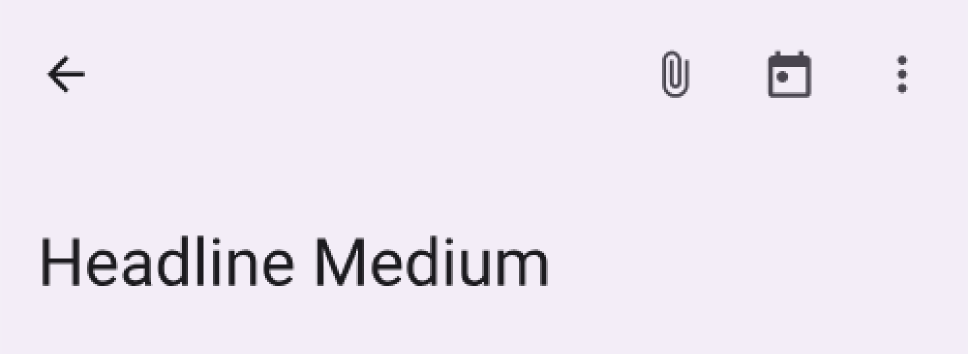
- Material3 provides us with a few different Top app bars we can use
  - Small



  - Center aligned

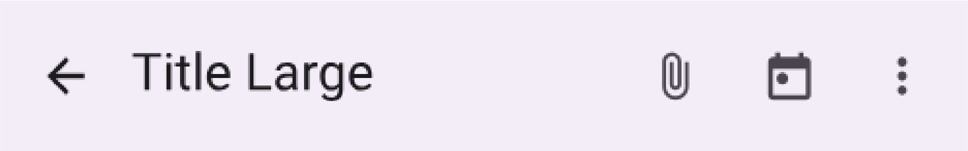


  - Medium

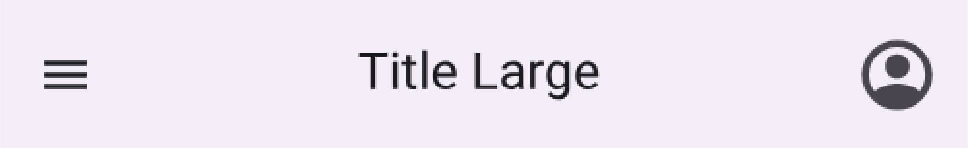


顶部应用栏

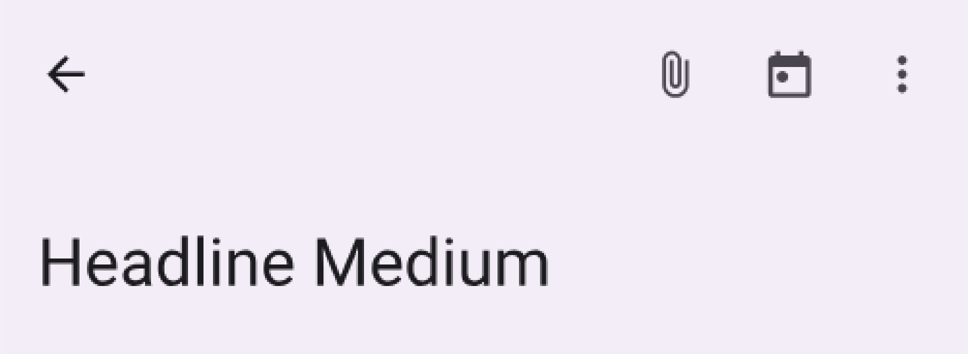
- Material3 为我们提供了几种不同的顶部应用栏可供使用
  - 小型



  - 居中对齐



  - 中型



Top app bar (cont.)

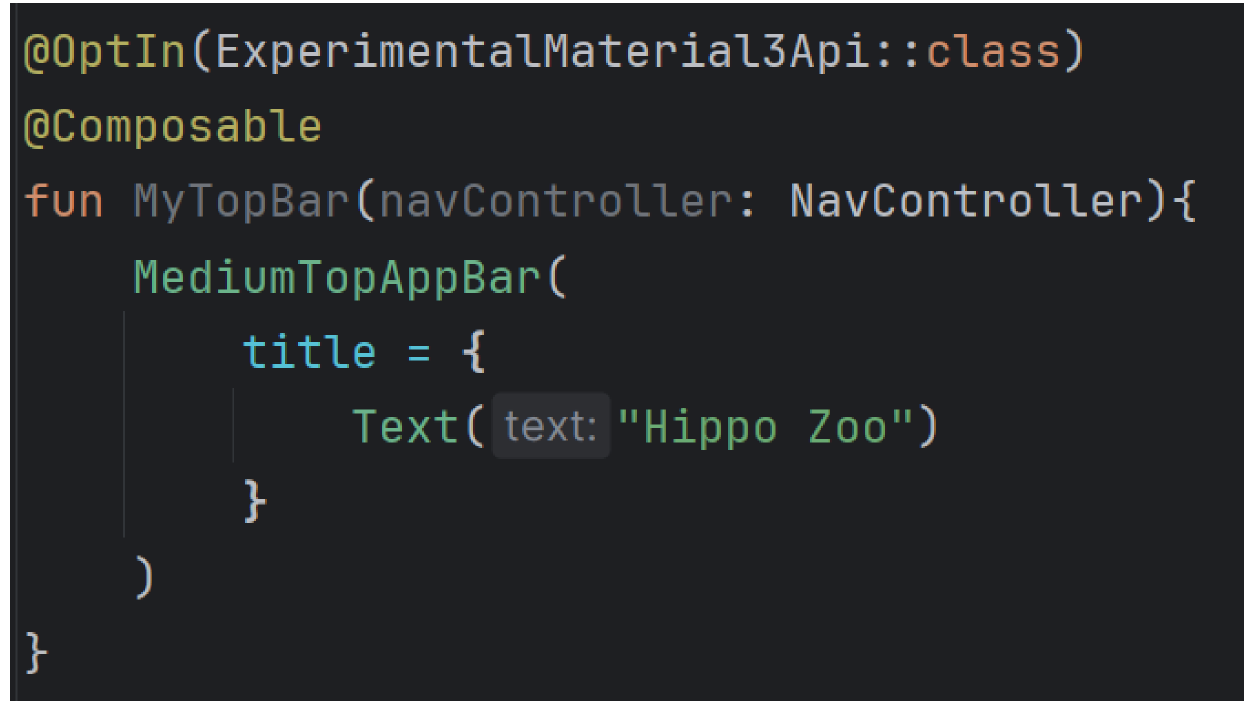
- All the top app bars have a few common parameters
  - *title,*
  - *navigationIcon*
  - *actions*

顶部应用栏（续）

- 所有顶部应用栏都有一些共同的参数
  - 标题，
  - 导航图标
  - 操作

Top app bar (cont.)

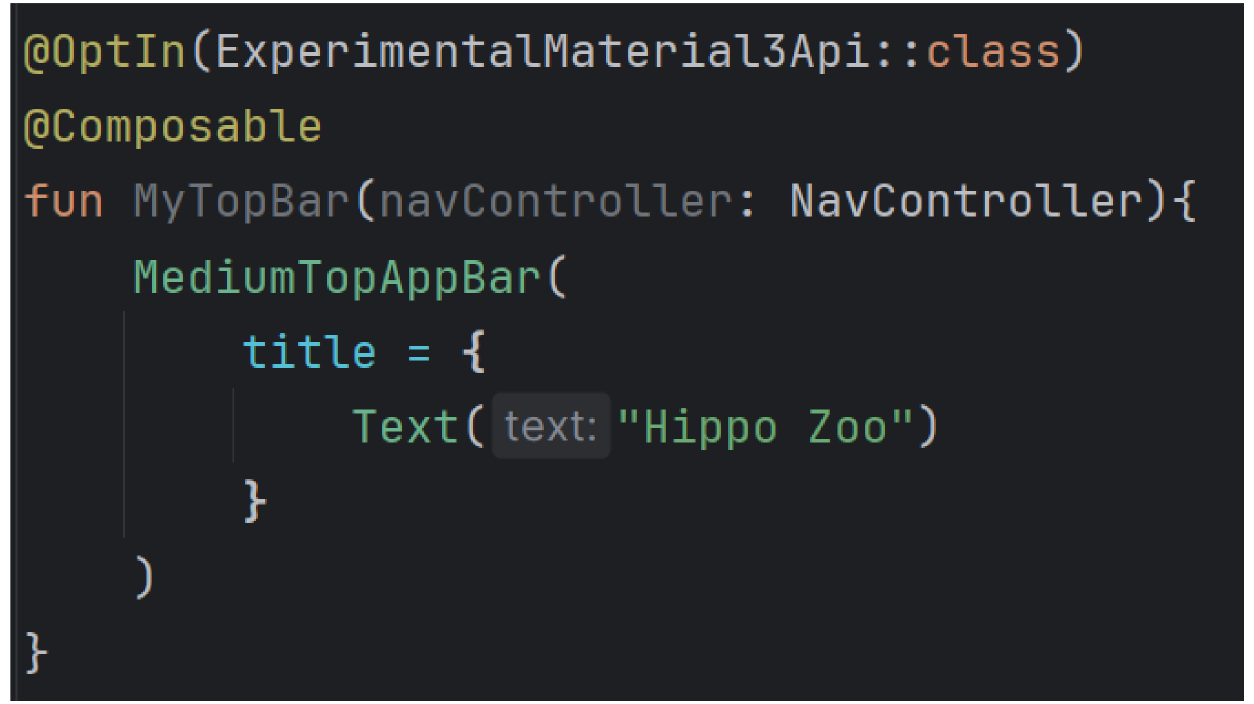
- *title* is the main text that displays on the top app bar



```
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun MyTopBar(navController: NavController){
    MediumTopAppBar(
        title = {
            Text( text: "Hippo Zoo")
        }
    )
}
```

