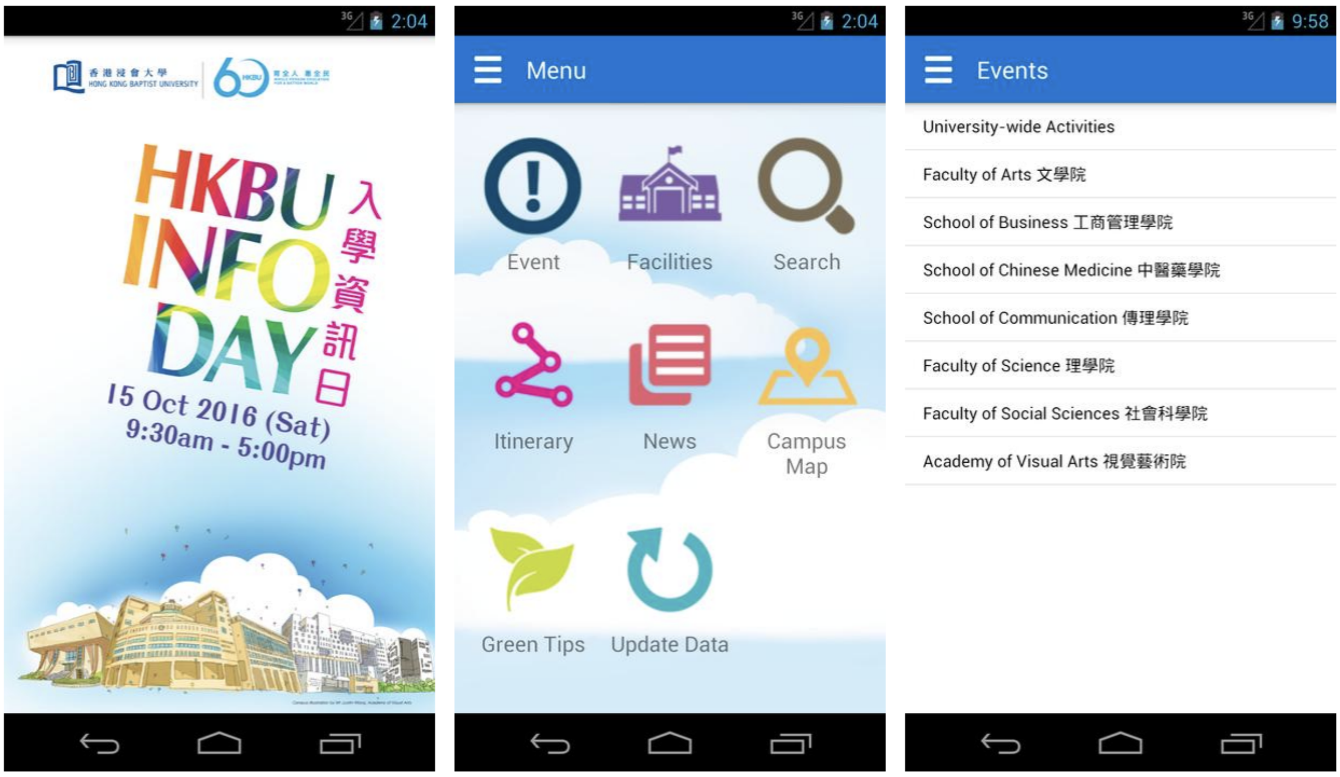
Lab 7 - Jetpack Compose

`COMP4107 - SDDT - HKBU - Spring2023`


2:04
HKBU INFO DAY 入學資訊日
15 Oct 2016 (Sat)
9:30am - 5:00pm
Menu
Event
Facilities
Search
Itinerary
News
Campus Map
Green Tips
Update Data
9:58
Events
University-wide Activities
Faculty of Arts 文學院
School of Business 工商管理學院
School of Chinese Medicine 中醫藥學院
School of Communication 傳理學院
Faculty of Science 理學院
Faculty of Social Sciences 社會科學院
Academy of Visual Arts 視覺藝術院

In upcoming labs, we'll build an Android app called `InfoDay` using **Android Studio** and **Jetpack Compose**, Android's modern toolkit for building native UI.

