기본 UI 및 상태 2

예제. 버튼과 이미지 활용

- 학습할 내용
  - 버튼
  - 이미지
  - SlotAPI
  - Badge
  - Column에 스크롤 기능 추가

Button

- 사용자가 정의된 작업을 트리거 할 수 있도록 하는 기본 구성요소

```
@Composable
fun Button(
    onClick: () -> Unit,
    modifier: Modifier = Modifier,
    enabled: Boolean = true,
    shape: Shape = ButtonDefaults.shape,
    colors: ButtonColors = ButtonDefaults.buttonColors(),
    elevation: ButtonElevation? = ButtonDefaults.buttonElevation(),
    border: BorderStroke? = null,
    contentPadding: PaddingValues = ButtonDefaults.ContentPadding,
    interactionSource: MutableInteractionSource = remember { MutableInteractionSource() },
    content: @Composable RowScope.() -> Unit
): Unit
```

- 버튼 종류

Button

```
@Composable
fun FilledButtonExample(onClick: () -> Unit) {
    Button(onClick = { onClick() }) {
        Text("Filled")
    }
}
```

Filled

```
@Composable
fun FilledTonalButtonExample(onClick: () -> Unit) {
    FilledTonalButton(onClick = { onClick() }) {
        Text("Tonal")
    }
}
```

Tonal

```
@Composable
fun OutlinedButtonExample(onClick: () -> Unit) {
    OutlinedButton(onClick = { onClick() }) {
        Text("Outlined")
    }
}
```

Outlined

```
@Composable
fun ElevatedButtonExample(onClick: () -> Unit) {
    ElevatedButton(onClick = { onClick() }) {
        Text("Elevated")
    }
}
```

Elevated

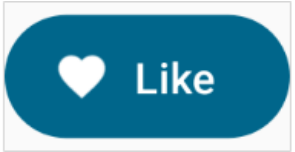
```
@Composable
fun TextButtonExample(onClick: () -> Unit) {
    TextButton(
        onClick = { onClick() }
    ) {
        Text("Text Button")
    }
}
```

Text Button

Button

- 아이콘 버튼

```
@Composable
fun ButtonWithIconSample() {
    Button(
        onClick = { /* Do something! */ },
        contentPadding = ButtonDefaults.ButtonWithIconContentPadding
    ) { this: RowScope
        Icon(
            Icons.Filled.Favorite,
            contentDescription = "Localized description",
            modifier = Modifier.size(ButtonDefaults.IconSize)
        )
        Spacer(Modifier.size(ButtonDefaults.IconSpacing))
        Text( text: "Like")
    }
}
```



Image

- 화면에 그래픽을 표시하는 컴포저블
  - 래스터(비트맵) 이미지(PNG, JPEG, WEBP)

```
Image(
    painter = painterResource(id = R.drawable.dog),
    contentDescription = stringResource(id = R.string.dog_content_description)
)
```

```
val imageBitmap = ImageBitmap.imageResource(R.drawable.dog)
```

  - 벡터 이미지

```
Image(
    painter = painterResource(id = R.drawable.baseline_shopping_cart_24),
    contentDescription = stringResource(id = R.string.shopping_cart_content_desc)
)
```

```
val imageVector = ImageVector.vectorResource(id = R.drawable.baseline_shopping_cart_24)
```

  - 네트워상의 이미지

```
AsyncImage(
    model = "https://example.com/image.jpg",
    contentDescription = "Translated description of what the image contains"
)
```

```
implementation("io.coil-kt:coil-compose:2.5.0")
```

Kotlin 코루틴에서 지원하는 이미지 로드 라이브러리

```
<uses-permission android:name="android.permission.INTERNET"/>
```

권한필요함

이미지 맞춤 설정

- ContentScale 옵션을 통한 크기 조정

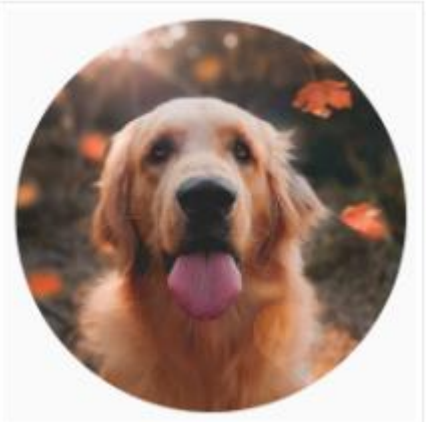
`ContentScale.Fit`: 가로세로 비율(기본값)을 유지하면서 이미지의 크기를 균일하게 조정합니다. 콘텐츠가 크기보다 작으면 이미지는 경계에 맞게 확대 조정됩니다.

