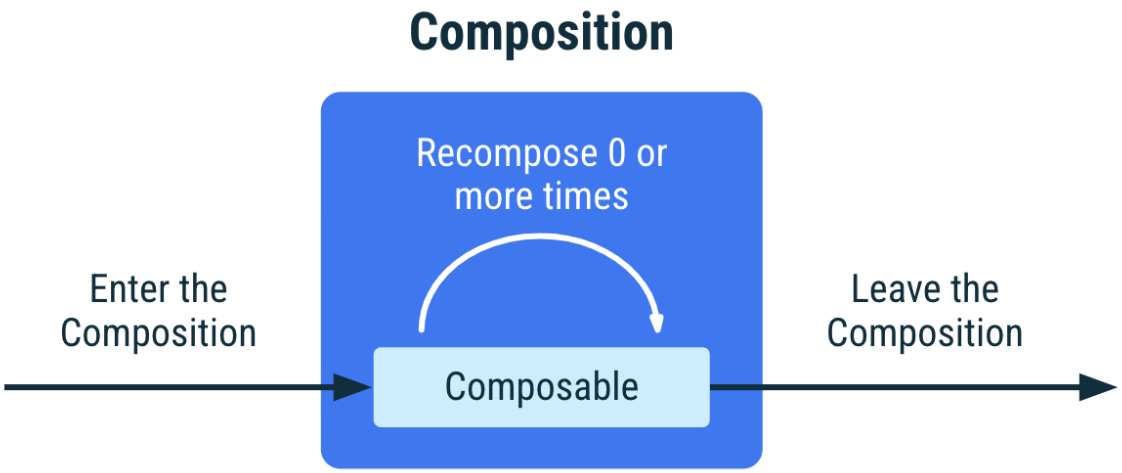
2. 활용

목차

Recomposition



컴포지션 - 앱의 UI 를 Composable 을 호출해 생성. UI 를 기술하는 Composable 의 트리 구조.

초기 컴포지션 - UI 를 그리기 위해 호출한 Composable 을 추적

리컴포지션 - **상태**가 바뀌었을 때 예약되며 (0회 이상) 변경점을 반영하려 컴포지션을 업데이트 함. (수정하는 유일한 방법)

State<T> 가 변경되면 리컴포지션이 트리거 됨.

State<T> 를 읽는 컴포저블과, 여기에서 호출되는 컴포저블이 대상이 됨.

모든 입력이 안정적이고 변경되지 않았으면 건너뛸 수 있음.

안정적인 타입은 아래를 지켜야 함.

- 두 인스턴스 equals 결과가 영원히 같은 경우.
- 공개 프로퍼티가 변경되면 컴포지션에 알려야 함.
- 모든 공개 프로퍼티는 안정적이어야 함.

안정적인 공통 타입들

- 모든 프리미티브 타입 : Boolean, Int, Long, Float 등
- 문자열
- 모든 함수 타입 (람다)

ConstraintLayout

```
@Composable
fun ConstraintLayoutExample() {
    ConstraintLayout(modifier = Modifier.fillMaxSize()) {
        val (redBox, magentaBox, greenBox, yellowBox) = createRefs()

        Box(
            modifier = Modifier
                .size(40.dp)
                .background(Color.Red)
                .constrainAs(redBox) {
                    bottom.linkTo(parent.bottom, margin = 8.dp)
                    end.linkTo(parent.end, margin = 4.dp)
                }
        )
        Box(
            modifier = Modifier
                .size(40.dp)
                .background(Color.Magenta)
                .constrainAs(magentaBox) {
                    start.linkTo(parent.start)
                    end.linkTo(parent.end)
                }
        )
        Box(
            modifier = Modifier
                .size(40.dp)
                .background(Color.Green)
                .constrainAs(greenBox) {
                    centerTo(parent)
                }
        )
        Box(
            modifier = Modifier
                .size(40.dp)
                .background(Color.Yellow)
                .constrainAs(yellowBox) {
                    start.linkTo(magentaBox.end)
                    top.linkTo(magentaBox.bottom)
                }
        )
    }
}
```

