04

04-4. 다양한 Composable

Compose

Text

- 리소스 문자열 출력

```
Text(stringResource(R.string.app_name))
```

- color / fontSize / fontStyle / fontWeight

```
Text(
    "HelloWorld",
    color = Color.Red,
    fontSize = 20.sp,
    fontStyle = FontStyle.Italic,
    fontWeight = FontWeight.Bold
)
```

- style 속성으로도 가능하다.

```
Text(
    "HelloWorld",
    style = TextStyle(
        color = Color.Red,
        fontSize = 20.sp,
        fontStyle = FontStyle.Italic,
        fontWeight = FontWeight.Bold
    )
```

Text

- 그림자 효과

```
Text(
    "HelloWorld",
    style = TextStyle(
      shadow = Shadow(color = Color.Blue, offset = Offset(5.0f, 5.0f),
blurRadius = 3f)
    )
)
```

Text

- 하나의 Text 컴포저블에 여러 스타일 추가
- AnnotatedStyle 을 buildAnnotatedString 으로 만들어서 적용해야 한다.

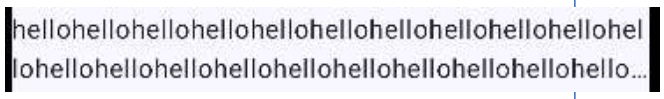
```
Text(
    buildAnnotatedString {
        withStyle(style = SpanStyle(color = Color.Blue)) {
            append("H")
        }
        append("ello ")

        withStyle(style = SpanStyle(fontWeight = FontWeight.Bold, color =
Color.Red)) {
            append("W")
        }
        append("orld")
    }
)
```

Hello World

Text

- maxLines, overflow
- 줄임 효과는 Ellipsis, Visible, Clip 중 하나 지정
- Ellipsis 이외에는 큰 의미가 없다.

```
Text(
    "hello".repeat(50),
    maxLines = 2,
    overflow = TextOverflow.Ellipsis
)
```



Image

- 컴포즈에서 뷰에서 사용하던 Bitmap, Drawable 을 직접 출력할 수는 없다.
- 이미지가 화면에 출력되는 것은 어떤 API 에 의해 드로잉이 되어야 하는데 뷰에서 사용하던 Bitmap, Drawable 은 android.graphic 의 API 이다.
- 컴포즈에서는 이 API 를 사용하지 않고 androidx.compose.ui.graphic 에 있는 Painter 를 이용하기 때문이다.

- Image 컴포저블에 이미지를 지정해야 하는데 3가지 타입 중 하나로 준비가 되어야 한다.
- ImageBitmap, ImageVector, Painter

- 편의성을 위해 ImageBitmap, ImageVector 를 제공하는 것 뿐이며 내부적으로는 모두 Painter 에 의해 이미지가 처리된다.

Image

Painter

- 그리는 역할, ImageBitmap, ImageVector 를 그리는 역할을 한다.

```
Image(
    painter = painterResource(R.drawable.flower1),
    contentDescription = "test image"
)
```

Image

ImageBitmap

- 래스퍼 이미지
- JPEG, PNG, WEBP
- 대부분의 데이터 사진
- 픽셀 데이터

- asImageBitmap() 으로 Bitmap 을 ImageBitmap 으로 변환해서 이용

```
val bitmap: Bitmap = BitmapFactory.decodeResource(context.resources,
R.drawable.flower1)
Image(
    bitmap = bitmap.asImageBitmap(),
    contentDescription = "test image"
)
```

Image

ImageBitmap

- 리소스 이미지를 ImageBitmap 으로 획득해서 이용

```
Image(
    bitmap = ImageBitmap.imageResource(R.drawable.flower1),
    contentDescription = "test image"
)
```

Image

ImageVector

- 벡터 이미지
- 화면에 표시되는 시각적인 요소의 수학적 표현
- 선, 점, 채우기 등의 그리기 위한 정보의 집합
- XML 이미지

```
Image(
    imageVector = ImageVector.vectorResource(id = R.drawable.vector_image),
    contentDescription = "test image"
)
```

