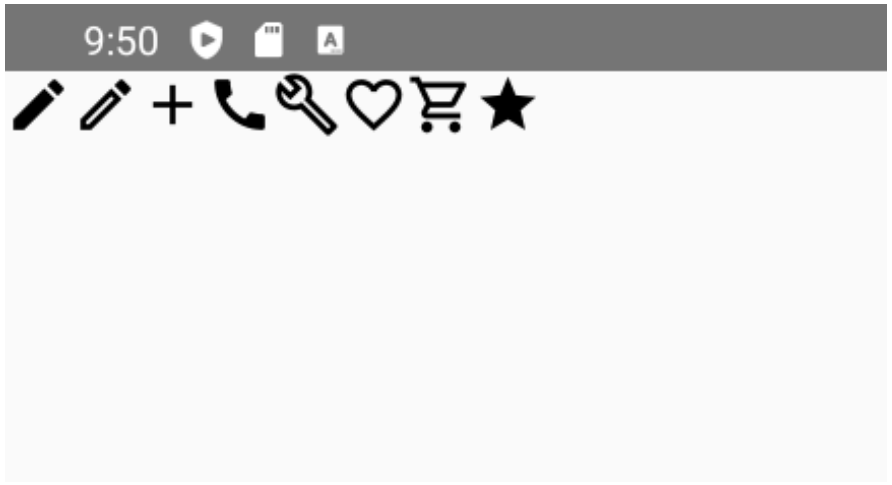
Lecture 8

COMP 3717- Mobile Dev with Android Tech

Icons

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

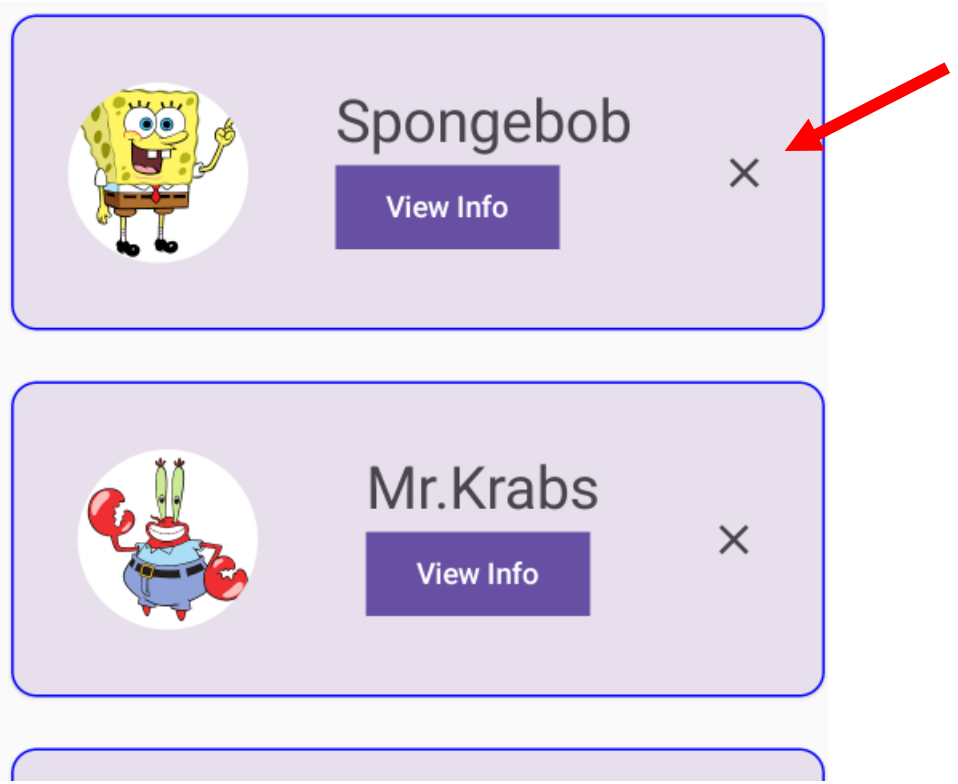
```
Row{ this: RowScope
    Icon(Icons.Filled.Create, contentDescription = null)
    Icon(Icons.Outlined.Create, contentDescription = null)
    Icon(Icons.Filled.Add, contentDescription = null)
    Icon(Icons.Filled.Call, contentDescription = null)
    Icon(Icons.Outlined.Build, contentDescription =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



IconButton

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

```
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 (cont.)

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

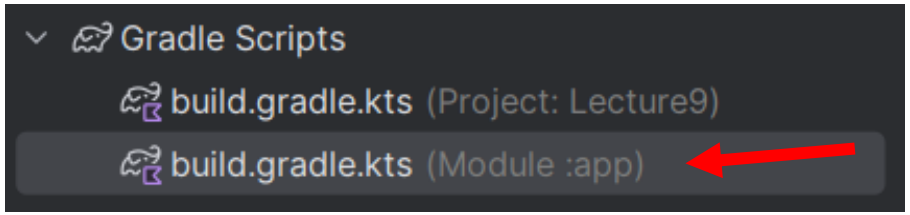
```
fun MyCard(
    cartoon: Cartoon,
    removeItemCallback:(Cartoon)->Unit,
){
```

```
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*