ConstraintLayout (ConstraintSet)

```
@Composable
fun ConstraintSetExample() {
    val constraintSet = ConstraintSet {
        val redBox = createRefFor("redBox")
        val magentaBox = createRefFor("magentaBox")
        val greenBox = createRefFor("greenBox")
        val yellowBox = createRefFor("yellowBox")

        constrain(redBox) {
            bottom.linkTo(parent.bottom, 10.dp)
            end.linkTo(parent.end, 30.dp)
        }
        constrain(magentaBox) {
            start.linkTo(parent.start, 10.dp)
            end.linkTo(parent.end, 30.dp)
        }
        constrain(greenBox) {
            centerTo(parent)
        }
        constrain(yellowBox) {
            start.linkTo(greenBox.end)
            top.linkTo(greenBox.bottom)
        }
    }

    ConstraintLayout(
        constraintSet = constraintSet,
        modifier = Modifier.fillMaxSize()
    ) {
        Box(
            modifier = Modifier
                .size(40.dp)
                .background(Color.Red)
                .layoutId("redBox")
        )
        Box(
            modifier = Modifier
                .size(40.dp)
                .background(Color.Magenta)
                .layoutId("magentaBox")
        )
        Box(
            modifier = Modifier
                .size(40.dp)
                .background(Color.Green)
                .layoutId("greenBox")
        )
        Box(
            modifier = Modifier
                .size(40.dp)
                .background(Color.Yellow)
                .layoutId("yellowBox")
        )
    }
}
```

ConstraintLayout (Chain, Barrier)

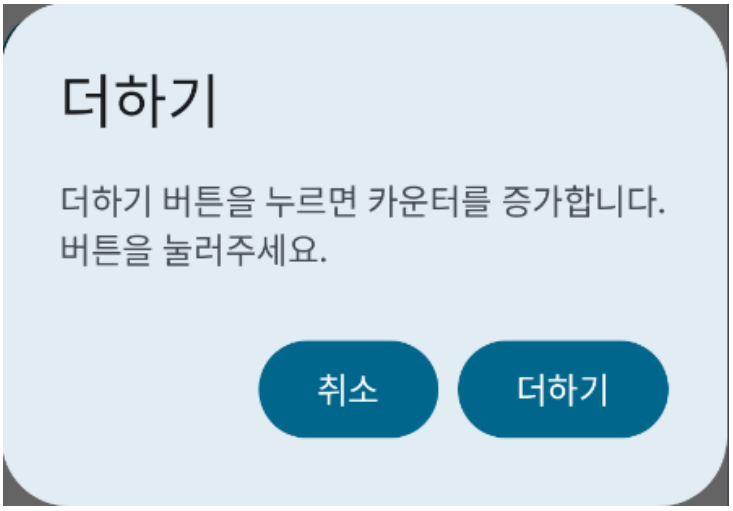
```
@Composable
fun ConstraintChainBarrierExample() {
    ConstraintLayout(Modifier.fillMaxSize()) {
        val (redBox, yellowBox, magentaBox, text) = createRefs()

        Box(
            modifier = Modifier
                .size(40.dp)
                .background(Color.Red)
                .constrainAs(redBox) {
                    top.linkTo(parent.top, margin = 18.dp)
                }
        )
        Box(
            modifier = Modifier
                .size(40.dp)
                .background(Color.Yellow)
                .constrainAs(yellowBox) {
                    top.linkTo(parent.top, margin = 32.dp)
                }
        )
        Box(
            modifier = Modifier
                .size(40.dp)
                .background(Color.Magenta)
                .constrainAs(magentaBox) {
                    top.linkTo(parent.top, margin = 20.dp)
                }
        )

//        createVerticalChain(redBox, yellowBox, magentaBox)
        createHorizontalChain(redBox, yellowBox, magentaBox, chainStyle = ChainStyle.SpreadInside)
        val barrier = createBottomBarrier(redBox, yellowBox, magentaBox)

        Text(
            text = "Hello World! Hello World! Hello World! Hello World! Hello World! ",
            modifier = Modifier.constrainAs(text) {
                top.linkTo(barrier)
            }
        )
    }
}
```