Image

아이콘 이미지

- https://developer.android.com/reference/kotlin/androidx/compose/material/icons/package-summary
- 머티리얼 아이콘을 이용할 수 있다.
- https://fonts.google.com/icons

```
implementation("androidx.compose.material:material-icons-extended:1.7.6")
```

```
Image(
    imageVector = Icons.Filled.Call,
    contentDescription = "test image"
)
```

Image

네트워크 이미지 출력

- Coil 라이브러리 혹은 Glide 라이브러리 이용 권장
- Coil : Kotlin Coroutine 에서 지원하는 이미지 로드 라이브러리
- Glide : Google 에서 만든 이미지 로드 라이브러리

```
implementation(libs.coil)
//아래의 라이브러리 있어야 이미지 다운로드 된다.
implementation(libs.coil.network.okhttp)
```

```
AsyncImage(
    model = "https://~~~~",
    contentDescription = "test image",
    onError = {

}
)
```

Image

ContentScale

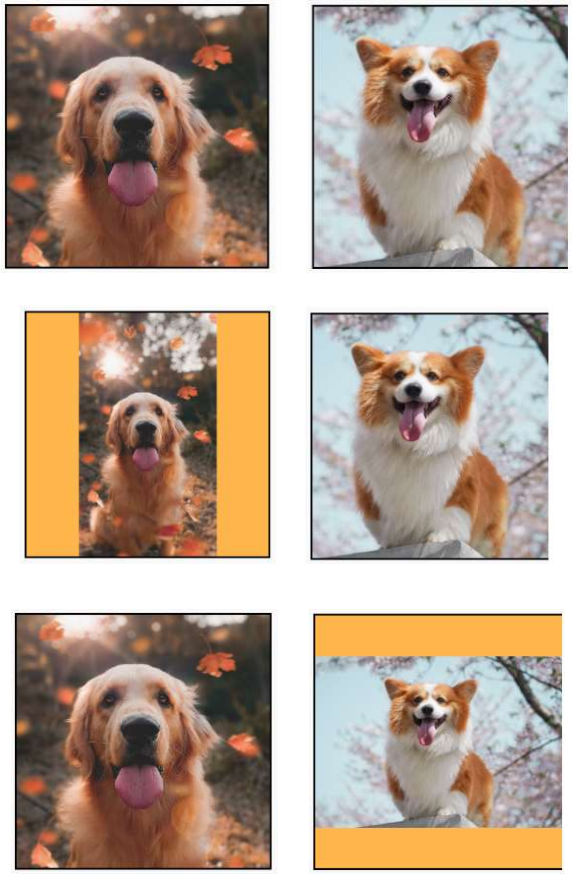
- 이미지 크기와 출력 사이즈가 맞지 않았을 때 설정

```
Image(
    painter = painterResource(id = R.drawable.dog1),
    contentDescription = "test",
    contentScale = ContentScale.Fit,
    modifier = imageModifier
)
```

- ContentScale.Fit
  - 가로세로 비율 유지하면서 이미지 크기를 확대하거나 축소해서 컴포저블 경계에 맞춘다.


Image

- ContentScale.Crop
  - 이미지 가운데를 중심으로 컴포저블에 맞게 잘라서 출력
  - 확대 축소되지는 않는다.
- ContentScale.FillHeight
  - 비율 유지하면서 컴포저블의 높이에 맞추어서 확대/ 축소
- ContentScale.FillWidth
  - 비율 유지하면서 컴포저블의 너비에 맞추어서 확대/축소



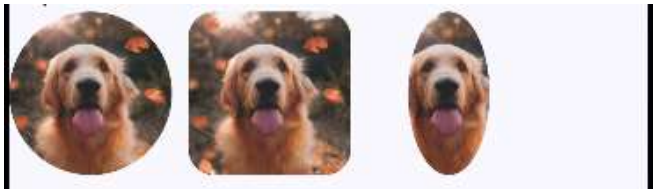
Image

- ContentScale.FillBounds
  - 가소 세로 컴포저블에 맞게 확대/ 축소, 비율이 유지되지 않는다.