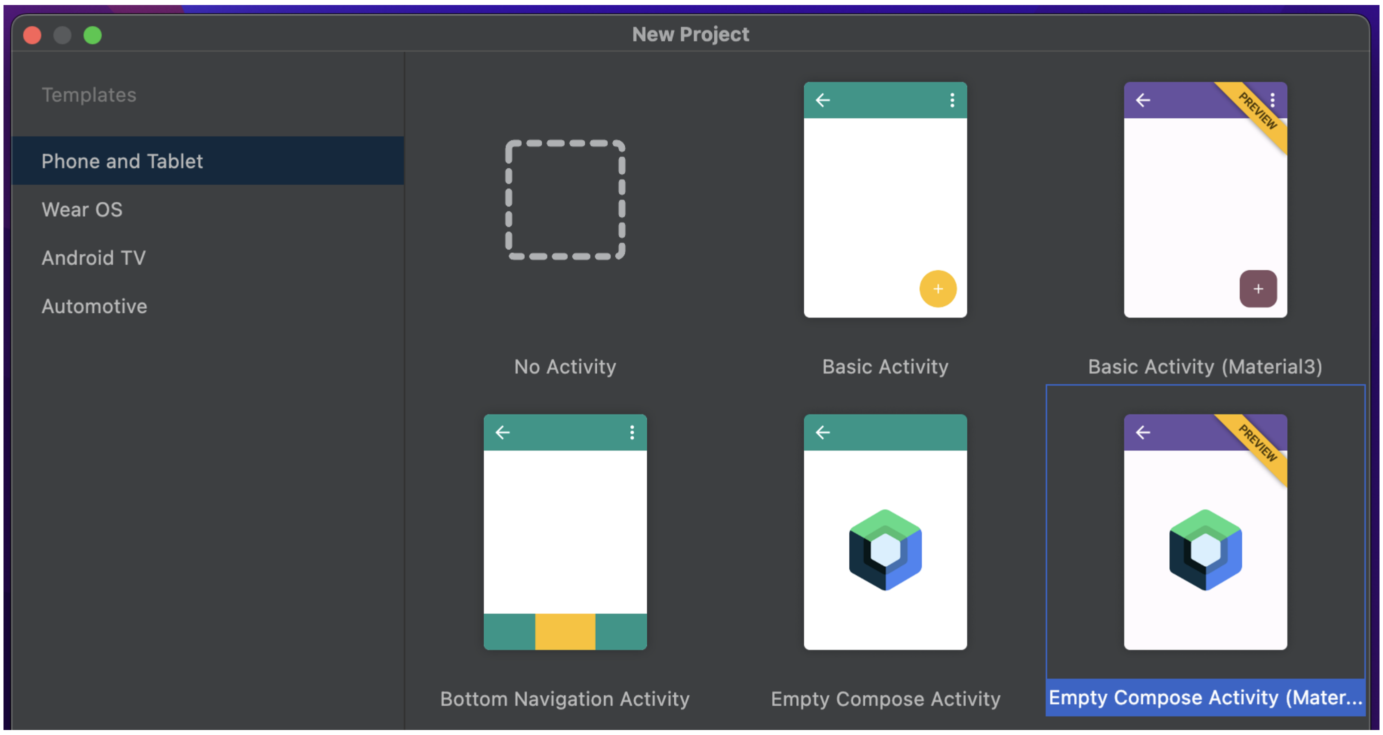
Jetpack Compose simplifies and speeds up UI development on Android. Using Compose, you can **define your UI programmatically instead of using XML layout files**.

Compose uses a **declarative syntax**, so you describe what your UI should look like and do, rather than defining steps to build the UI.

Getting Started

Native Android applications can be developed using **Android Studio** (https://developer.android.com/studio). Launch it, upgrade it to the latest version. Restart the IDE after upgrade.

Next, create a `New Project` and select `Empty Compose Activity (Material 3)` under `Phone and Tablet`.



Name the project `InfoDay` and use the default settings. It will take a while for **Gradle** to download all dependencies.