Canvas

```
@Composable
fun CanvasExample() {
    Canvas(modifier = Modifier.size(20.dp)) {
        drawLine(Color.Red, Offset(30f, 10f), Offset(50f, 40f))
        drawCircle(Color.Yellow, 10f, Offset(15f, 40f))
        drawRect(Color.Magenta, Offset(30f, 30f), Size(10f, 10f))

        // Search Icon
        drawLine(Color.Green, Offset(2.01f, 21.0f), Offset(23.0f, 12.0f))
        drawLine(Color.Green, Offset(23.0f, 12.0f), Offset(2.01f, 3.0f))
        drawLine(Color.Green, Offset(2.01f, 3.0f), Offset(2.0f, 10.0f))
        drawLine(Color.Green, Offset(2.0f, 10.0f), Offset(17.0f, 12.0f))
        drawLine(Color.Green, Offset(17.0f, 12.0f), Offset(2.0f, 14.0f))
        drawLine(Color.Green, Offset(2.0f, 14.0f), Offset(2.01f, 21.0f))
    }
}
```

Dialog

```
@Composable
fun DialogExample() {
    var openDialog by remember { mutableStateOf(false) }
    var counter by remember { mutableStateOf(0) }

    Column {
        Button(onClick = { openDialog = true }) {
            Text(" ")
        }
        Text(": $counter")
    }

    if (openDialog) {
        AlertDialog(onDismissRequest = {
            openDialog = false
        }, confirmButton = {
            Button(onClick = {
                counter++
                openDialog = false
            }) {
                Text(text = "")
            }
        }, dismissButton = {
            Button(onClick = {
                openDialog = false
            }) {
                Text(text = "")
            }
        }, title = {
            Text(text = "")
        }, text = {
            Text(text = "    .\n .")
        })
    }
}
```

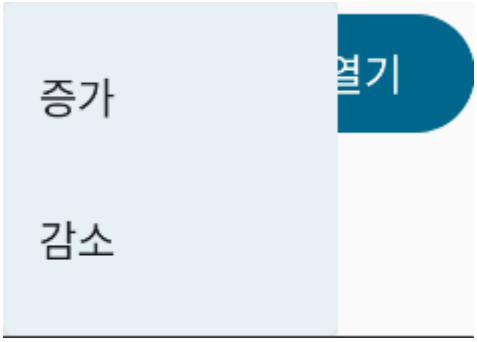


Custom Dialog

```
@Composable
fun CustomDialogExample() {
    var openDialog by remember { mutableStateOf(false) }
    var counter by remember { mutableStateOf(0) }

    Column {
        Button(onClick = {
            openDialog = true
        }) {
            Text(" ")
        }
        Text(": $counter")
    }

    if (openDialog) {
        Dialog(onDismissRequest = {
            openDialog = false
        }) {
            Surface {
                Column(modifier = Modifier.padding(8.dp)) {
                    Text(text = " .\n * +1   .\n * -1   .")
                    Row(modifier = Modifier.align(Alignment.End)) {
                        Button(onClick = {
                            openDialog = false
                        }) {
                            Text(text = "")
                        }
                        Button(onClick = {
                            counter++
                            openDialog = false
                        }) {
                            Text(text = "+1")
                        }
                        Button(onClick = {
                            counter--
                            openDialog = false
                        }) {
                            Text(text = "-1")
                        }
                    }
                }
            }
        }
    }
}
```