- ContentScale.Inside
  - 컴포저블의 크기보다 이미지가 작으면 None
  - 크다면 Fit 처럼 동작
  - 즉 큰 경우에만 비율 유지해 축소해서 출력

Image

- Clip
  - 이미지 속성이라기 보다는 Modifier 속성
  - 주로 이미지 출력에 많이 이용.



```
Image(
    painter = painterResource(id = R.drawable.dog1),
    contentDescription = "test",
    contentScale = ContentScale.Crop,
    modifier = Modifier
        .size(100.dp)
        .clip(CircleShape)
)
```

  - CircleShape, RoundedCornerShape 제공
  - 만약 임의 도형대로 자르려면 Shape 상속받은 커스컴 클래스를 만들어 적용

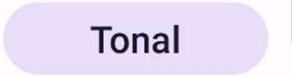
Button

- Button

```
Button(onClick = { onClick() }) {
        Text("Filled")
}
```

- FilledTonalButton

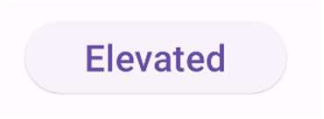
```
FilledTonalButton(onClick = { onClick() }) {
        Text("Tonal")
}
```



Button

- OutlinedButton

- ElevatedButton



- TextButton

Checkbox

- checked : 체크박스 값
- onCheckedChanged() : 이벤트 콜백
- 상태 값이 유지되어야 하고 그 값의 변경에 따른 re-composition 이 이루어져야 한다.

```
var checkState by remember {
    mutableStateOf(true)
}
Checkbox(
    checked = checkState,
    onCheckedChange = {
        checkState = it
    }
)
```

RadioButton

- selectableGroup
- 라디오 버튼을 이용하려면 selectableGroup 에 대해 먼저 이해해야 한다.
- seletableGroup 은 Modifier 속성으로 모든 컴포저블에 사용이 가능하다.
- 여러 컴포저블을 묶고, 그중 하나만 선택되게 하고자 할 때 이용된다.
- 각 항목의 selectable() 에서 해당 항목이 선택된 것인지, 클릭했을 때 이벤트 처리를 명시

```
Row(modifier = Modifier.fillMaxWidth().selectableGroup().padding(10.dp)) {
    arrayDatas.forEach {
        Card(
            ……
            modifier = Modifier
                .weight(1f)
                .selectable(
                    selected = (it == selected),
                    onClick = { onChangeSelected(it)}
                )

        ) {              ……          }
    }
}
```

RadioButton

- RadioButton 은 여러 개중 하나만 선택됨을 목적으로 함으로 대부분 selectableGroup 으로 묶어서 사용이 된다.

```
Column(
    modifier = Modifier
        .selectableGroup()
        .padding(16.dp)
) {
    options.forEach { option ->
        Row(…) {
            RadioButton(
                selected = selectedOption == option,
                onClick = { selectedOption = option },
                modifier = Modifier.padding(end = 16.dp)
            )
            ……          }
    }

    ……
}
```

Switch

- checked: 스위치 상태 값, Boolean
- onCheckedChange : 이벤트 콜백
- enabled : 활성 상태
- colors
- thumbContent

```
Switch(
    checked = switchState,
    onCheckedChange = { switchState = it },

)
```

Slider

- value
- onValueChange
- valueRange
- color
- step : 단계

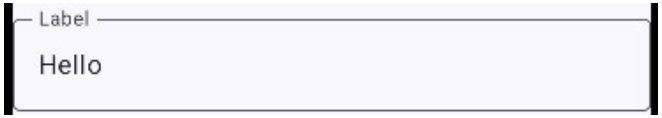
```
Slider(
    value = sliderPosition,
    onValueChange = { sliderPosition = it },
    valueRange = 0f..100f
)
```