`ContentScale.Crop`: 사용 가능한 공간에 맞게 이미지를 가운데를 중심으로 자릅니다.

이미지 맞춤 설정

`ContentScale.FillBounds`: 대상 경계를 채우도록 콘텐츠의 크기를 **균일하지 않게** 세로 및 가로로 조정합니다. (참고: 이미지의 정확한 비율과 일치하지 않는 컨테이너에 배치하면 이미지가 왜곡됩니다.)

```
Image(
    painter = painterResource(id = R.drawable.dog),
    contentDescription = stringResource(id = R.string.dog_content_description),
    contentScale = ContentScale.Crop,
    modifier = Modifier
        .size(200.dp)
        .clip(CircleShape)
)
```



Resource 가져오기

- 문자열

```
// In the res/values/strings.xml file
// <string name="compose">Jetpack Compose</string>

// In your Compose code
Text(
    text = stringResource(R.string.compose)
)
```

```
// In the res/values/strings.xml file
// <string name="congratulate">Happy %1$s %2$d</string>

// In your Compose code
Text(
    text = stringResource(R.string.congratulate, "New Year", 2021)
)
```

- 크기

```
// In the res/values/dimens.xml file
// <dimen name="padding_small">8dp</dimen>

// In your Compose code
val smallPadding = dimensionResource(R.dimen.padding_small)
Text(
    text = "...",
    modifier = Modifier.padding(smallPadding)
)
```

- 색상

```
// In the res/colors.xml file
// <color name="colorGrey">#757575</color>

// In your Compose code
Divider(color = colorResource(R.color.colorGrey))
```

Resource 가져오기

- AnimatedImageVector

```
// Files in res/drawable folders. For example:
// - res/drawable/animated_vector.xml

// In your Compose code
val image = AnimatedImageVector.animatedVectorResource(R.drawable.animated_vector)
val atEnd by remember { mutableStateOf(false) }
Icon(
    painter = rememberAnimatedVectorPainter(image, atEnd),
    contentDescription = null // decorative element
)
```

- 아이콘

```
// Files in res/drawable folders. For example:
// - res/drawable-nodpi/ic_logo.xml
// - res/drawable-xxhdpi/ic_logo.png

// In your Compose code
Icon(
    painter = painterResource(id = R.drawable.ic_logo),
    contentDescription = null // decorative element
)
```

```
import androidx.compose.material.Icon

Icon(Icons.Rounded.Menu, contentDescription = "Localized description")
```

SlotAPI

- Slot API?
  - 컴포저블에 대한 API를 추가하여 **호출자가 슬롯 안에 표시할 컴포저블을 지정할 수 있도록 허가하는 것**
  - 동적 layout 구성할 수 있음

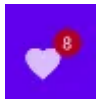
```
@Composable
fun SlotDemo(middleContent:@Composable ()-> Unit) {
    Column{ this: ColumnScope
        Text( text: "Top Text")
        middleContent()
        Text( text: "Button Text")
    }
}

SlotDemo(middleContent = { ButtonDemo()})
SlotDemo(middleContent = { TextCell( text: "2") })
```

BadgeBox 사용하기

- Badge는 아이콘이나 짧은 텍스트로 새로운 알림 표시

```
@Composable
fun BadgedBox(
    badge: @Composable BoxScope.() -> Unit,
    modifier: Modifier = Modifier,
    content: @Composable BoxScope.() -> Unit,
)
```

```
BadgedBox(badge = { Badge { Text("8") } }) {
    Icon(
        Icons.Filled.Favorite,
        contentDescription = "Favorite"
    )
}
```