顶部应用栏（续）

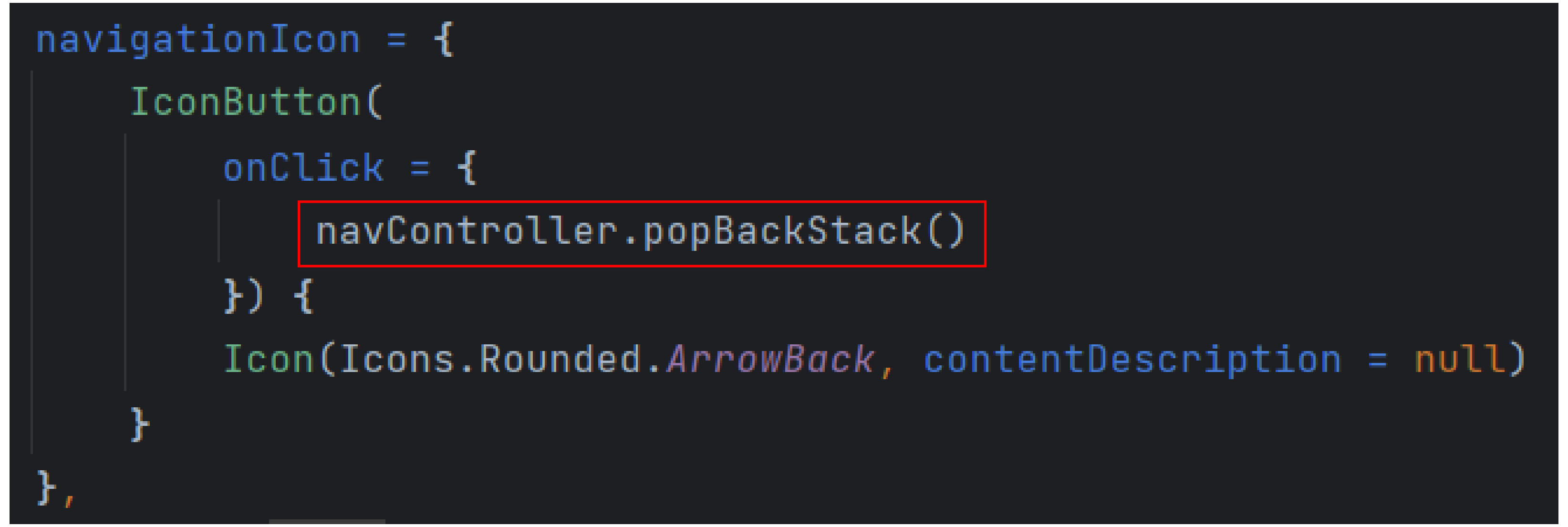
- 标题 是显示在顶部应用栏上的主要文本



```
@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun MyTopBar(navController: NavController){
    MediumTopAppBar(
        title = {
            Text( text: "Hippo Zoo")
        }
    )
}
```

Top app bar (cont.)

- *navigationIcon* sits on the left side of the top app bar; usually a back or home button

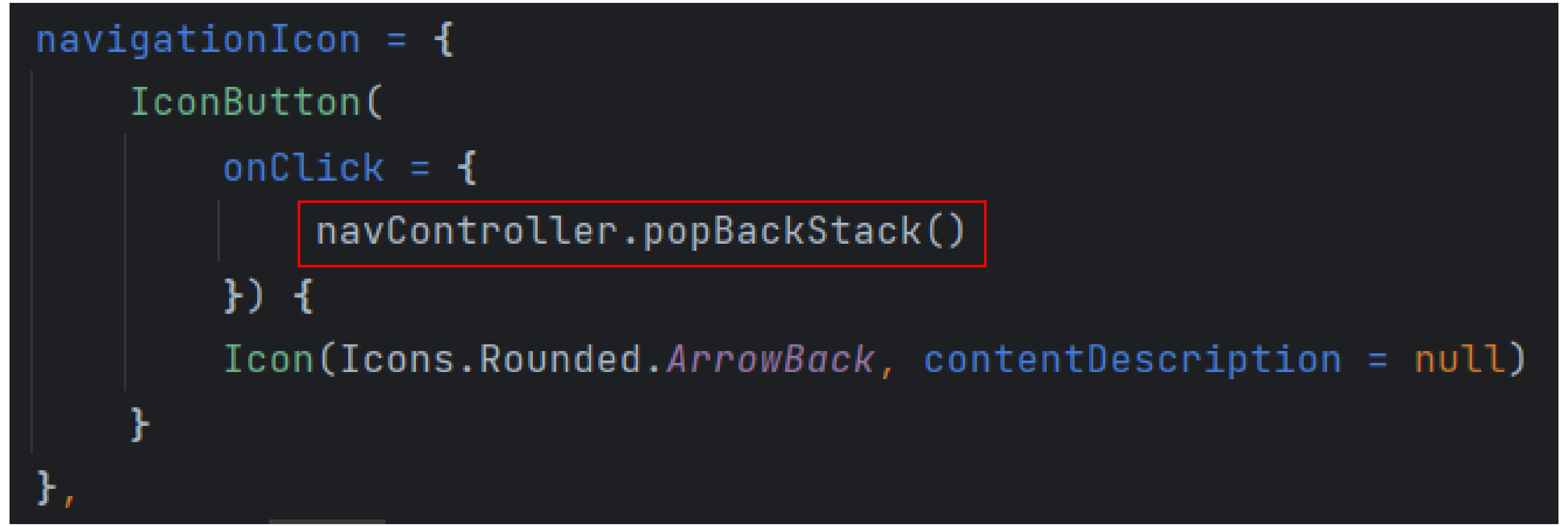


```
navigationIcon = {
    IconButton(
        onClick = {
            navController.popBackStack()
        }) {
        Icon(Icons.Rounded.ArrowBack, contentDescription = null)
    }
},
```

- The nav controller's *popBackStack* takes us back to the previous destination

顶部应用栏（续）

- *navigationIcon* 位于顶部应用栏的左侧；通常为返回或主页按钮

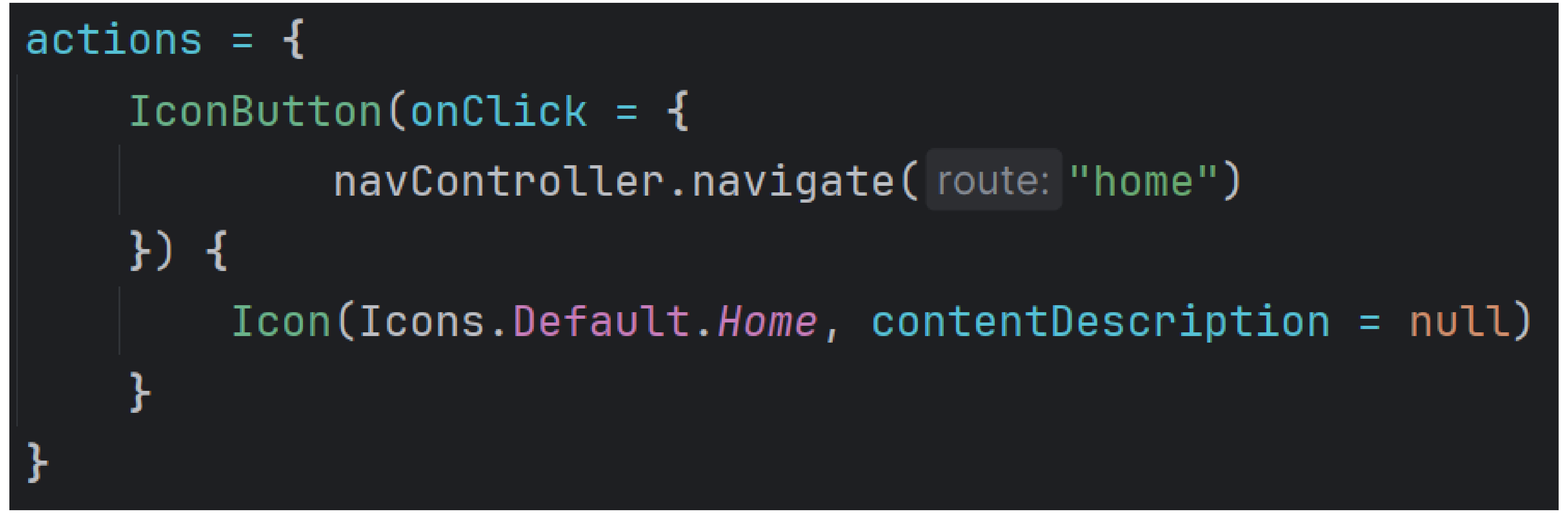


```
navigationIcon = {
    IconButton(
        onClick = {
            navController.popBackStack()
        }) {
        Icon(Icons.Rounded.ArrowBack, contentDescription = null)
    }
},
```

- 导航控制器'的 *popBackStack* 将我们带回上一个目的地

Top app bar (cont.)