TextField / OutlinedTextField

- 상태에 의해 입력되는 값이 관리되어야 입력이 된다.

```
var text by remember { mutableStateOf("Hello") }
TextField(
    value = text,
    onValueChange = { text = it},
    label = { Text("입력") },
    modifier = Modifier.fillMaxWidth()
)
```

- OutlinedTextField



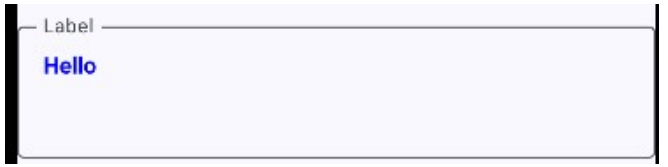
TextField / OutlinedTextField

- singleLine / maxLines
- 기본은 한줄로 출력되다가 키보드에 의해 여러줄로 늘어난다.
  - singleLine : 한줄 고정
  - maxLines : 최대 라인 수
  - minLines : 최소 라인 수
- maxLines 와 minLines 를 동일한 수로 지정하여 고정 사이즈

```
OutlinedTextField(
            value = text,
            onValueChange = { text = it },
            label = { Text("Label") },

            maxLines = 3,
            minLines = 3,
            modifier = Modifier.fillMaxWidth()
        )
```

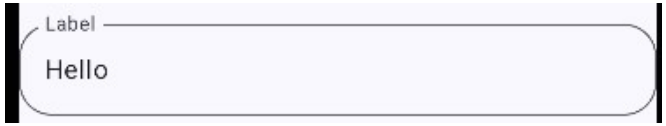
TextField / OutlinedTextField

- textStyle
- Text 와 마찬가지로 입력되는 문자열의 스타일 지정 가능




```
textStyle = TextStyle(color = Color.Blue, fontWeight = FontWeight.Bold),
```

- shape
- OutlinedTextField 인 경우 테두리 모양 지정 가능
  - CircleShape
  - RoundedCornerShape
  - RectangleShape
  - CutCornerShape




```
shape = RoundedCornerShape(20.dp),
```

TextField / OutlinedTextField

- leadingIcon / trailingIcon

```
leadingIcon = {
    Icons.Outlined.Lock
},
trailingIcon = trailingIconView
```

- keyboardType
  - KeyboardType.Email
  - KeyboardType.Number
  - KeyboardType.NumberPassword
  - KeyboardType.Password
  - KeyboardType.Phone
  - KeyboardType.Text
  - KeyboardType.Uri


```
keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Number)
```

TextField / OutlinedTextField

- imeAction / keyboardActions
- imeAction 은 우측 하단 키를 어떤 키로 사용할 것인가에 대한 설정
- keyboardActions 는 우측 하단 키를 클릭 했을 때의 이벤트 처리

```
keyboardOptions = KeyboardOptions(
    keyboardType = KeyboardType.Number,
    imeAction = ImeAction.Search
),
keyboardActions = KeyboardActions(
    onSearch = {
        Toast.makeText(context, "search click", Toast.LENGTH_SHORT).show()
    },
    onDone = {
        Toast.makeText(context, "done click", Toast.LENGTH_SHORT).show()
    })
```

List

리스트

- 목록 화면을 구성하기 위해 LazyColumn, LazyRow, LazyVerticalGrid와 같은 함수를 제공합니다.
- 'Lazy'가 붙는 이유는 '늦은 로딩'을 지 원하는 컴포저블이기 때문입니다.
- LazyColumn을 이용하면 다음과 같이 항목을 세로 방향으로 나열
- items() 를 이용해 각 항목이 어떻게 구성되는지를 명시

• LazyColumn으로 세로 방향으로 항목 나열

```
LazyColumn {
    val datas = listOf<String>("one","two","three","four")
    items(datas) { item ->
        Text("$item ")
    }
}
```