You will receive a file called `MainActivity.kt`, which is the starting point of your app. The code in `MainActivity.kt` is as follows:

```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            InfoDayTheme {
                // A surface container using the 'background' color from the theme
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color = MaterialTheme.colorScheme.background
                ) {
                    Greeting("Android")
                }
            }
        }
    }
}

@Composable
fun Greeting(name: String) {
    Text(text = "Hello $name!")
}
```

```
@Preview(showBackground = true)
@Composable
fun DefaultPreview() {
    InfoDayTheme {
        Greeting("Android")
    }
}
```

Composable Function

While the entry point of this app is the `onCreate()` function, the main focus of development will be on **composable functions**.

```
@Composable
fun Greeting(name: String) {
    Text(text = "Hello $name!")
}
```

Composable functions (also called **composables** or **components**) are special Kotlin functions marked with the `@Composable` annotation. They are **used to build user interfaces with Compose**.

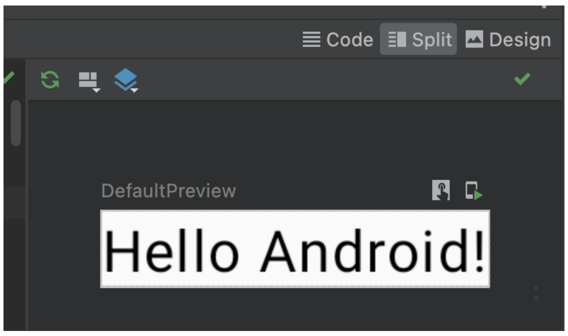
A composable function differs from a regular Kotlin function in that it can be called from Compose. Composable functions allow you to define your UI in a declarative, modular style.

Preview

Jetpack Compose provides **instant UI previews**. Add the `@Preview()` annotation before a composable function to see a live preview of the UI.

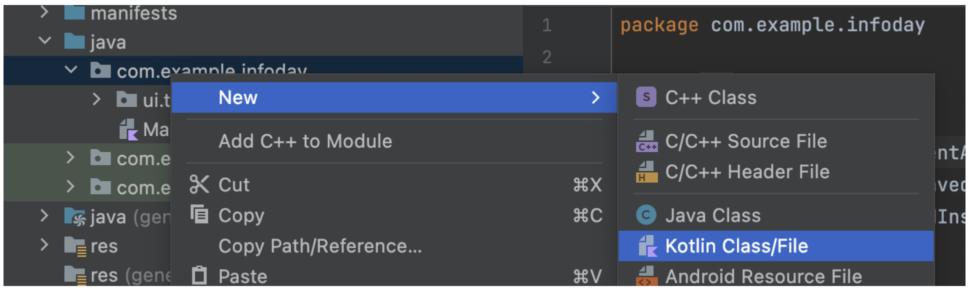
```
@Preview(showBackground = true)
@Composable
fun DefaultPreview() {
    InfoDayTheme {
        Greeting("Android")
    }
}
```

To preview your app, select the `Split` mode in the top-right corner of Android Studio. Then click `Build and Refresh`. You should see a preview like this:

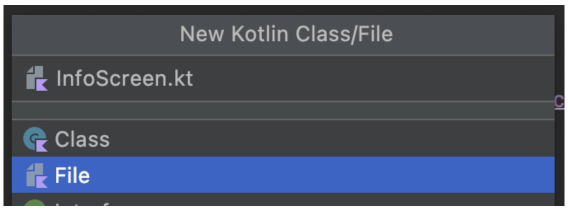


Info Screen

Next, right-click on `com.example.infoday` and create a **new Kotlin file**. Name this file `InfoScreen.kt`.