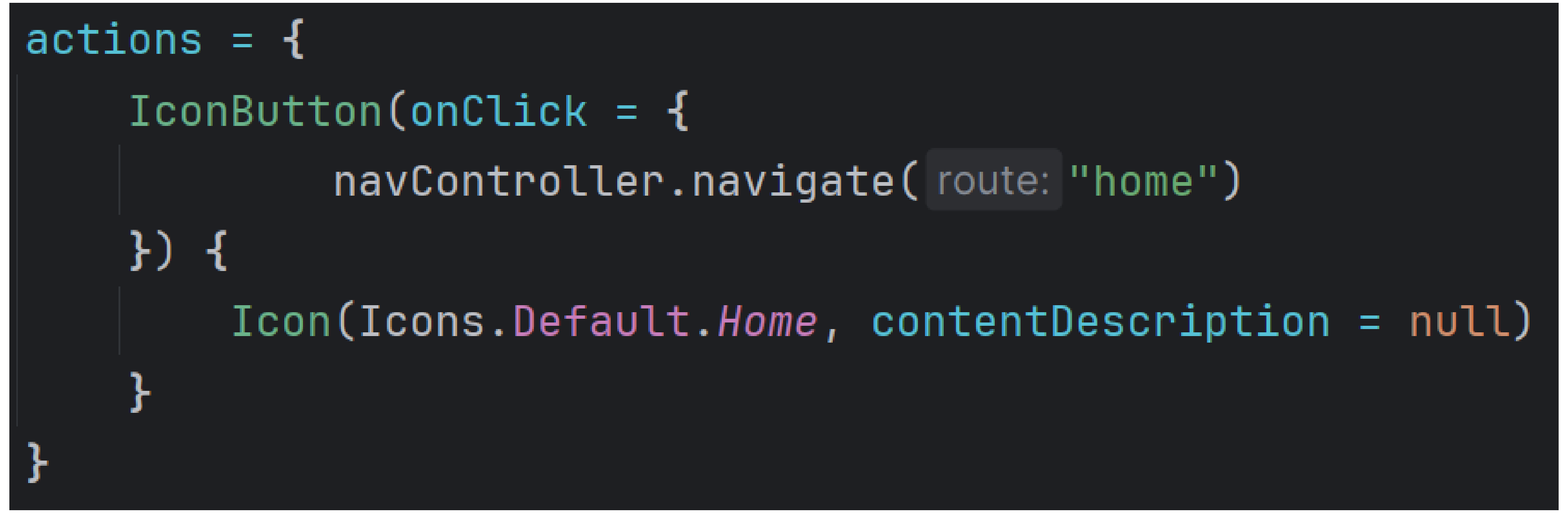
- *actions* are a row of composables that sit on the right side of the top app bar



```
actions = {
    IconButton(onClick = {
            navController.navigate( route: "home")
    }) {
        Icon(Icons.Default.Home, contentDescription = null)
    }
}
```

顶部应用栏（续）

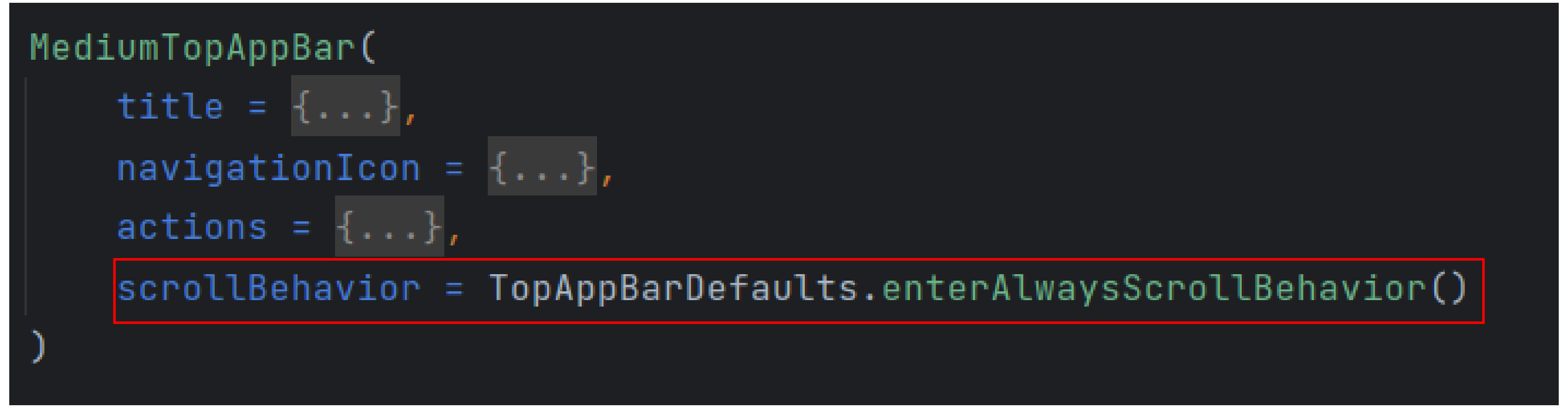
- 操作 是一排位于顶部应用栏右侧的可组合项



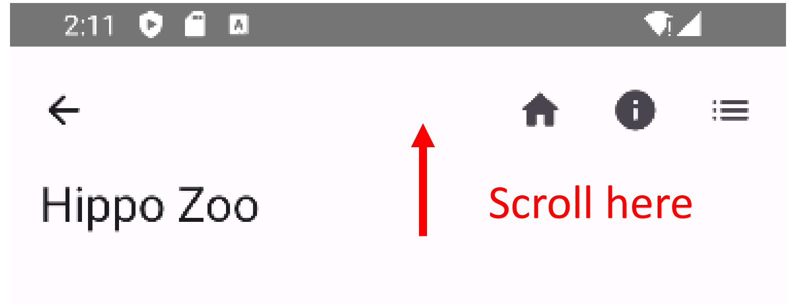
```
actions = {
    IconButton(onClick = {
            navController.navigate( route: "home")
    }) {
        Icon(Icons.Default.Home, contentDescription = null)
    }
}
```

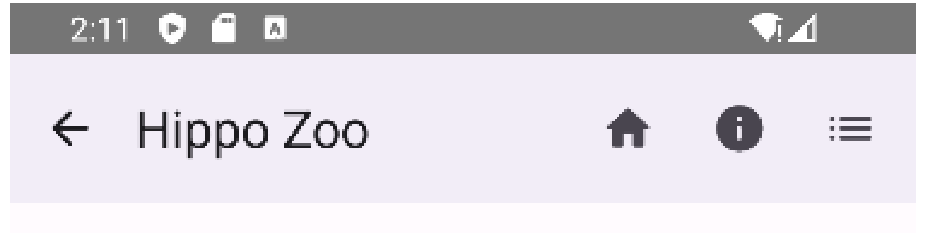
Top app bar (cont.)