▶ 실행 결과

List

- items() 대신 itemsIndexed()를 사용할 수 있는데 이렇게 되면 각 항목별로 index까지 출력

• itemsIndex 함수로 항목별 인덱스 출력

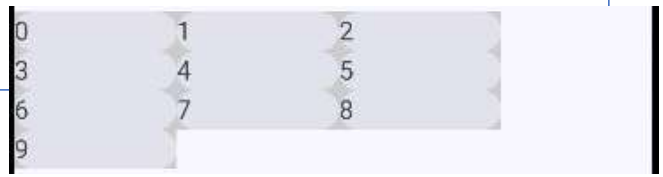
```
LazyColumn {
    val datas = listOf<String>("one","two","three","four")
    itemsIndexed(datas){index, item ->
        Text("$index $item")
    }
}
```

▶ 실행 결과

LazyVerticalGrid, LazyHorizontalGrid

- LazyVerticalGrid 에서는 columns 로, LazyHorizontalGrid 에서는 rows 로 한 줄에 몇 개의 item 을 배치할 것인지를 결정
- columns 와 rows 의 값은 GridCells 이어야 하는데 이 값을 주는 방법이 여러가지가 있다.
  - GridCells.FixedSize(100.dp) : 각 셀이 차지할 사이즈 지정
  - GridCells.Fixed(4) : 한 줄에 셀 개수 지정
  - GridCells.Adaptive() : 동적 셀 개수 지정

```
LazyVerticalGrid(columns = GridCells.FixedSize(100.dp)) {
    items(10){ item ->
        Card(modifier = Modifier.background(Color.LightGray)) {
            Text("$item")
        }
    }
}
```



LazyVerticalGrid, LazyHorizontalGrid

```
LazyVerticalGrid(columns = GridCells.Fixed(4) ) {
    items(10){ item ->
        Card(modifier = Modifier.background(Color.LightGray)) {
            Text("$item")
        }
    }
}
```


```
LazyVerticalGrid(columns = GridCells.Adaptive(minSize = 100.dp) ) {
    items(10){ item ->
        Card(modifier = Modifier.background(Color.LightGray)) {
            Text("$item")
        }
    }
}
```

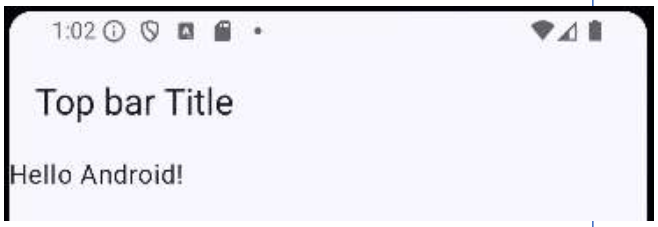
Scaffold

- 앱의 구조를 빠르게 조합하기 위한 컴포저블
- Material2 와 Material3 에 차이가 있다. Material3를 사용할 것을 권장
  - topBar
  - bottomBar
  - snackbarHost
  - floatingActionButton
  - floatingActionButtonPosition
  - contentColor
  - content
  - containerColor
- Material3에서는 drawerContent 는 제공하지 않는다. ModalNavigationDrawer 활용해야 한다.

TopAppBar

- 화면 상단 구성
  - title
  - colors
  - navigationIcon
  - actions

```
Scaffold(
    modifier = Modifier.fillMaxSize(),
    topBar = {
        TopAppBar(
            title = {
                Text("Top bar Title")
            }
        )
    }
)
```



TopAppBar

- **topBar 에 색상 지정하기**
- colors 속성을 이용해 TopBar 의 색상 및 TopBar 컨텐츠의 색상 지정

```
TopAppBar(
    title = {
        Text("Top bar Title")
    },
    colors = topAppBarColors(
        containerColor = Color.Red,
        titleContentColor = Color.White
    )
)
```

TopAppBar

Theme 칼라로 색상 지정하기

```
colors = topAppBarColors(
    containerColor = MaterialTheme.colorScheme.primary,
    titleContentColor = MaterialTheme.colorScheme.primaryContainer
)
```