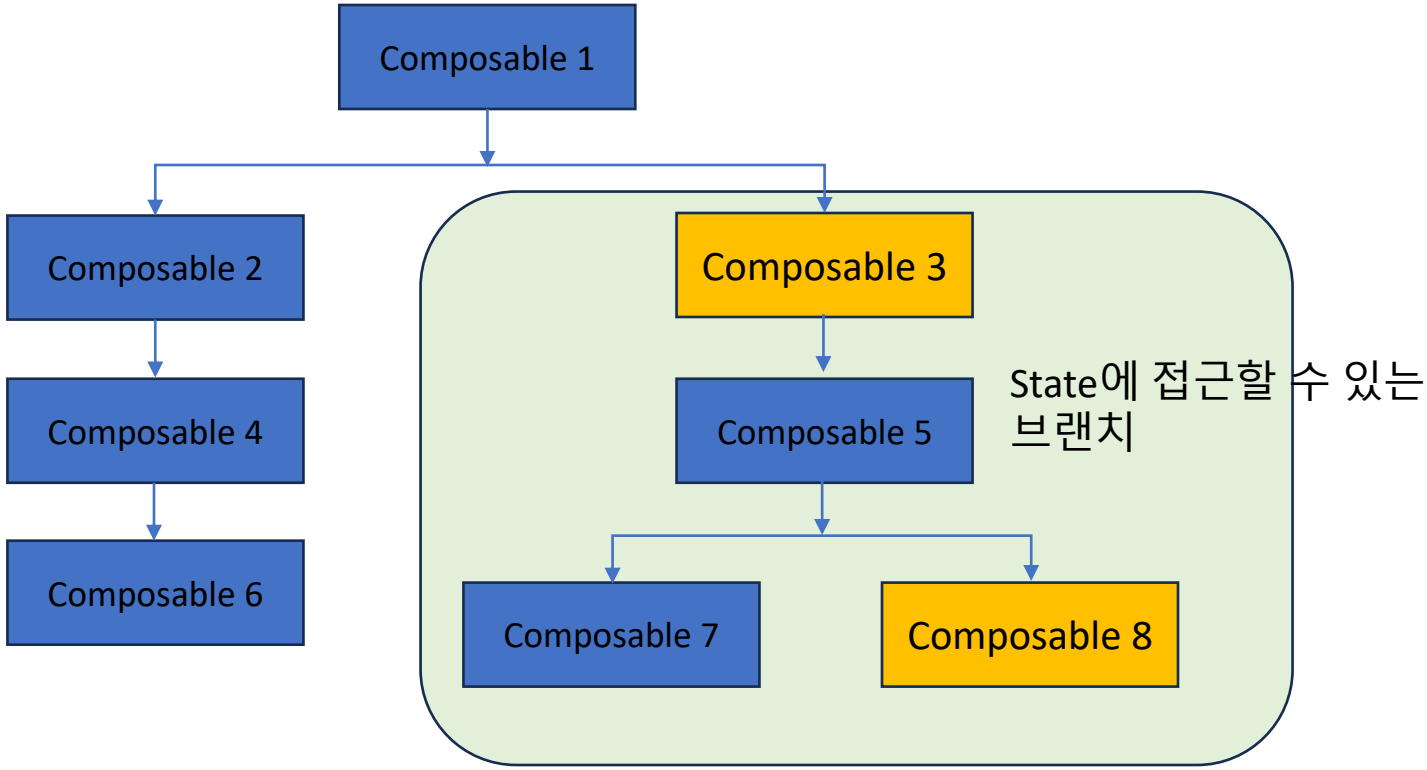
Column에 Scroll 기능 추가

- 화면을 벗어나는 경우 Scroll 기능 추가

```
val scrollState = rememberScrollState()
Column(modifier = Modifier.verticalScroll(scrollState)) {


}
```

CompositionLocal

- CompositionLocal을 이용한 데이터 전달
  - 컴포저블 계층 트리 상위에서 선언된 데이터를 계층 트리 하위의 함수에 전달하는 기능
  - 매개변수를 선언하지 않아도 하위 Composable에서 사용할 수 있음



CompositionLocal

- 단점
  - 컴포저블의 동작을 추론하기 어렵게 함
  - 문제가 발생할 때 앱 디버깅을 어렵게 함
    - **CompositionLocal을 과도하게 사용하지 않는 것이 좋음**
- CompositionLocal 사용여부
  - 적절한 기본값이 있어야 함
  - 하위 계층 구조 범위로 간주되지 않을 때는 사용하지 않음
- CompositionLocal 인스턴스 생성
  - **compositionLocalOf**
    - 제공된 값을 변경하면 current 값을 읽는 콘테츠만 리컴포지션
    - 동적으로 변경되는 상태값 저장
  - **staticCompositionLocalOf**
    - 값을 변경하면 CompositionLocal이 제공된 content 전체가 리컴포지션
    - 자주 변경되지 않는 상태값 저장

CompositionLocal

- 객체 생성

```
data class User(val name: String, val age: Int)
val LocalActiveUser = compositionLocalOf<User> { error("No user found!") }
```

기본값을 정의하는 람다

- CompositionLocalProvider를 통한 전달

```
@Composable
private fun MyUserScreen() {
    val user = User("Jens", 31)
    CompositionLocalProvider(LocalActiveUser provides user) {
        UserInfo()
    }
}
```