- We can also allow our top app bar to collapse and expand



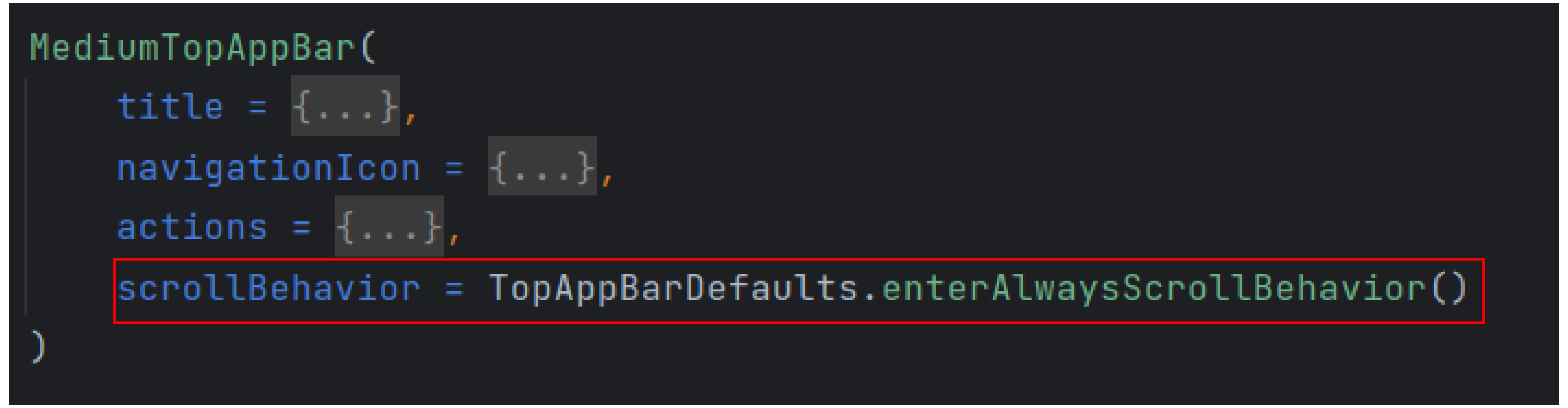
```
MediumTopAppBar(
    title = {...},
    navigationIcon = {...},
    actions = {...},
    scrollBehavior = TopAppBarDefaults.enterAlwaysScrollBehavior()
)
```



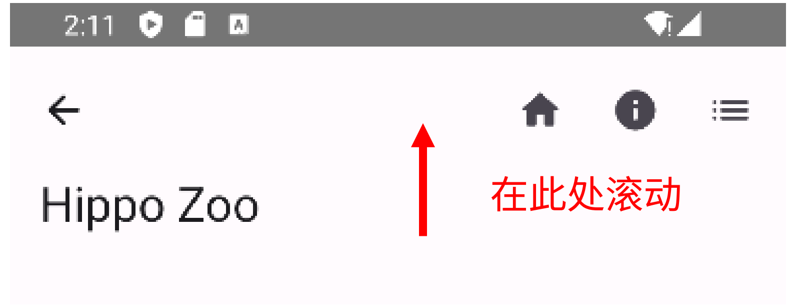


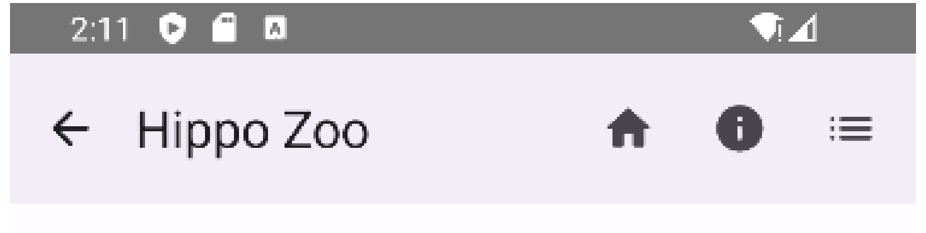
顶部应用栏（续）

- 我们还可以让顶部应用栏能够收起和展开



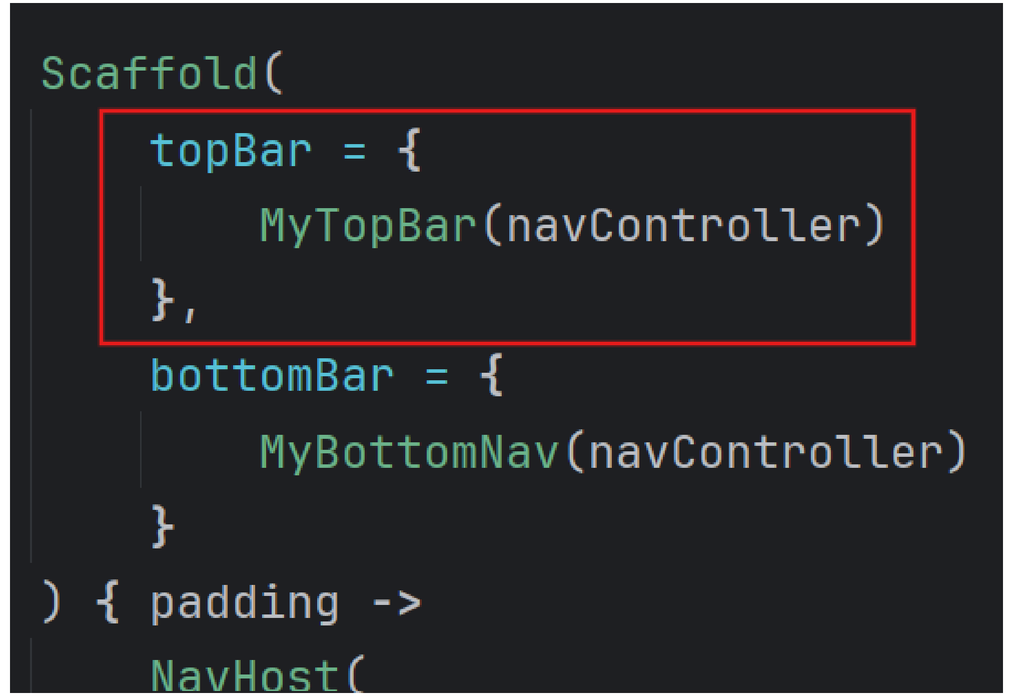
```
MediumTopAppBar(
    title = {...},
    navigationIcon = {...},
    actions = {...},
    scrollBehavior = TopAppBarDefaults.enterAlwaysScrollBehavior()
)
```





Top app bar (cont.)

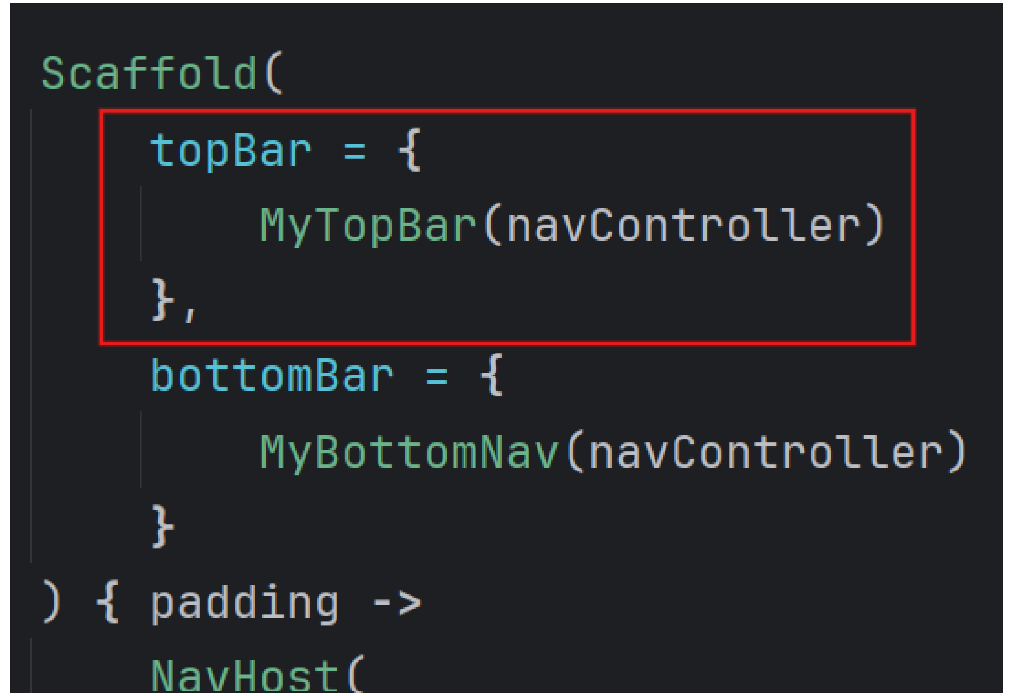
- Add the top app bar to your *Scaffold* now



```
Scaffold(
    topBar = {
        MyTopBar(navController)
    },
    bottomBar = {
        MyBottomNav(navController)
    }
) { padding ->
    NavHost(
```