Custom Theme Color 지정

- Theme.kt 파일

```
private val DarkColorScheme = darkColorScheme(
    primary = CustomPrimaryColor,
    primaryContainer = CustomPrimaryContainer,
    secondary = PurpleGrey80,
    tertiary = Pink80
)

private val LightColorScheme = lightColorScheme(
    primary = CustomPrimaryDarkColor,
    primaryContainer = CustomPrimaryContainerDark,
    secondary = PurpleGrey40,
    tertiary = Pink40
)
```

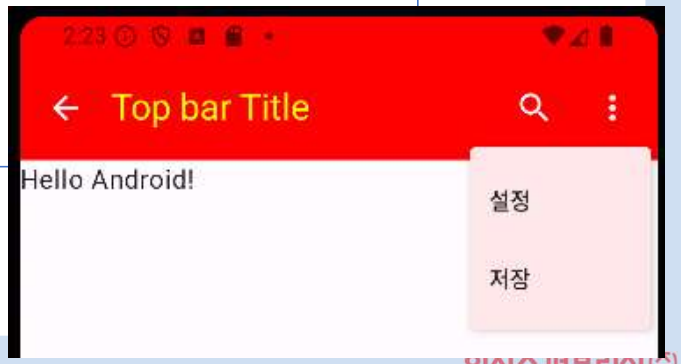
TopAppBar

navigationIcon

```
navigationIcon = {
    IconButton(onClick = {}) {
        Icon(Icons.AutoMirrored.Filled.ArrowBack, contentDescription = "")
    }
}
```

actions

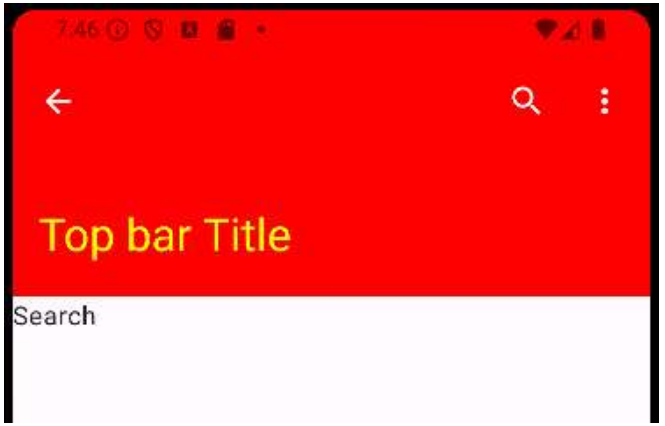
```
actions = {
    IconButton(onClick = {}) {
        Icon(Icons.Filled.Search, contentDescription = "")
    }
    .....
}
```



TopAppBar

AppBar 종류

- TopAppBar
- CenterAlignedTopAppBar : title 이 가운데 정렬
- MediumTopAppBar : title 이 아이콘 아랫줄에 정렬
- LargeTopAppBar : title 과 아이콘 사이 공간이 있는 큰 사이즈



LargeTopAppBar

TopAppBar

앱바 접히기

- 아랫 부분이 스크롤 될 때 앱바가 같이 스크롤 되려면 behavior 가 선언되어야 한다.

```
val scrollBehavior =
TopAppBarDefaults.exitUntilCollapsedScrollBehavior(rememberTopAppBarState())
```

- 그런후에 Scaffold Modifier 에 선언된 behavior 를 등록해 이 behavior 에 화면의 스크롤 정보가전달되게 해주어야 한다.

```
Scaffold(
    modifier =
Modifier.fillMaxSize().nestedScroll(scrollBehavior.nestedScrollConnection),
```

- 또한 AppBar 에 behavior 를 등록해 이 behavior 의 정보에 의해 앱바가 접히게 설정해 주어야 한다.

```
LargeTopAppBar(
    ……
    scrollBehavior = scrollBehavior
)
```