```
dependencies {

    implementation("androidx.navigation:navigation-compose:2.8.4")
```

2.8.8

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

Navigation controller

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

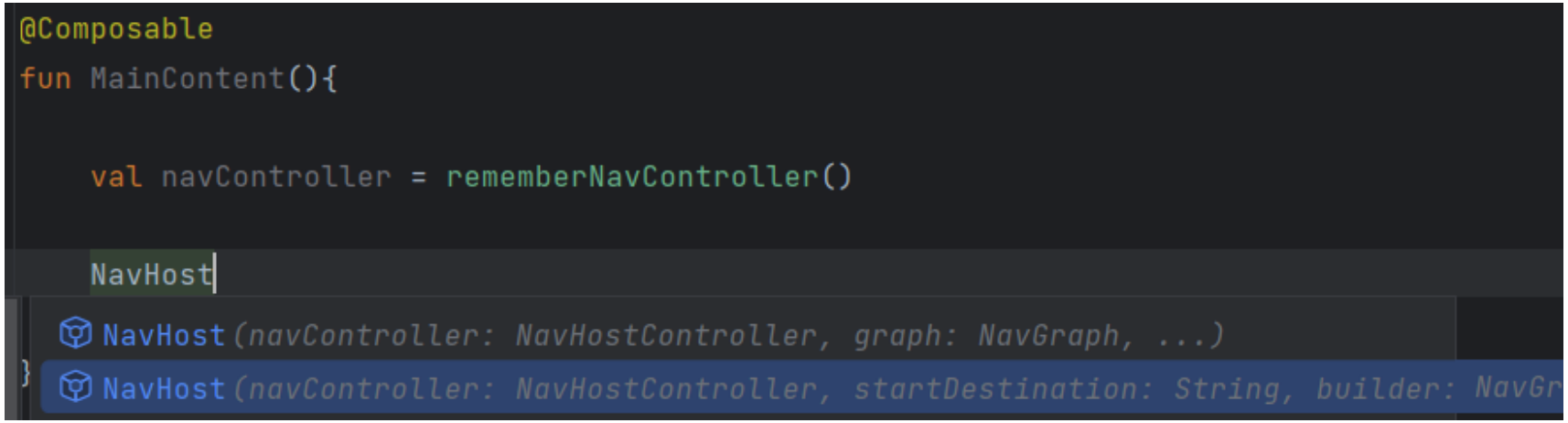
```
@Composable
fun MainContent(){