顶部应用栏（续）

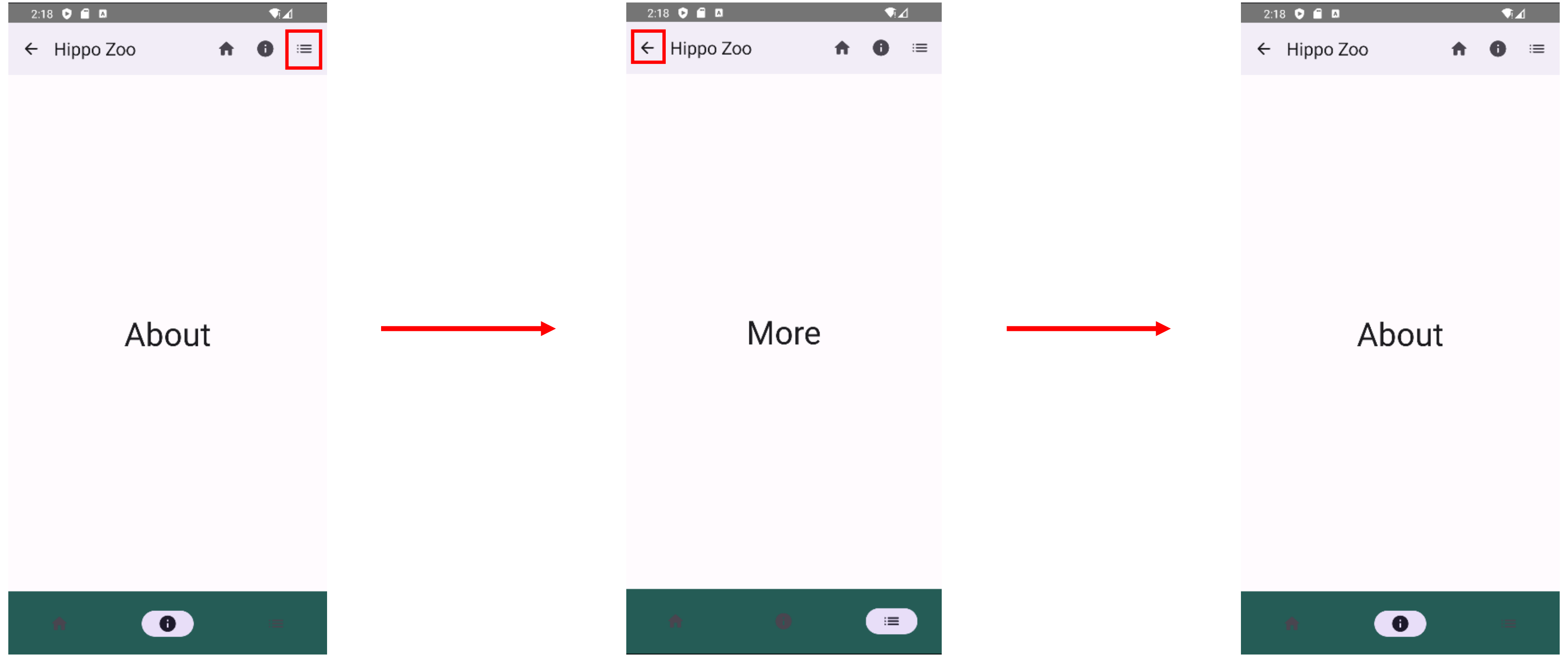
- 现在将顶部应用栏添加到您的*Scaffold*中



```
Scaffold(
    topBar = {
        MyTopBar(navController)
    },
    bottomBar = {
        MyBottomNav(navController)
    }
) { padding ->
    NavHost(
```

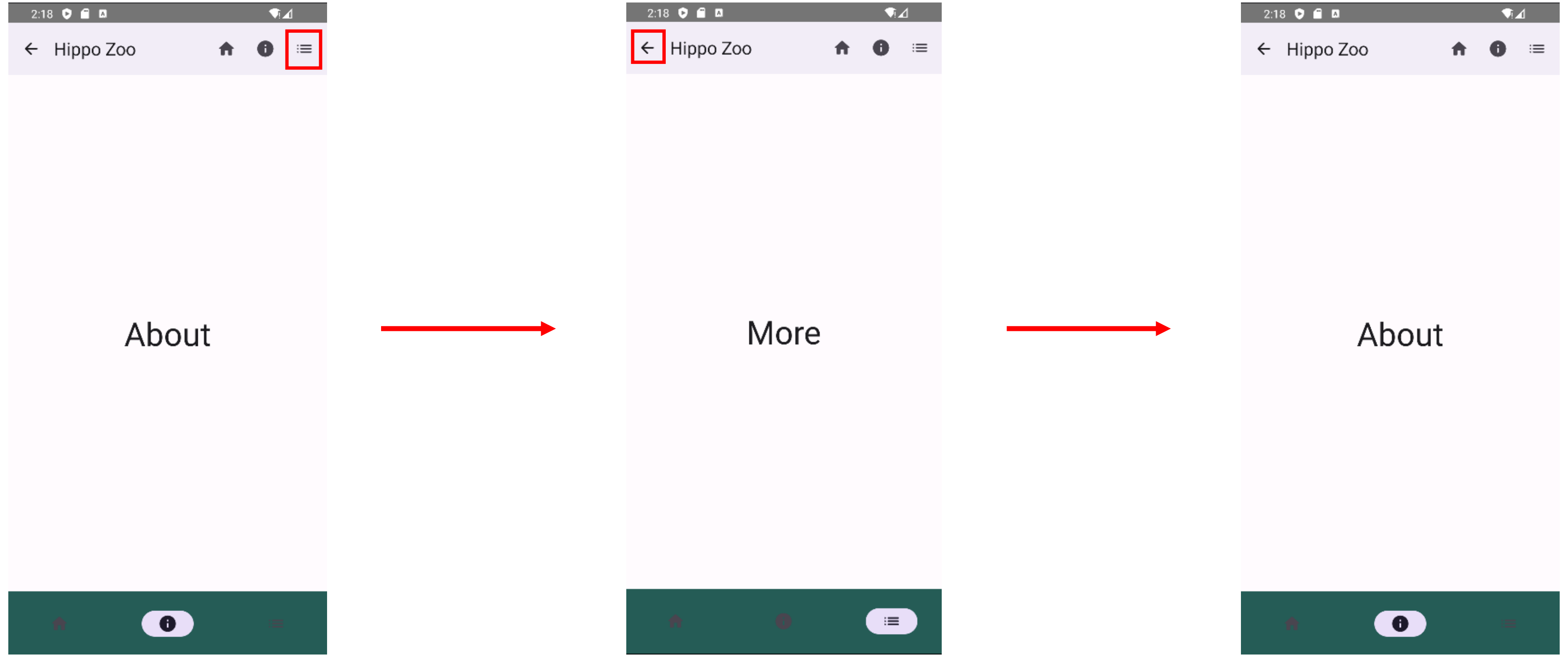
Navigation (cont.)

- Your top app bar now works well with the bottom nav bar and the navigation back stack



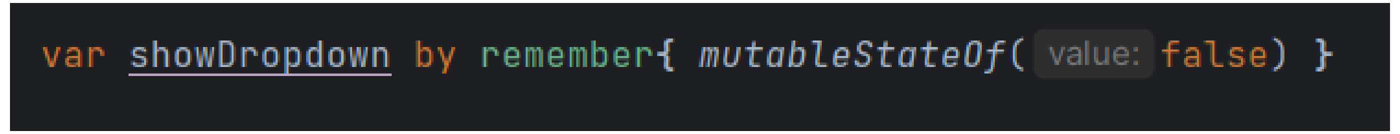
导航（续）

- 您的顶部应用栏现在可与底部导航栏以及导航返回栈

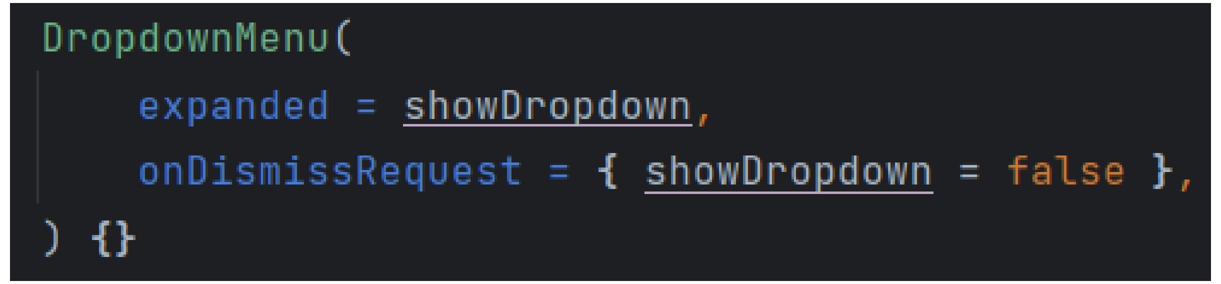


Dropdown menu

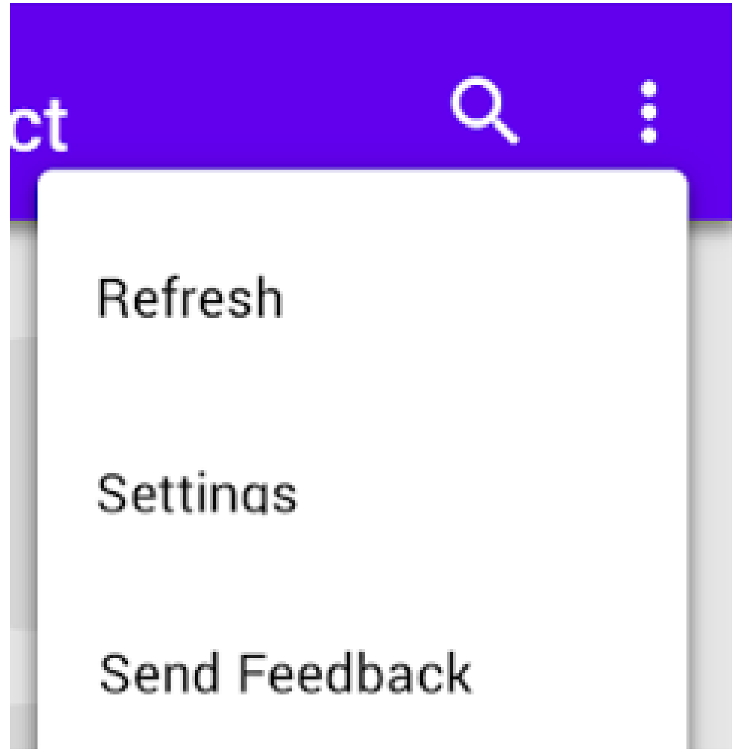
- To create a dropdown menu, we first need to create some state as a boolean



```
var showDropdown by remember{ mutableStateOf( value: false) }
```