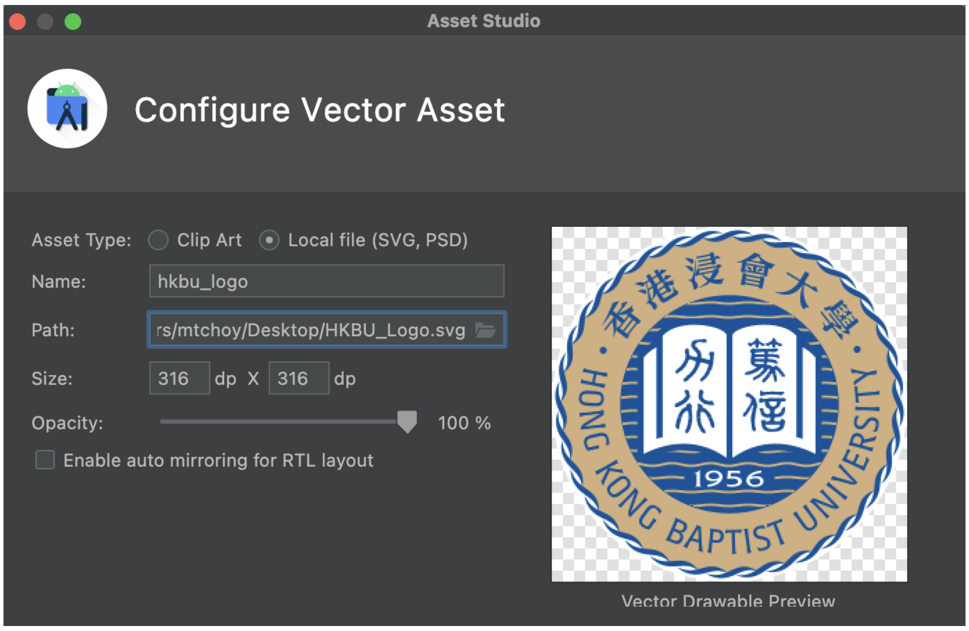
```
package com.example.infoday
```



Image

Download the following **HKBU logo** in **SVG** format:

Back in Android Studio, right-click on `com.example.infoday` and select `Vector Asset`. Choose `Local File` then select the SVG file.



Asset Type: Clip Art [✓ Local file (SVG, PSD)]
Name: hkbu_logo
Path: rs/mtchoy/Desktop/HKBU_Logo.svg
Size: 316 dp X 316 dp
Opacity: 100 %
Enable auto mirroring for RTL layout

The vector graphic will be referred to as `hkbu_logo`. Next, develop a **composable function** to display this logo along with text below it:

```
@Composable
fun InfoGreeting() {
    Image(
        painter = painterResource(id = R.drawable.hkbu_logo),
        contentDescription = stringResource(id = R.string.hkbu_logo),
    )
    Text(text = "Hello Android!")
}
```

The Image requires importing `androidx.compose.foundation.Image`.

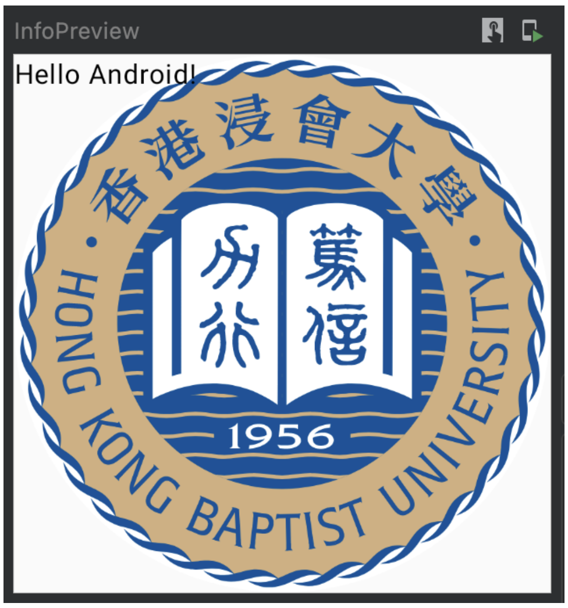
To preview, develop a **preview function**:

```
@Preview(showBackground = true)
@Composable
fun InfoPreview() {
    InfoDayTheme {
        InfoGreeting()
    }
}
```

While the `R.drawable.hkbu_logo` vector image is imported, the `R.string.hkbu_logo` string resource is currently undefined. Let's create it under `res` -> `values` -> `string.xml`. Add a new string resource as follows:

```
<resources>
    <string name="app_name">InfoDay</string>
    <string name="hkbu_logo">HKBU logo</string>
</resources>
```

A preview will then be shown.



Hello Android!

The image and text are stacking together. To develop a vertical layout, we should place the elements inside a `Column`. Revise the code as follows:

```
@Composable
fun InfoGreeting() {
    Column {
        Image(
            painter = painterResource(id = R.drawable.hkbu_logo),
            contentDescription = stringResource(id = R.string.hkbu_logo),
        )
        Text(text = "Hello Android!")
    }
}
```

You can use `Ctrl`+`Alt`+`L` on Windows or `Option`+`Command`+`L` on Mac for **code formatting**.