    val navController = rememberNavController()
```

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

Navigation host

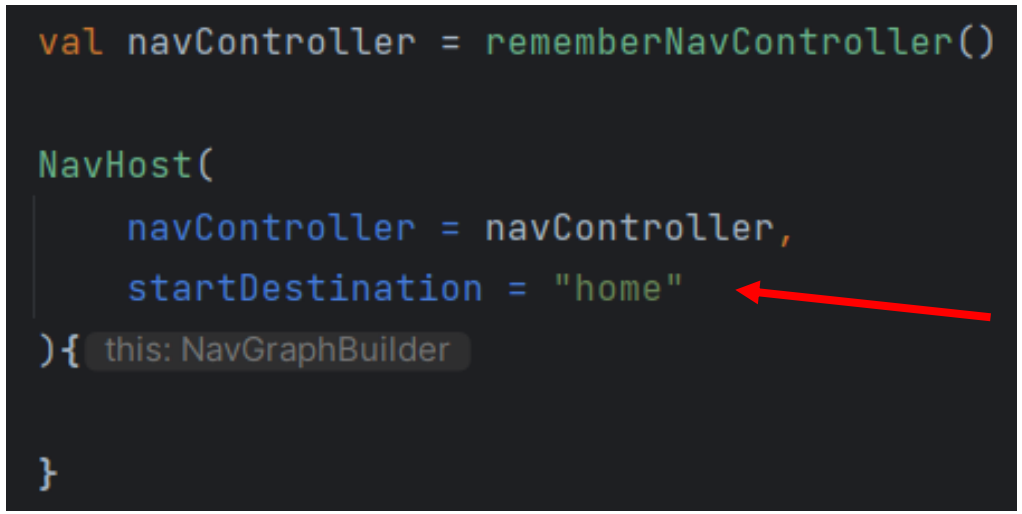
- A navigation host is a single container where destinations are swapped in and out

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



Navigation host (cont.)

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

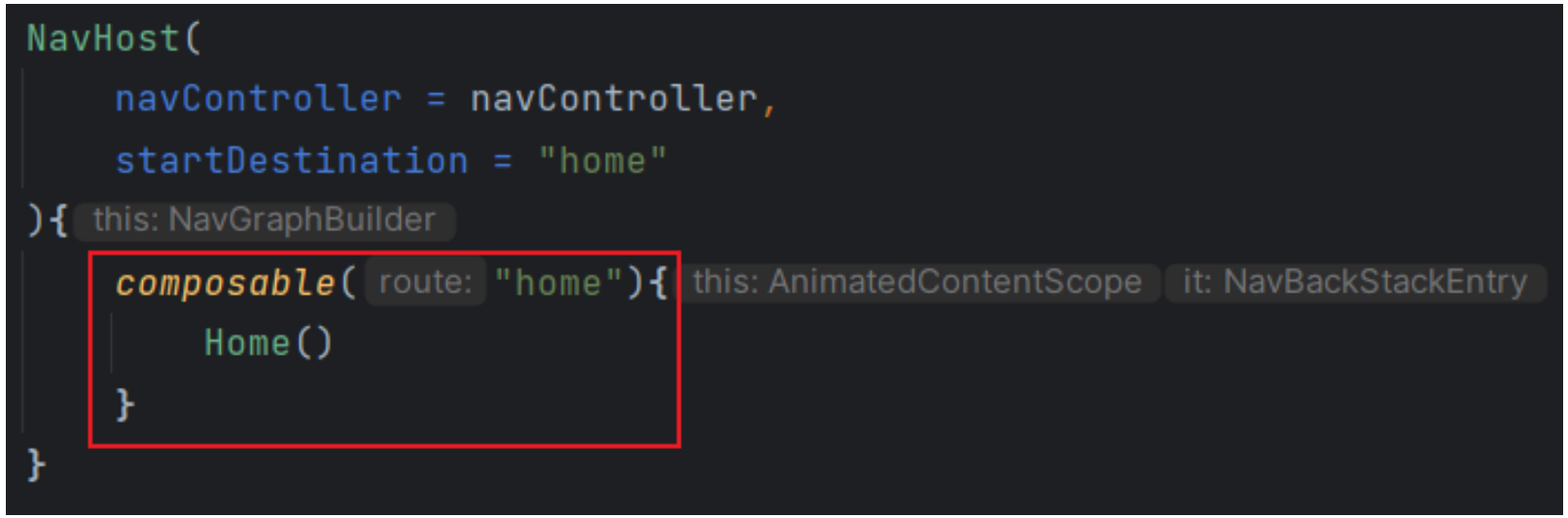


- Each destination is associated with a *route*
  - *route:* a string that defines the path to your composable

Navigation Graph

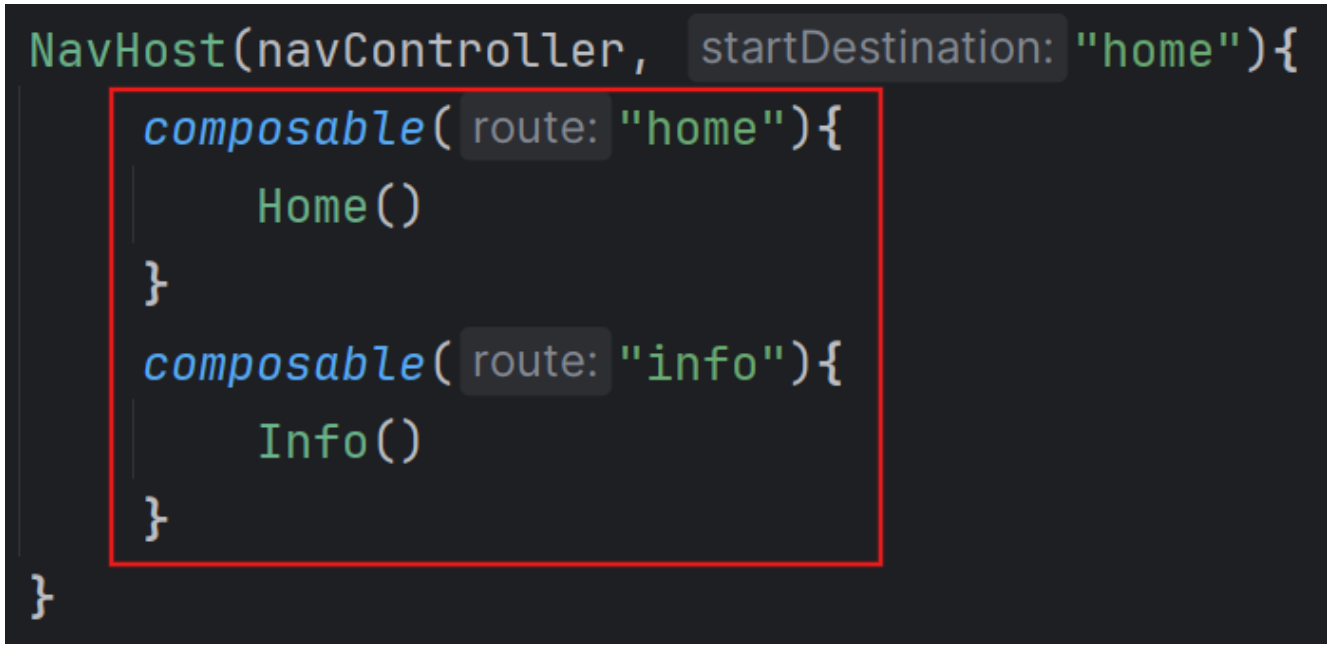
- The NavHost also links your navigation controller to your navigation graph

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



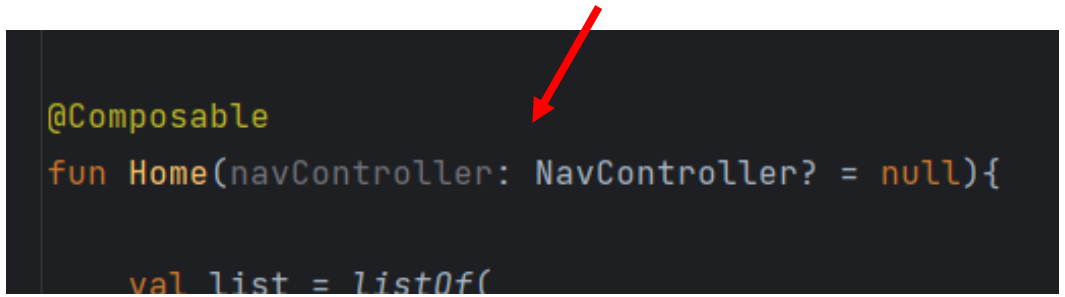
Navigation Graph (cont.)

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

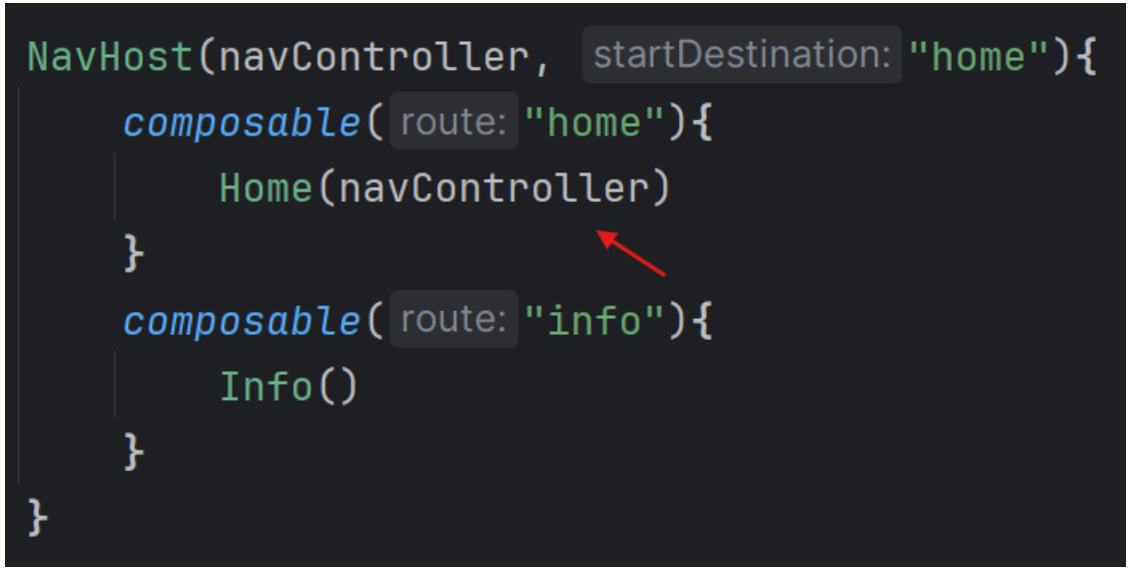


Navigation controller (cont.)

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



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

    val list = listOf(
```