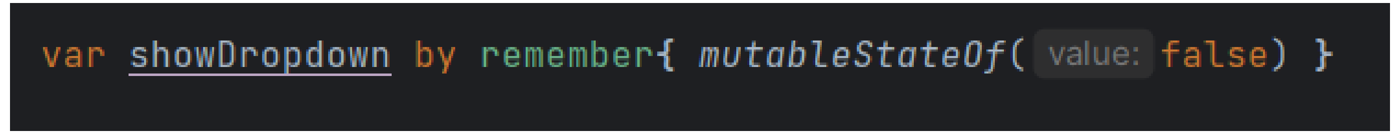


```
DropdownMenu(
    expanded = showDropdown,
    onDismissRequest = { showDropdown = false },
) {}
```

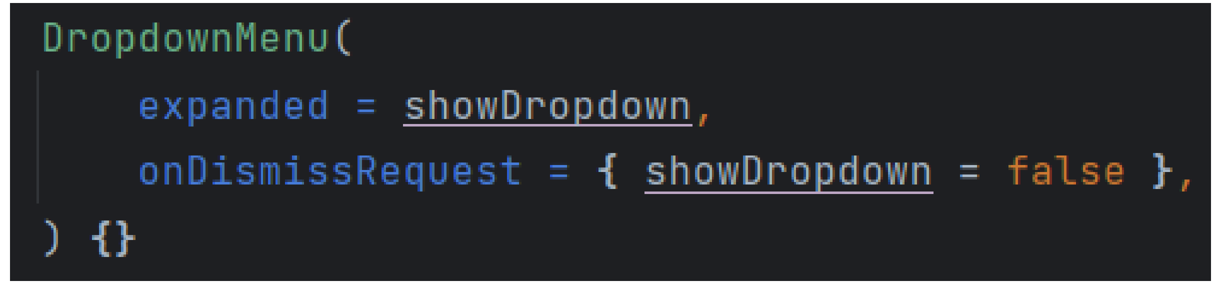


下拉菜单

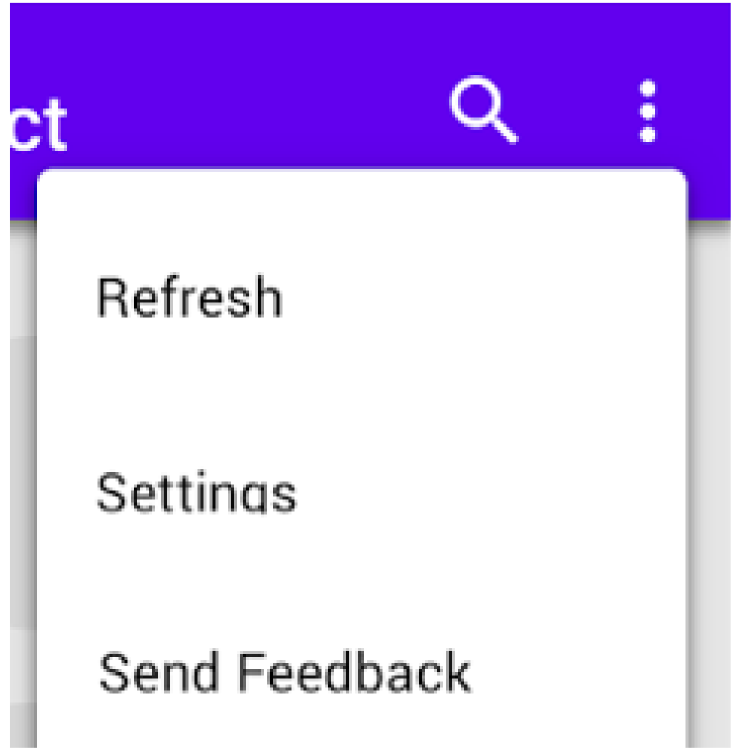
- 要创建一个下拉菜单，我们首先需要创建一个布尔类型的状态



```
var showDropdown by remember{ mutableStateOf( value: false) }
```

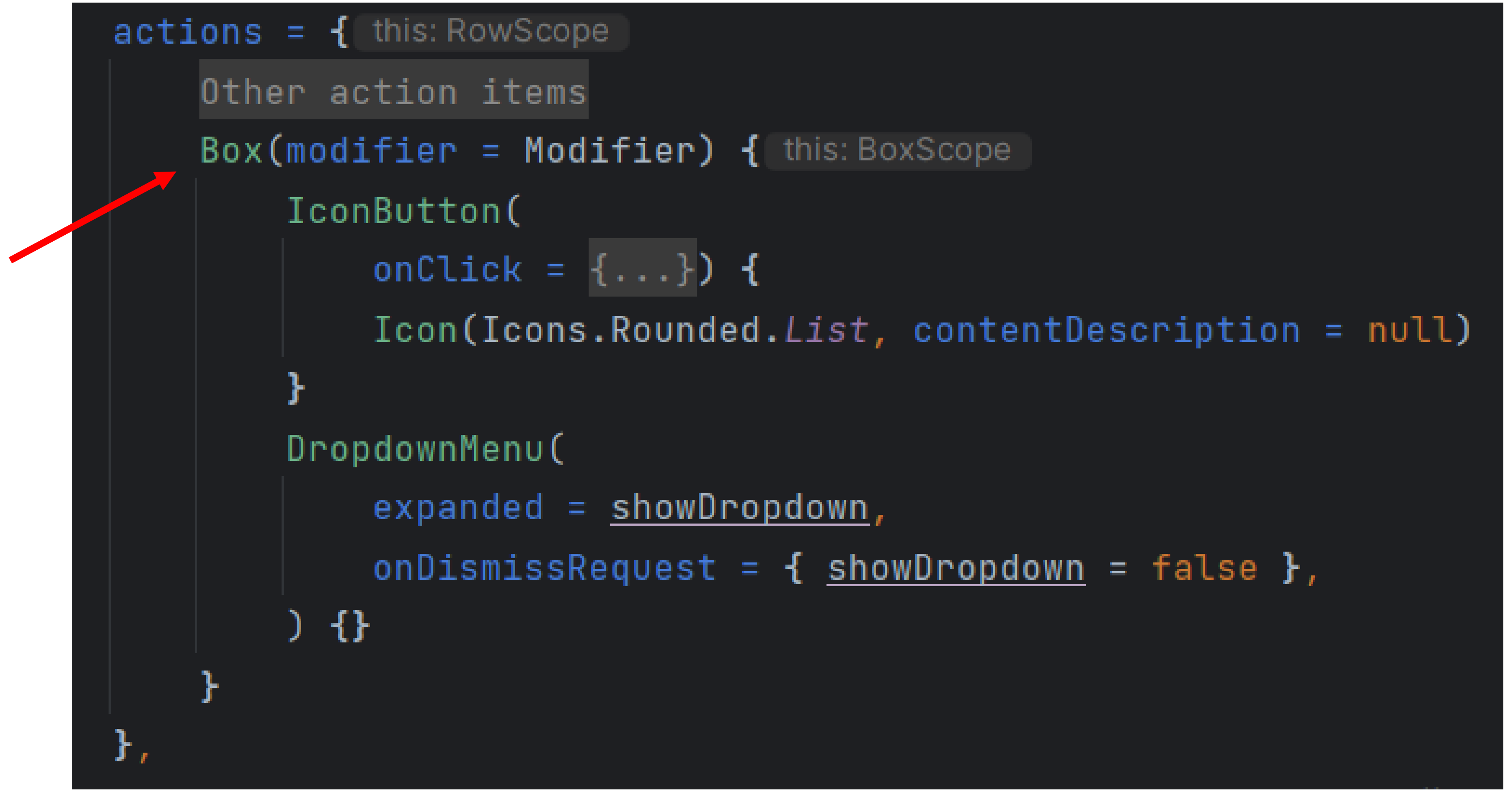


```
DropdownMenu(
    expanded = showDropdown,
    onDismissRequest = { showDropdown = false },
) {}
```



Dropdown menu (cont.)