Spacers and Modifiers

Finally, we can enhance the layout with some **spacers** and **modifiers**:

```
@Composable
fun InfoGreeting() {
    val padding = 16.dp
    Column(horizontalAlignment = Alignment.CenterHorizontally) {
        Spacer(Modifier.size(padding))
        Image(
            painter = painterResource(id = R.drawable.hkbu_logo),
            contentDescription = stringResource(id = R.string.hkbu_logo),
            modifier = Modifier.size(240.dp)
        )
        Spacer(Modifier.size(padding))
        Text(text = "HKBU InfoDay App", fontSize = 30.sp)
    }
}
```

List

On the same file, develop the following `Contact` data class and the corresponding data:

```
data class Contact(val office: String, val tel: String) {
    companion object {
        val data = listOf(
            Contact(office = "Admission Office", tel = "3411-2200"),
            Contact(office = "Emergencies", tel = "3411-7777"),
            Contact(office = "Health Services Center", tel = "3411-7447")
        )
    }
}
```

If you know your use case **does not require scrolling**, you may want to use a simple `Column` or `Row` (depending on direction) and populate each item by **iterating over a list**, like this:

```
@Composable
fun PhoneList() {
    Column {
        Contact.data.forEach { message ->
            Text(message.office)
        }
    }
}
```

Then, modify the preview composable as follows:

```
@Preview(showBackground = true)
@Composable
fun InfoPreview() {
    InfoDayTheme {
        Column {
            InfoGreeting()
            PhoneList()
        }
    }
}
```

Lambda Expression

In Kotlin, an **unnamed function** used primarily as an argument to a function call is called a **lambda expression**. Here is an example:

```
{ message ->
    Text(message.office)
}
```

Material Design 3 and List Item

Material Design 3 is the latest **specification of Material Design by Google**. It provides updated guidance on visual, motion, and interaction design across platforms and devices.

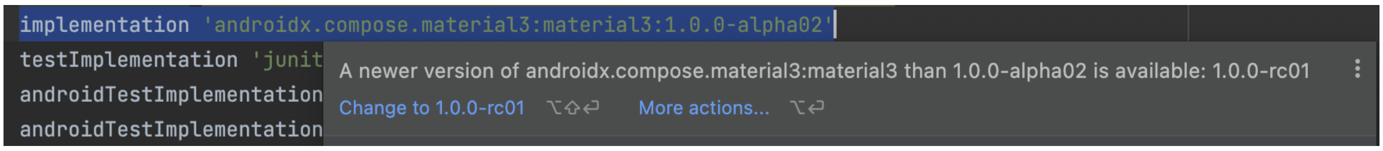
Material Design 3 provides a helpful `ListItem` component (Reference): (For reference only, don't copy)

```
@Composable
@ExperimentalMaterial3Api
fun ListItem(
    headlineText: @Composable () -> Unit,
    modifier: Modifier = Modifier,
    overlineText: (@Composable () -> Unit)? = null,
    supportingText: (@Composable () -> Unit)? = null,
    leadingContent: (@Composable () -> Unit)? = null,
    trailingContent: (@Composable () -> Unit)? = null,
    colors: ListItemColors = ListItemDefaults.colors(),
    tonalElevation: Dp = ListItemDefaults.Elevation,
    shadowElevation: Dp = ListItemDefaults.Elevation
): Unit
```

Here, we only use the `headlineText`, `leadingContent`, and `trailingContent` settings as follow:

```
@Composable
fun PhoneList() {
    Column {
        Contact.data.forEach { message ->
            ListItem(
                headlineText = { Text(message.office) },
                leadingContent = {
                    Icon(
                        Icons.Filled.Call,
                        contentDescription = null
                    )
                },
                trailingContent = { Text(message.tel) }
            )
        }
    }
}
```

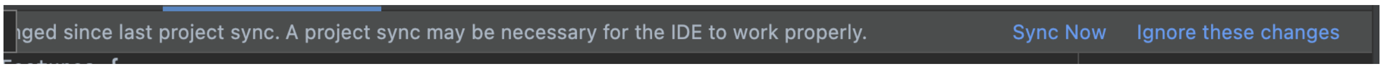
In case of errors, it's likely the Material Design dependency needs updating. Let's go to the **module-level Gradle file**, hover over the `material3` dependency and upgrade it to version `1.0.0-rc01` or `1.0.1`.



```
implementation 'androidx.compose.material3:material3:1.0.0-alpha02'
testImplementation 'junit
androidTestImplementation
androidTestImplementation
```