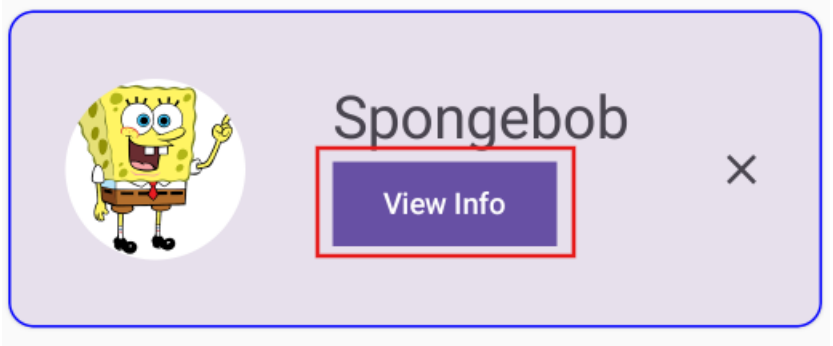


```
NavHost(navController, startDestination: "home"){
    composable( route: "home"){
        Home(navController)
    }
    composable( route: "info"){
        Info()
    }
}
```

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")
    }
```



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

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*

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

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)
}
```

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)
}
```

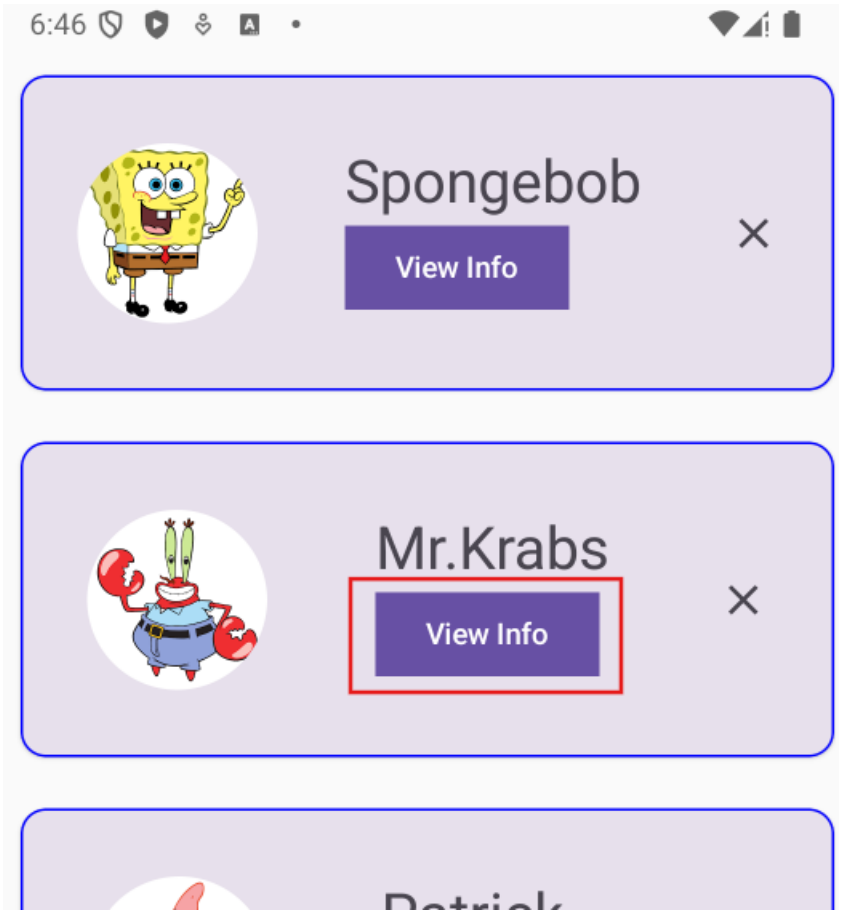
- If we want to pass a different type, we need to handle it appropriately

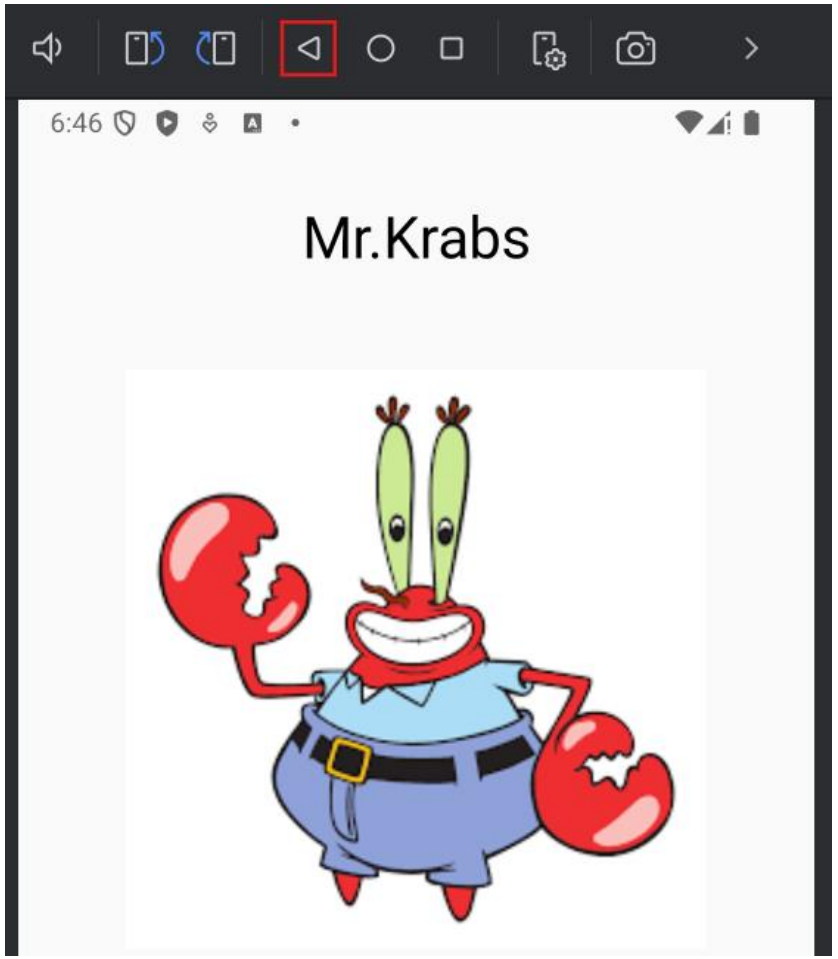
Pass multiple arguments with navigation (cont.)

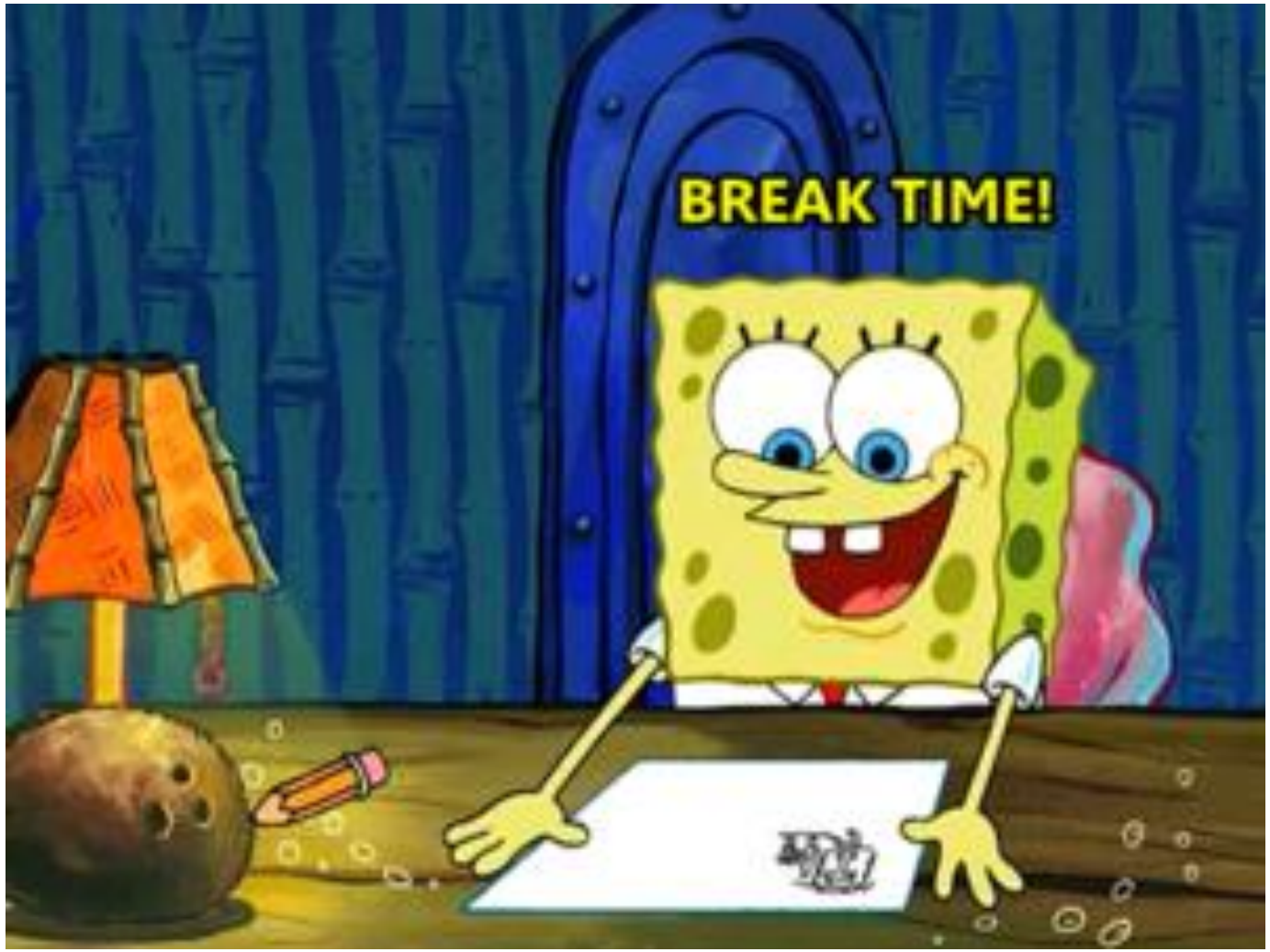
- We would again need to provide the arguments that we want to pass

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

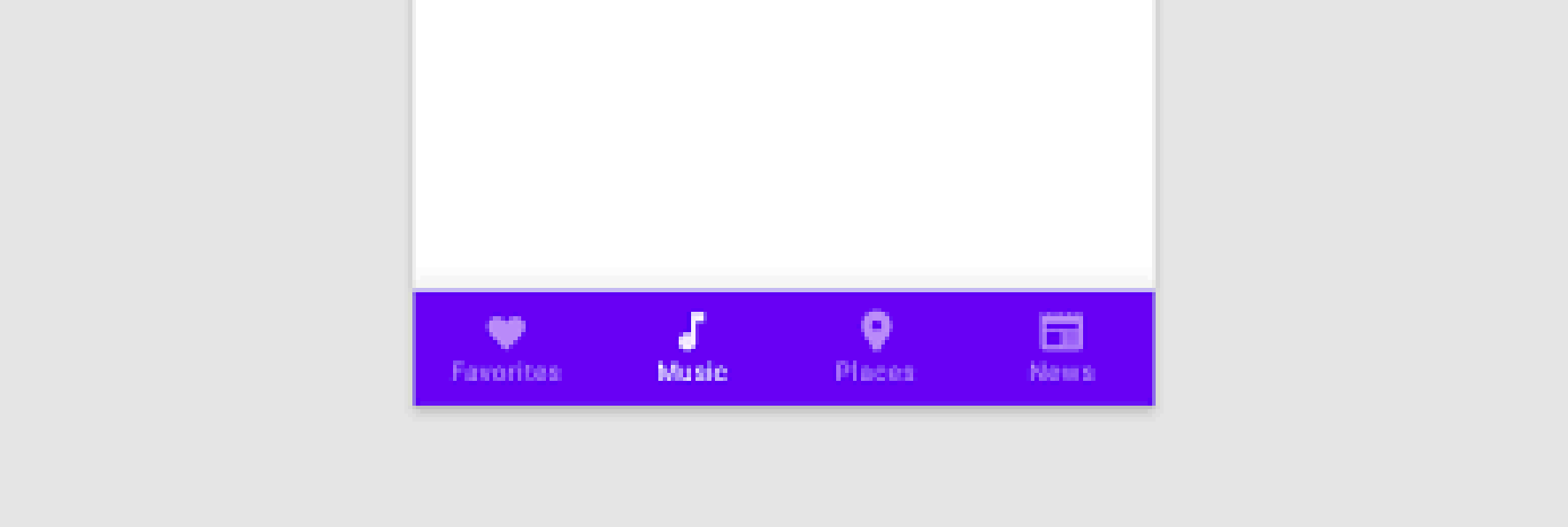
Pass multiple arguments with navigation (cont.)






BREAK TIME!

Bottom navigation bar



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)
    }
}
```