DropDownMenu

```
@Composable
fun DropDownMenuExample() {
    var expandDropDownMenu by remember { mutableStateOf(false) }
    var counter by remember { mutableStateOf(0) }

    Column {
        Button(onClick = { expandDropDownMenu = true }) {
            Text("  ")
        }
        Text(": $counter")
    }

    DropdownMenu(
        expanded = expandDropDownMenu,
        onDismissRequest = {
            expandDropDownMenu = false
        }
    ) {
        DropdownMenuItem(
            text = {
                Text(text = "")
            },
            onClick = {
                counter++
            }
        )
        DropdownMenuItem(
            text = {
                Text(text = "")
            },
            onClick = {
                counter--
            }
        )
    }
}
```



SnackBar

```
@Composable
fun SnackbarExample() {
    var counter by remember { mutableStateOf(0) }

    val coroutineScope = rememberCoroutineScope()
    val snackbarHostState = remember { SnackbarHostState() }
    Scaffold(
        snackbarHost = { SnackbarHost(hostState = snackbarHostState) }
    ) { paddingValues ->
        Button(
            onClick = {
                counter++
                coroutineScope.launch {
                    snackbarHostState.showSnackbar(
                        message = " ${counter}.",
                        actionLabel = "",
                        duration = SnackbarDuration.Short
                    )
                }
            },
            modifier = Modifier.padding(paddingValues)
        ) {
            Text("")
        }
    }
}
```

BottomAppBar

```
@Composable
fun BottomAppBarExample() {
    val snackbarHostState = remember { SnackbarHostState() }
    val coroutineScope = rememberCoroutineScope()
    var counter by remember { mutableStateOf(0) }

    Scaffold(
        snackbarHost = { SnackbarHost(hostState = snackbarHostState) },
        bottomBar = {
            BottomAppBar {
                Text(text = "")
                Button(onClick = {
                    coroutineScope.launch {
                        snackbarHostState.showSnackbar("")
                    }
                }) {
                    Text(text = "")
                }
                Button(onClick = {
                    counter++
                    coroutineScope.launch {
                        snackbarHostState.showSnackbar("${counter}")
                    }
                }) {
                    Text(text = "")
                }
                Button(onClick = {
                    counter--
                    coroutineScope.launch {
                        snackbarHostState.showSnackbar("${counter}")
                    }
                }) {
                    Text(text = "")
                }
            }
        }
    ) { paddingValues ->
        Box(
            modifier = Modifier
                .fillMaxSize()
                .padding(paddingValues)
        ) {
            Text(
                text = " ${counter}.",
                modifier = Modifier.align(Alignment.Center)
            )
        }
    }
}
```