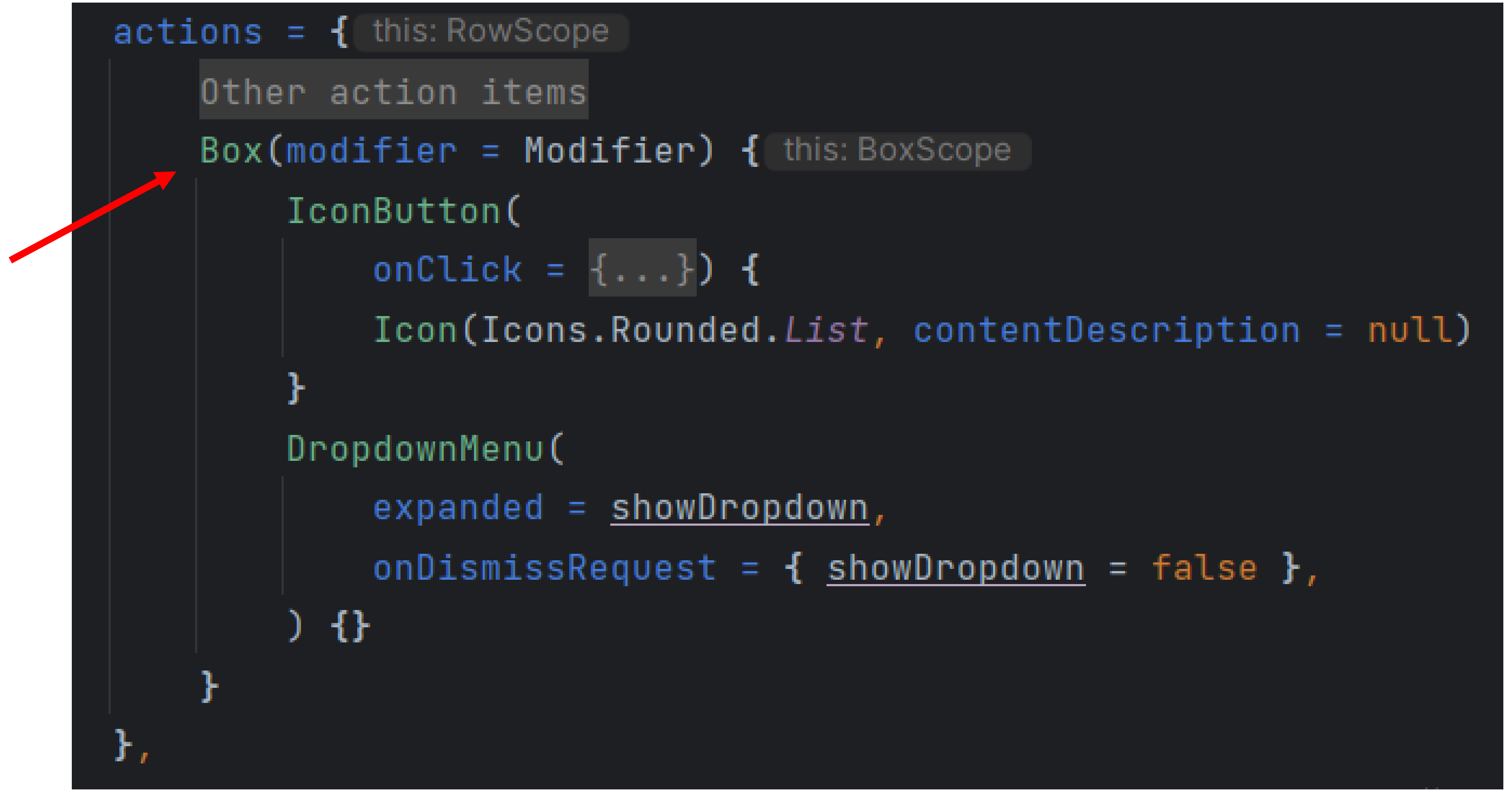
- A dropdown menu will be positioned/anchored to its parent



```
actions = { this: RowScope
    Other action items
    Box(modifier = Modifier) { this: BoxScope
        IconButton(
            onClick = {...}) {
            Icon(Icons.Rounded.List, contentDescription = null)
        }
        DropdownMenu(
            expanded = showDropdown,
            onDismissRequest = { showDropdown = false },
        ) {}
    }
},
```

下拉菜单（续）

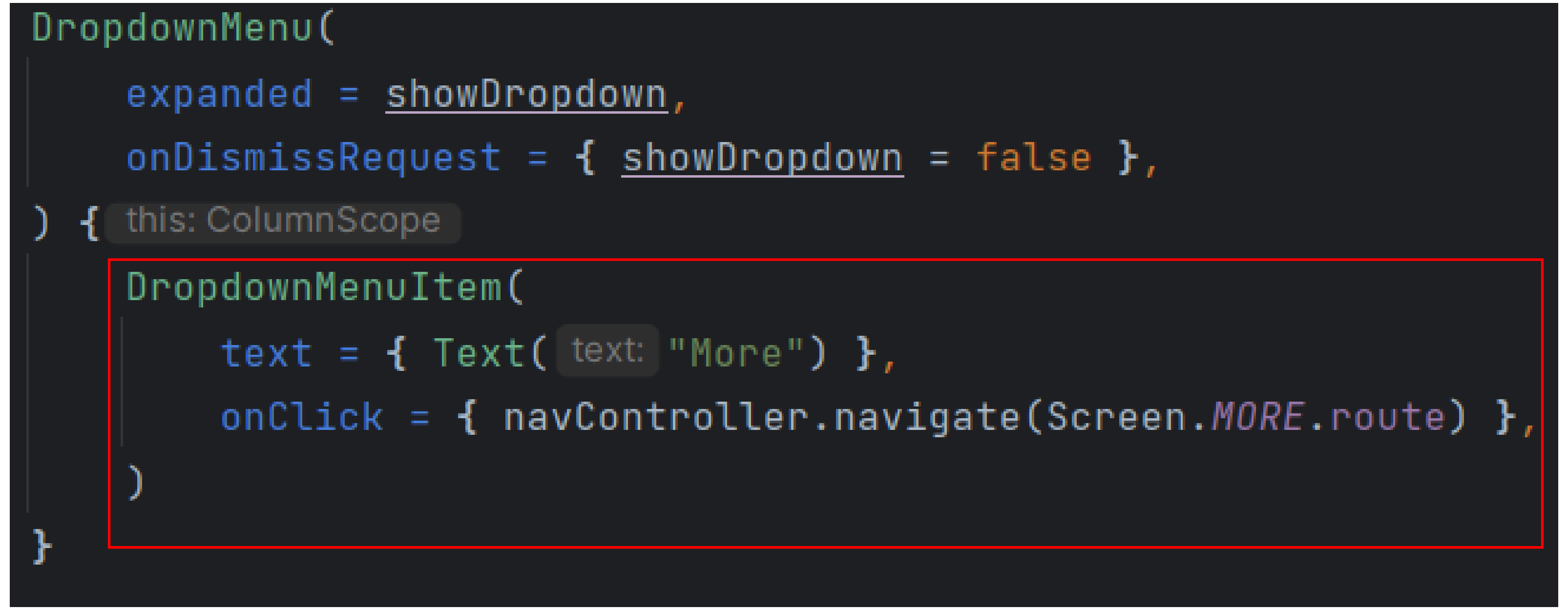
- 下拉菜单将相对于其 父级 进行定位/锚定



```
actions = { this: RowScope
    Other action items
    Box(modifier = Modifier) { this: BoxScope
        IconButton(
            onClick = {...}) {
            Icon(Icons.Rounded.List, contentDescription = null)
        }
        DropdownMenu(
            expanded = showDropdown,
            onDismissRequest = { showDropdown = false },
        ) {}
    }
},
```

Dropdown menu (cont.)