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")
)
```

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 {}

}
```

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
```

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()
}
```

Bottom navigation bar (cont.)

- Now we can get access to the current route

```
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

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

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)
        },
```

Bottom navigation bar (cont.)

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

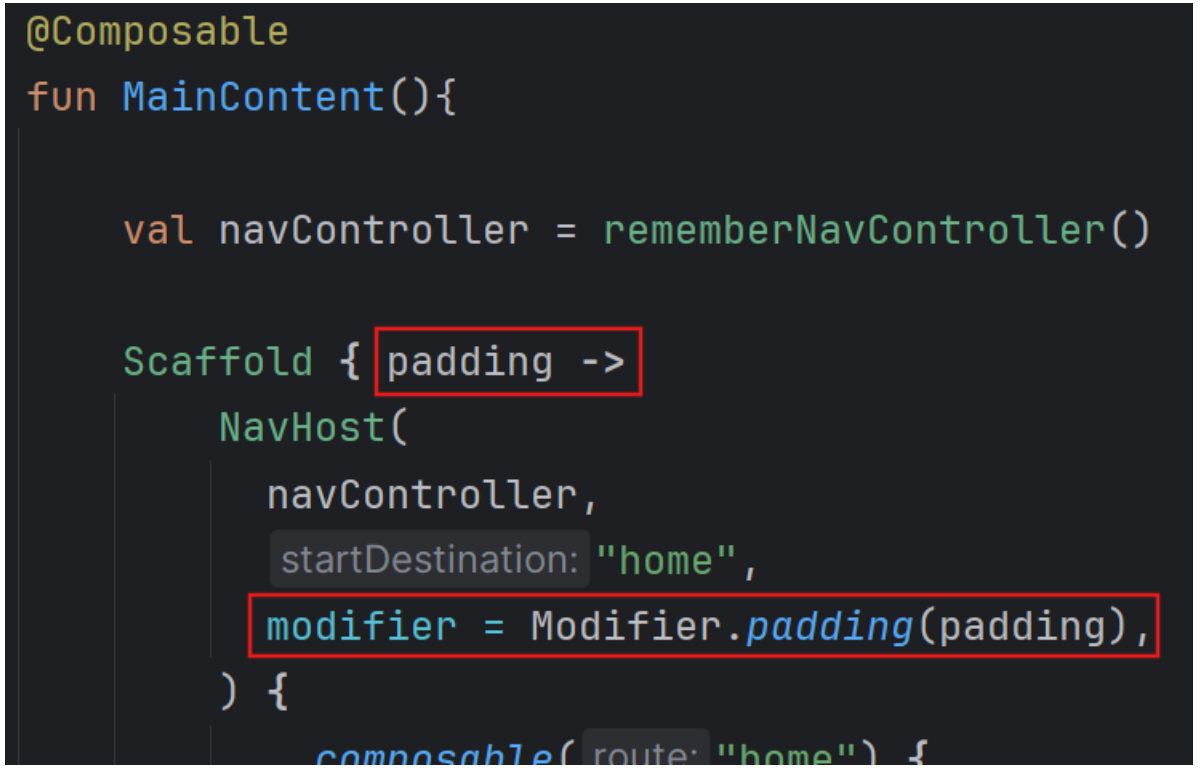
```
navItems.forEach{ item ->
    NavigationBarItem(
        selected = currentRoute == item.navRoute,
        onClick = {
            navController.navigate(item.navRoute)
        },
        icon = {
            Icon(item.icon, contentDescription = null)
        },
    )
```

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

Scaffold (cont.)