- current 프로퍼티를 통한 상태 접근

```
@Composable
fun UserInfo() {
    Column {
        Text("Name: " + LocalActiveUser.current.name)
        Text("Age: " + LocalActiveUser.current.age)
    }
}
```

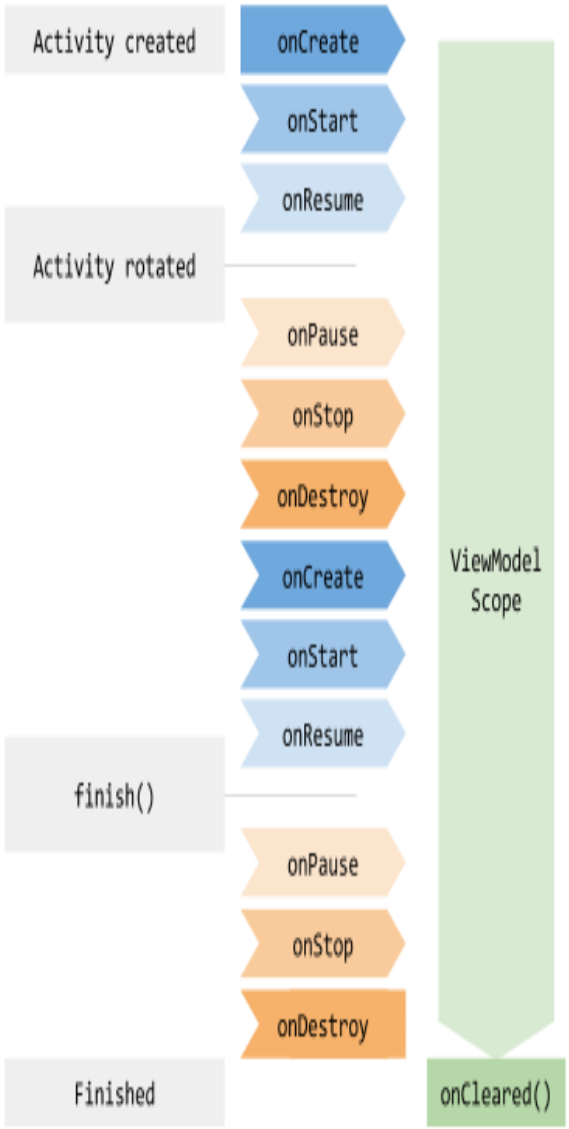
예제. CompositionLocalDemo

LightPreview

DarkPreview

ViewModel

- ViewModel 클래스
  - UI 관련 데이터를 관리하고, UI 구성요소 간의 상호작용을 처리하는데 사용되는 클래스
  - 상태를 캐시하여, 구성이 변경되어도 데이터 유지
    - 화면 회전 및 구성 변경시 데이터 다시 가져올 필요 없음
- ViewModel을 사용한 데이터 공유
  - ViewModel은 연결된 ViewModelStoreOwner(예, Activity, Fragment)의 생명주기에 의해 관리
    - 예) Activity의 생명주기와 일치
    - 한번 생성된 ViewModel은 다른 컴포저블에서 동일한 ViewModel 객체 생성시 동일한 객체를 반환함
  - UI 구성요소 간의 의존성을 줄이고 데이터를 보다 효율적으로 관리할 수 있음



ViewModel

- ViewModel 클래스 정의

```
class MyViewModel : ViewModel(){
    private var _data by mutableStateOf(0)
    var data: Int
        get() = _data
        set(value) {
            _data = value
        }
}
```

```
class MyViewModel : ViewModel(){
    private var _data = mutableStateOf(0)
    val data = _data
    fun setData(value:Int){
        _data.value = value
    }
}
```

- Activity에서 ViewModel 생성

```
val viewModel by viewModels<MyViewModel>()
```

- 컴포저블에서 ViewModel 생성
  - 라이브러리 추가 ➔ 검색 : androidx.lifecycle:lifecycle-viewmodel-compose

implementation(*libs*.*androidx*.*lifecycle*.*viewmodel*.*compose*)

```
val myViewModel = viewModel<MyViewModel>()
```

```
val myViewModel : MyViewModel  = viewModel()
```

```
Text(text=myViewModel.data.toString())

myViewModel.data=10
```

```
Text(text=myViewModel.data.value.toString())

myViewModel.setData(20)
```

실습. Calculate Tip의 ViewModel 적용

- Calculate Tip 앱에 ViewModel을 적용하여 동일한 기능이 수행되도록 수정해 주세요.

수고하셨습니다.