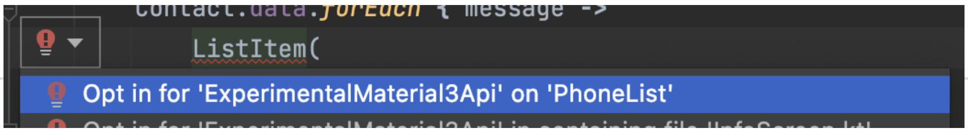
A newer version of androidx.compose.material3:material3 than 1.0.0-alpha02 is available: 1.0.0-rc01

Hit `Sync Now` to download the latest dependency updates.



ged since last project sync. A project sync may be necessary for the IDE to work properly.

There could also be an error stating this API is **experimental**. Locate the **red lightbulb** next to `ListItem` and `Opt in`:

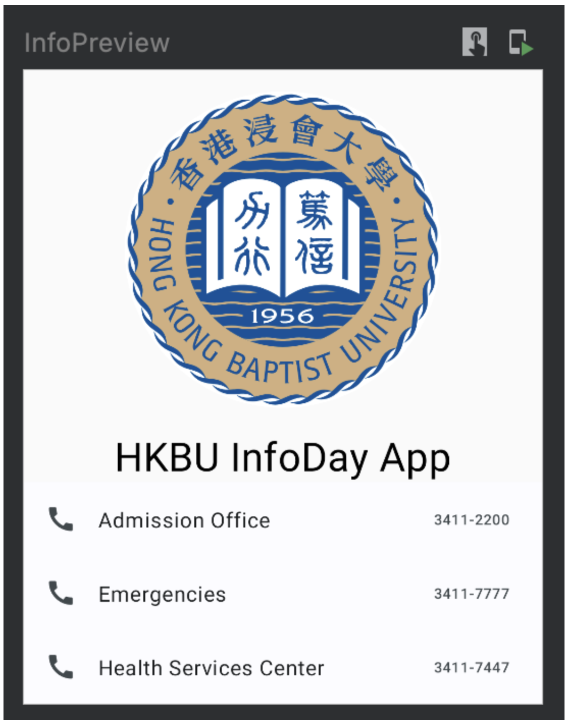


```
ListItem(
```

Info Screen

Finally, let's construct a new composable as follows:

```
@Composable
fun InfoScreen() {
    Column(horizontalAlignment = Alignment.CenterHorizontally) {
        InfoGreeting()
        PhoneList()
    }
}
```



HKBU InfoDay App

Admission Office 3411-2200

Emergencies 3411-7777

Health Services Center 3411-7447

Update preview composable as follows:

```
@Preview(showBackground = true)
@Composable
fun InfoPreview() {
    InfoDayTheme {
        InfoScreen()
    }
}
```

Scaffold

Head back to `MainActivity.kt` and we will develop a [Scaffold](), which implements the basic **material design visual layout structure**.

This component provides API to **assemble multiple material components** to construct your screen. It ensures proper layout and gathers necessary data so these components work together correctly. **For reference only, don't copy**:

```
@ExperimentalMaterial3Api
@Composable
fun Scaffold(
    modifier: Modifier = Modifier,
    topBar: @Composable () -> Unit = {},
    bottomBar: @Composable () -> Unit = {},
    snackbarHost: @Composable () -> Unit = {},
    floatingActionButton: @Composable () -> Unit = {},
    floatingActionButtonPosition: FabPosition = FabPosition.End,
    containerColor: Color = MaterialTheme.colorScheme.background,
    contentColor: Color = contentColorFor(containerColor),
    contentWindowInsets: WindowInsets = ScaffoldDefaults.contentWindowInsets,
    content: @Composable (PaddingValues) -> Unit
): Unit
```

We will develop a `topBar`, a `bottomBar` and of course the `content`. Here's our `ScaffoldScreen()`

```
@Composable
fun ScaffoldScreen() {

    Scaffold(
        topBar = {
            TopAppBar(
                title = { Text("HKBU InfoDay App") }
            )
        },
        bottomBar = {},
        content = { innerPadding ->
            Column(
                modifier = Modifier.padding(innerPadding),
            ) {}
        }
    )
}
```

Again, we **must** enable the `ExperimentalMaterial3Api`.