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

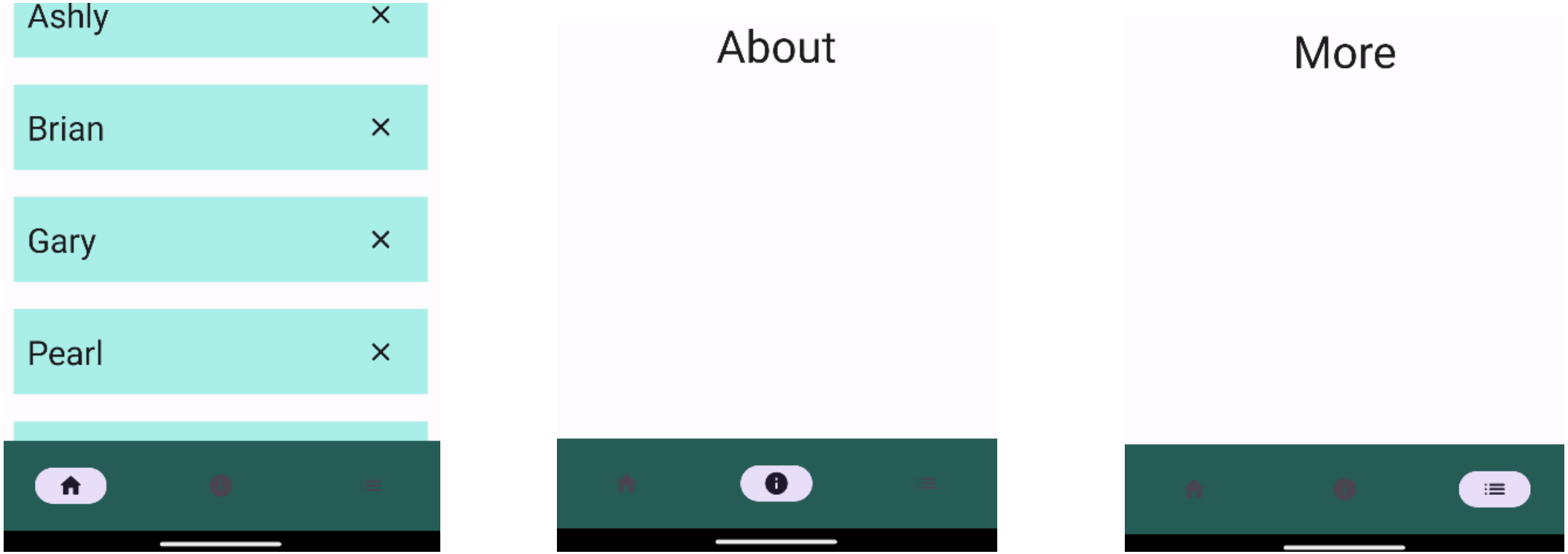


Scaffold (cont.)

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

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

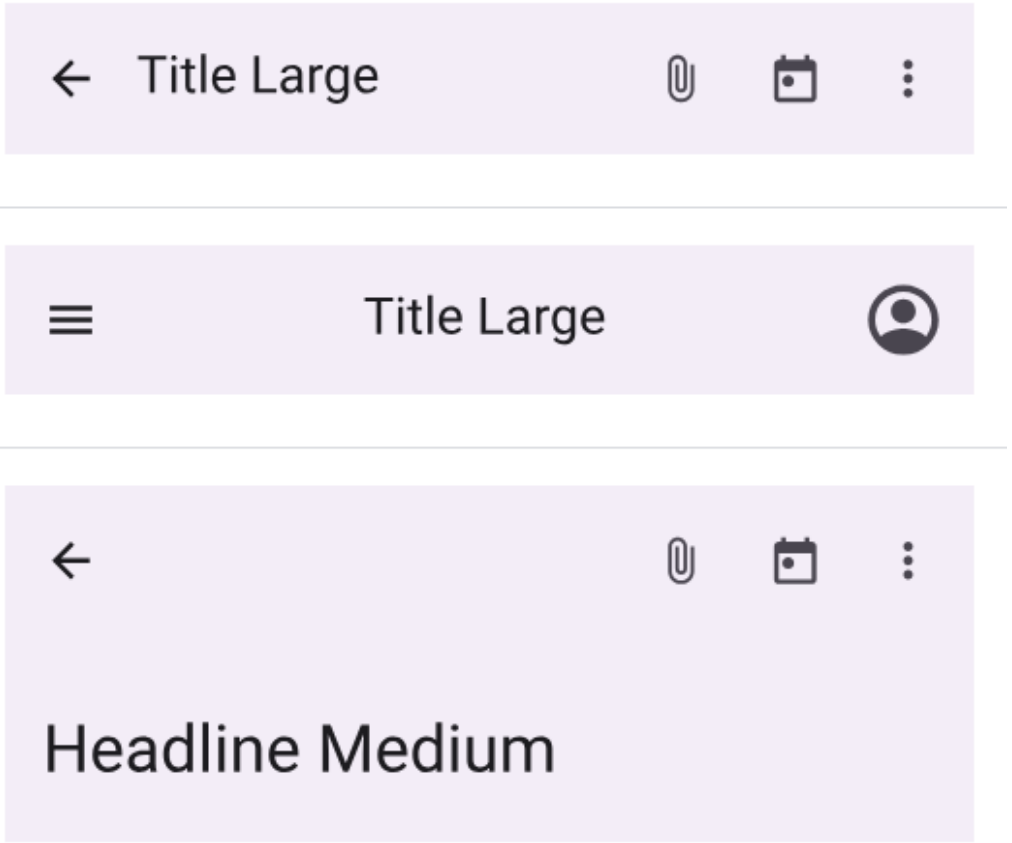
Bottom navigation bar (cont.)



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

Top app bar

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



Top app bar (cont.)