TopAppBar

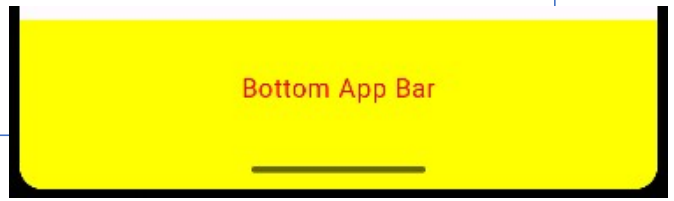
앱바 접히기

- Behavior 를 만들 때 여러 함수가 제공되며 어느 함수를 이용하는지에 따라 스크롤이 다르게 동작한다.
  - exitUntilCollapsedScrollBehavior() : 거꾸로 스크롤 할 때 맨 마지막에 앱바가 펼처진다.
  - enterAlwaysScrollBehavior() : 거꾸로 스크롤 할 때 가장 먼저 앱바가 펼처진다.
  - pinnedScrollBehavior : 스크롤에 반응하지 않는다.

BottomAppBar

- bottomBar 는 Scaffold 영역의 하단에 나오며 BottomAppBar 이외에 다른 컴포저블을 사용해도 된다.
- 대표적으로 많이 사용되는 것이 BottomAppBar 와 NavigationBar 이다.

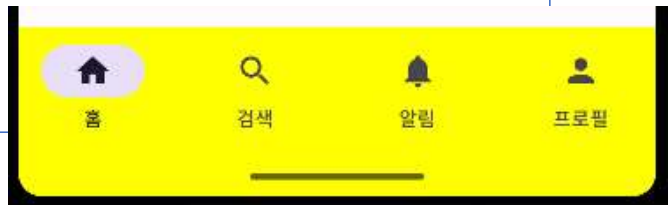
```
bottomBar = {
    BottomAppBar(
        containerColor = MaterialTheme.colorScheme.primaryContainer,
        contentColor = MaterialTheme.colorScheme.primary
    ) {
        Text(
            modifier = Modifier.fillMaxWidth(),
            textAlign = TextAlign.Center,
            text = "Bottom App Bar"
        )
    }
}
```



BottomAppBar

- NavigationBar 를 이용해서 탭 화면 효과를 낸다.

```
NavigationBar(
    containerColor = MaterialTheme.colorScheme.primaryContainer,
    contentColor = MaterialTheme.colorScheme.onPrimaryContainer,
) {
    items.forEachIndexed { index, item ->
        NavigationBarItem(
            icon = { Icon(icons[index], contentDescription = item) },
            label = { Text(item) },
            selected = selectedItem == index,
            onClick = { selectedItem = index }
        )
    }
}
```



FloatingActionButton

- NavigationBar 를 이용해서 탭 화면 효과를 낸다.
  - FloatingActionButton
  - SmallFloatingActionButton
  - LargeFloatingActionButton
  - ExtendedFloatingActionButton

- FloatingActionButton 은 Scaffold 의 구성요소로 추가해도 되고, 독립적으로 사용해도 된다.
- FAB 를 구성하는 것은 Text 등 다양한 것으로 구성이 가능하지만 일반적으로 Icon

```
SmallFloatingActionButton(
            onClick = {},
            containerColor = MaterialTheme.colorScheme.secondaryContainer,
            contentColor = MaterialTheme.colorScheme.secondary
        ) {
            Icon(Icons.Filled.Add, "ADD")
        }
```

FloatingActionButton

- Scaffold 구성요소로 추가

```
floatingActionButton = {
    FloatingActionButton(onClick = {  }) {
        Icon(Icons.Default.Add, contentDescription = "Add")
    }
}
```

ModalNavigationDrawer

- Drawer 를 제공하는 컴포저블

```
@Composable
fun HomeDrawer(
    closeDrawer: () -> Unit,
    modifier: Modifier = Modifier
) {
    ModalDrawerSheet(modifier = modifier) {
        DrawerHeader()
        DrawerButton(label = "menu1", closeDrawer = closeDrawer)
        ……….
    }
}
```

```
ModalNavigationDrawer(
    drawerState = drawerState,
    drawerContent = {
        HomeDrawer(
            closeDrawer = { coroutineScope.launch { drawerState.close() }}
        )
    }
) {
    Scaffold(……)
}
```