Navigation Bar

Navigation bars provide a consistent way to navigate between the main sections of an app. To add one, include the following within the `ScaffoldScreen` composable:

```
var selectedItem by remember { mutableStateOf(0) }
val items = listOf("Home", "Events", "Itin", "Map", "Info")
```

Then, use the following **navigation bar** as our **bottom bar**:

```
NavigationBar {
    items.forEachIndexed { index, item ->
        NavigationBarItem(
            icon = { Icon(Icons.Filled.Favorite, contentDescription = item) },
            label = { Text(item) },
            selected = selectedItem == index,
            onClick = { selectedItem = index }
        )
    }
}
```

You need to import `androidx.compose.runtime.*`.

`remember { mutableStateOf() }`

Reference: https://dev.to/zachklipp/remember-mutablestateof-a-cheat-sheet-10ma

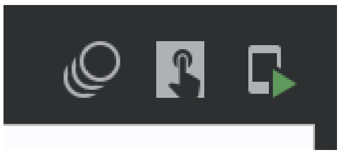
- `remember` keeps a value (any value) **consistent across recompositions**. "Recomposition" means when a composable function is called multiple times to update the UI.
- `mutableStateOf` returns a `MutableState`.
- `MutableState` is just a thing that holds a value, where Compose will **automatically observe changes to the value**.

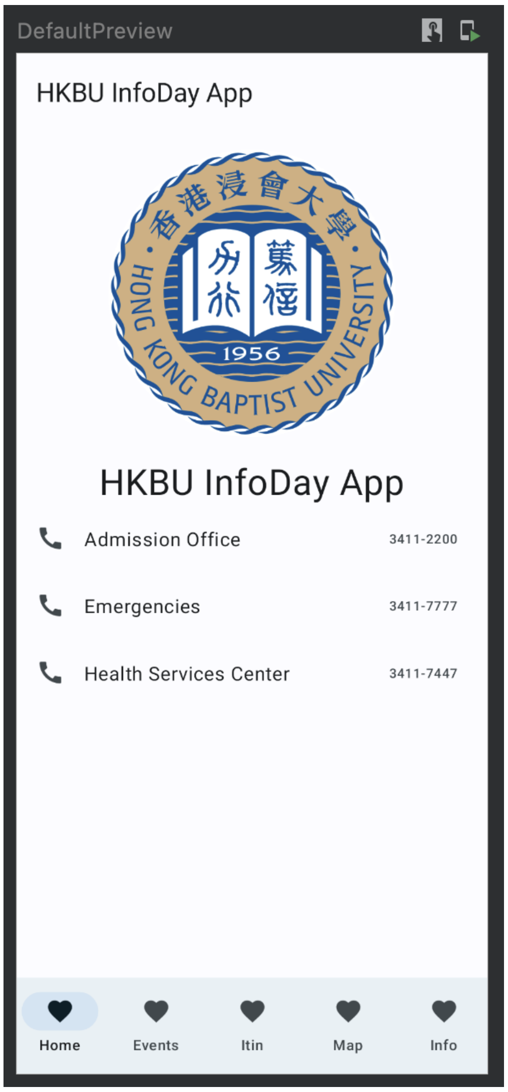
Content

Here, `selectedItem` holds the chosen item. Using a `when()` statement on `selectedItem`, we can switch to different screens.

```
content = { innerPadding ->
    Column(
        modifier = Modifier.padding(innerPadding),
    ) {
        when (selectedItem) {
            0 -> InfoScreen()
            1 -> InfoScreen()
            2 -> InfoScreen()
            3 -> InfoScreen()
            4 -> InfoScreen()
        }
    }
}
```

In the `DefaultPreview()` composable, replace `Greeting("Android")` with `ScaffoldScreen()`. Run it in **interactive mode** and tap the navigation tabs.





You may have to create a new virtual device. Choose `Pixel 4` with `R` as the OS.

Version Control

Claim your **private repo** at https://classroom.github.com/a/t86LHCCf.

To enable Git, go to `VCS` -> `Enable Version Control Integration...`.

Then go to `VCS` -> `Git` -> `Remotes` to add your **GitHub private repo** URL as `origin`. You can then add, commit, and push your work to GitHub. We will continue maintaining source code on GitHub for upcoming Android Labs.

27/02/2023 13:43