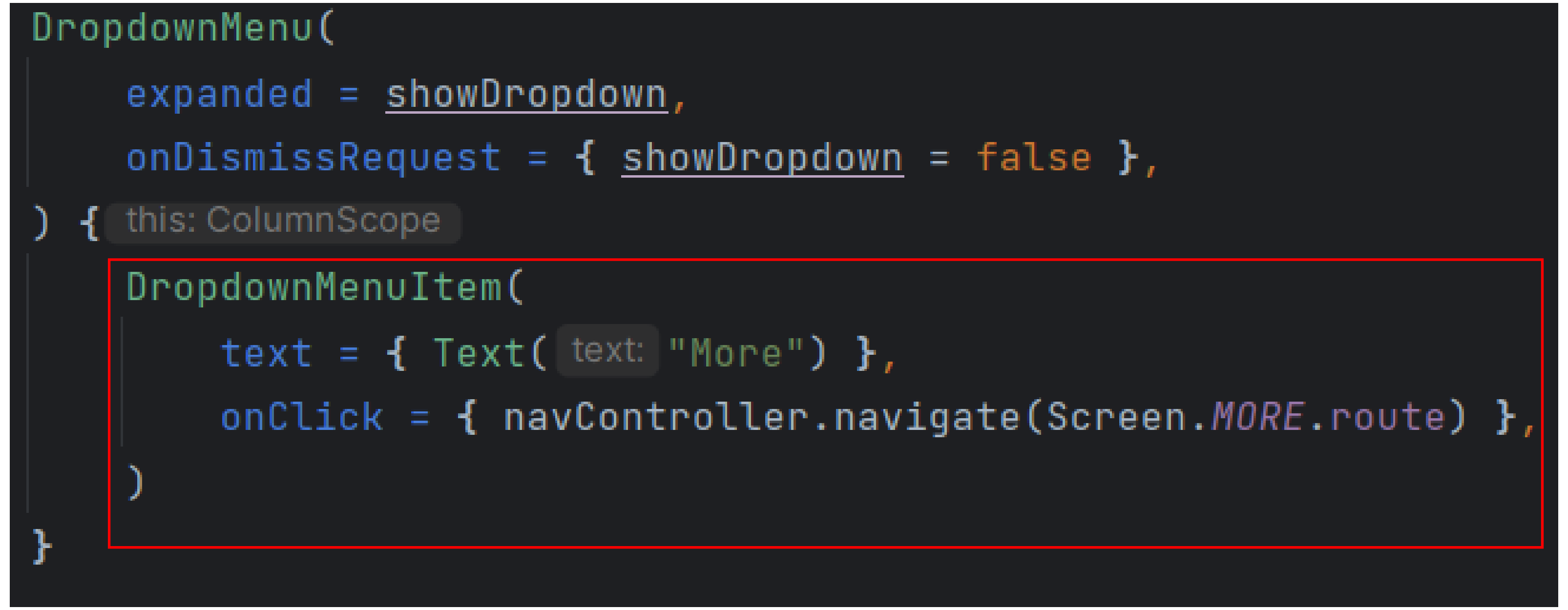
- You can add as many *DropDownMenuItem*'s as needed



```
DropdownMenu(
    expanded = showDropdown,
    onDismissRequest = { showDropdown = false },
) { this: ColumnScope
    DropdownMenuItem(
        text = { Text( text: "More") },
        onClick = { navController.navigate(Screen.MORE.route) },
    )
}
```

下拉菜单（续）

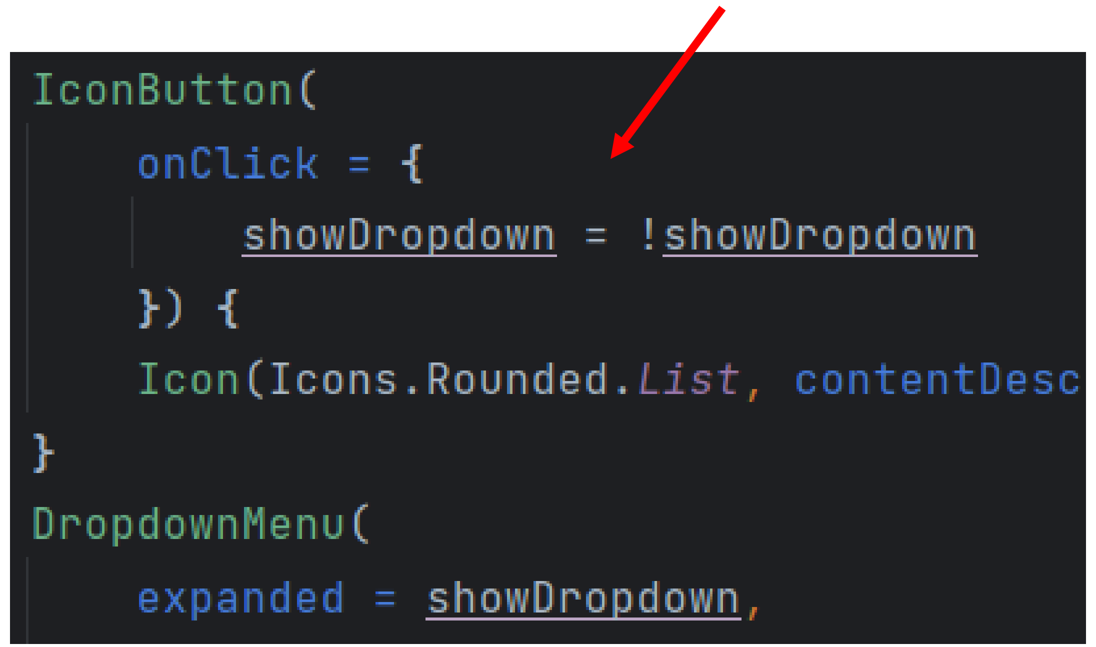
- 可以根据需要添加任意数量的 *DropDownMenuItem*项



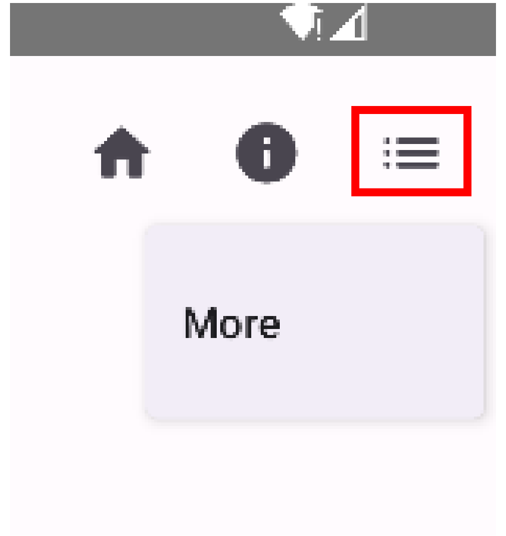
```
DropdownMenu(
    expanded = showDropdown,
    onDismissRequest = { showDropdown = false },
) { this: ColumnScope
    DropdownMenuItem(
        text = { Text( text: "More") },
        onClick = { navController.navigate(Screen.MORE.route) },
    )
}
```

Dropdown menu (cont.)

- Then trigger the dropdown through an event

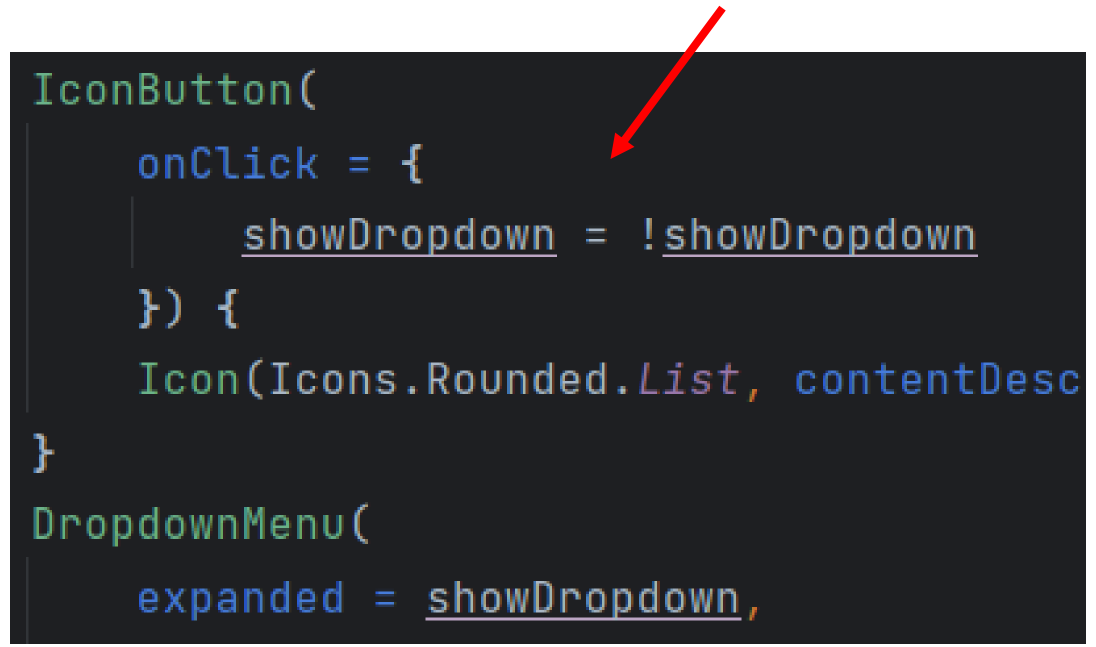


```
IconButton(
    onClick = {
        showDropdown = !showDropdown
    }) {
    Icon(Icons.Rounded.List, contentDesc
}
DropdownMenu(
    expanded = showDropdown,
```



下拉菜单（续）

- 然后通过事件触发下拉菜单



```
IconButton(
    onClick = {
        showDropdown = !showDropdown
    }) {
    Icon(Icons.Rounded.List, contentDesc
}
DropdownMenu(
    expanded = showDropdown,
```