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

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")
        }
    )
}
```

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

Top app bar (cont.)

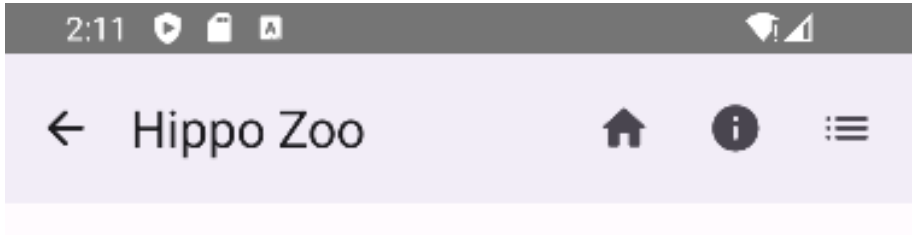
- *actions* are a row of composables that sit on the right side of the top app bar

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

Top app bar (cont.)

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

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



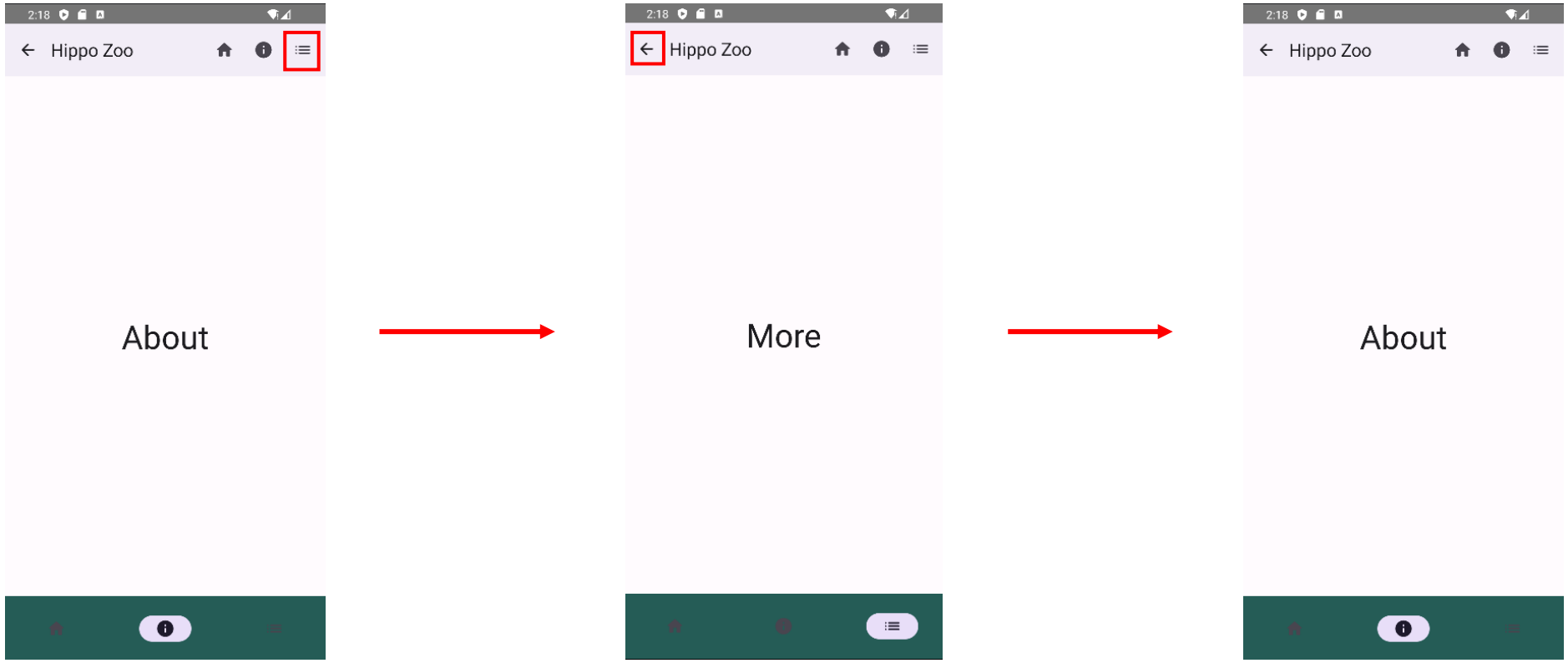
Top app bar (cont.)

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

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

Navigation (cont.)

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

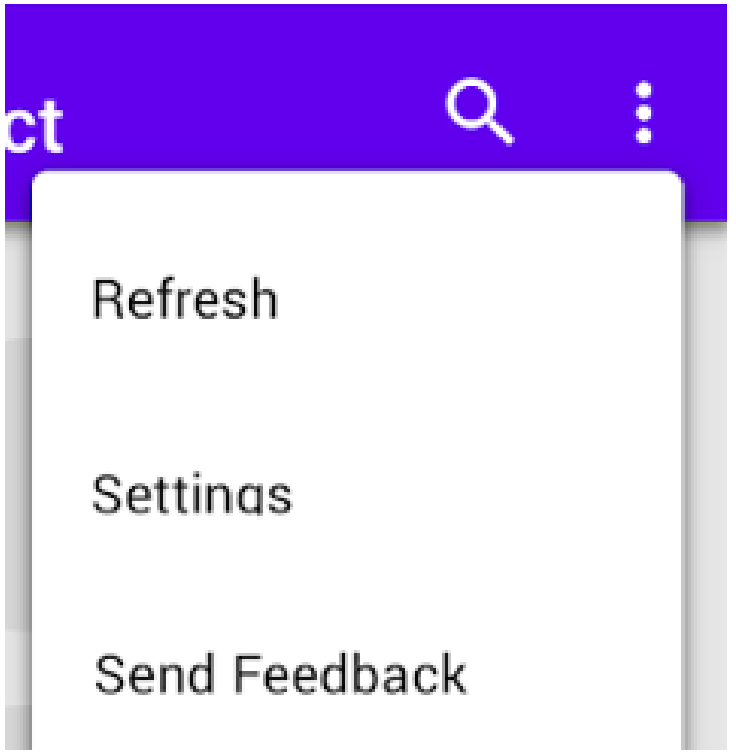


Dropdown menu

- 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 },
) {}
```



Dropdown menu (cont.)

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

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) },
    )
}
```

Dropdown menu (cont.)

- Then trigger the dropdown through an event

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