AnimatedVisibility

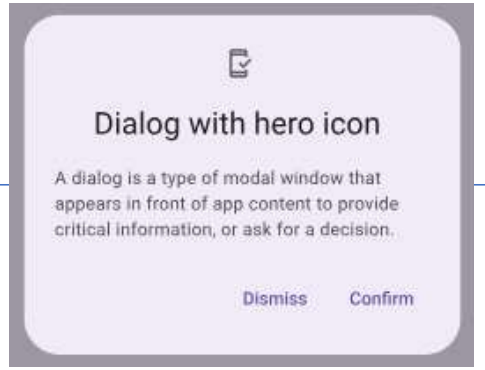
- 애니메이션 효과로 컴포저블 visibility 를 제어하는 컴포저블

```
AnimatedVisibility(
        visible = !isSearchActive,
        enter = expandHorizontally() + fadeIn(),
        exit = shrinkHorizontally() + fadeOut()
    ) {
        HomeAppBar(
            openDrawer = { coroutineScope.launch { drawerState.open() }},
            showSearchbar = { isSearchActive = true}
        )
    }
```

AlertDialog

- onDismissRequest : 사용자가 다이얼로그 외부를 탭 했을 때 호출되는 함수

```
AlertDialog(
    icon = {....},
    title = {....},
    text = {....},
    onDismissRequest = {....},
    confirmButton = {.....},
    dismissButton = {.....}
)
```



스크롤

- 스크롤을 구현하는 가장 효율적인 방법은 LazyRow, LazyColumn 을 이용하는 것이다.

verticalScroll, horizontalScroll modifier 이용

- 주로 Row, Column 의 modifier에 설정이 되지만 다른 컴포저블도 가능하다.

```
Row(modifier = Modifier.horizontalScroll(rememberScrollState())) {
    ……

}
Text(text = "hello world", modifier = Modifier.size(50.dp,
                20.dp).horizontalScroll(
                    rememberScrollState()
))
```

스크롤

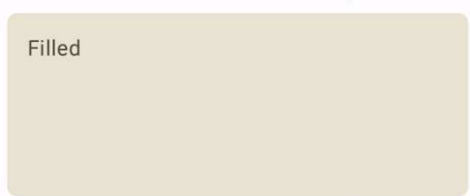
verticalScroll, horizontalScroll modifier 이용

- 스크롤 위치를 조정하거나 획득하기 위해서 rememberScrollState() 를 이용할 수도 있다.

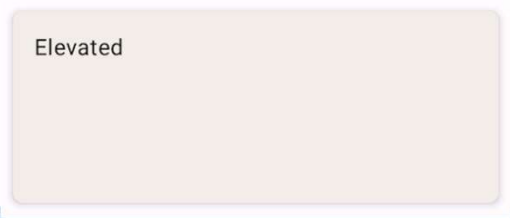
```
val state = rememberScrollState()
LaunchedEffect(Unit) { state.animateScrollTo(200) }

Column {
    Column(
        modifier = Modifier
            .verticalScroll(state)
    ) {……..}

    }
     Button(onClick = {
        scope.launch {
            state.scrollTo(400)
        }
    }) {……..}
}
```

Card

```
Card(
    colors = CardDefaults.cardColors(
        containerColor = MaterialTheme.colorScheme.surfaceVariant,
    ),
    modifier = Modifier
        .size(width = 240.dp, height = 100.dp)
) {
    Text(
        text = "Filled",
        modifier = Modifier
            .padding(16.dp),
        textAlign = TextAlign.Center,
    )
}
```



- ElevatedCard



OutlinedCard

감사합니다

단단히 마음먹고 떠난 사람은
산꼭대기에 도착할 수 있다.
산은 올라가는 사람에게만 정복된다.

윌리엄 셰익스피어
William Shakespeare