Animation

```
@Composable
fun AnimationExample() {
    var helloWorldVisible by remember { mutableStateOf(true) }
    var isRed by remember { mutableStateOf(false) }

    val backgroundColor by animateColorAsState(
        targetValue = if (isRed) Color.Red else Color.White
    )

    val alpha by animateFloatAsState(
        targetValue = if (isRed) 1.0f else 0.5f
    )

    Column(
        modifier = Modifier
            .padding(16.dp)
            .background(backgroundColor)
            .alpha(alpha)
```

```
) {
    // expand(), scale(), slide(), fade()
    AnimatedVisibility(
        visible = helloWorldVisible,
        enter = slideInHorizontally() + expandVertically(),
        exit = slideOutHorizontally()
    ) {
        Text(text = "Hello World!")
    }

    Row(
        Modifier.selectable(
            selected = helloWorldVisible,
            onClick = {
                helloWorldVisible = true
            }
        ),
        verticalAlignment = Alignment.CenterVertically
    ) {
        RadioButton(
            selected = helloWorldVisible,
            onClick = { helloWorldVisible = true }
        )
        Text(
            text = "Hello World "
        )
    }

    Row(
        Modifier.selectable(
            selected = !helloWorldVisible,
            onClick = {
                helloWorldVisible = false
            }
        ),
        verticalAlignment = Alignment.CenterVertically
    ) {
        RadioButton(
            selected = !helloWorldVisible,
            onClick = { helloWorldVisible = false }
        )
        Text(
            text = "Hello World "
        )
    }

    Text(text = "  .")

    Row(
        Modifier.selectable(
            selected = !isRed,
            onClick = {
                isRed = false
            }
        ),
        verticalAlignment = Alignment.CenterVertically
    ) {
        RadioButton(
            selected = !isRed,
            onClick = { isRed = false }
        )
        Text(
            text = ""
        )
    }

    Row(
        Modifier.selectable(
            selected = isRed,
            onClick = {
                isRed = true
            }
        ),
        verticalAlignment = Alignment.CenterVertically
    ) {
        RadioButton(
```

```
                selected = isRed,
                onClick = { isRed = true }
            )
            Text(
                text = ""
            )
        }
    }
}
```

```
@Composable
fun Animation2Example() {
    var isDarkMode by remember { mutableStateOf(false) }

    val transition = updateTransition(targetState = isDarkMode, label = "  ")
    val backgroundColor by transition.animateColor(label = "   ") { state ->
        if (state) {
            Color.Black
        } else {
            Color.White
        }
    }
    val textColor by transition.animateColor(label = "    ") { state ->
        if (state) {
            Color.White
        } else {
            Color.Black
        }
    }
    val alpha by transition.animateFloat(label = "   ") { state ->
        if (state) {
            1.0f
        } else {
            0.7f
        }
    }

    Column(
        modifier = Modifier
            .background(backgroundColor)
            .alpha(alpha)
    ) {
        RadioButtonWithText(text = " ", color = textColor, selected = !isDarkMode) {
            isDarkMode = false
        }
        RadioButtonWithText(text = " ", color = textColor, selected = isDarkMode) {
            isDarkMode = true
        }

        Crossfade(targetState = isDarkMode) { state ->
            if (state) {
                Row {
                    Box(
                        modifier = Modifier
                            .background(Color.Red)
                            .size(20.dp)
                    ) {
                        Text("A")
                    }
                    Box(
                        modifier = Modifier
                            .background(Color.Magenta)
                            .size(20.dp)
                    ) {
                        Text("B")
                    }
                    Box(
                        modifier = Modifier
                            .background(Color.Blue)
                            .size(20.dp)
                    ) {
                        Text("C")
                    }
                }
```

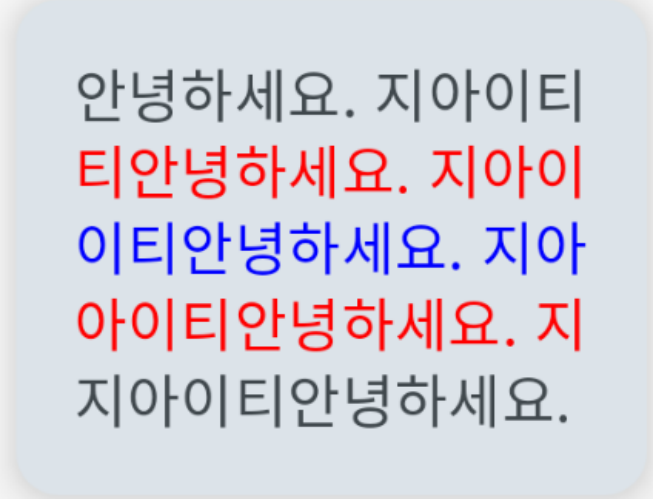
```
            } else {
                Column {
                    Box(
                        modifier = Modifier
                            .background(Color.Red)
                            .size(20.dp)
                    ) {
                        Text("1")
                    }
                    Box(
                        modifier = Modifier
                            .background(Color.Magenta)
                            .size(20.dp)
                    ) {
                        Text("2")
                    }
                    Box(
                        modifier = Modifier
                            .background(Color.Blue)
                            .size(20.dp)
                    ) {
                        Text("3")
                    }
                }
            }
        }
    }
}

@Composable
fun RadioButtonWithText(
    text: String,
    color: Color = Color.Black,
    selected: Boolean,
    onClick: () -> Unit
) {
    Row(
        modifier = Modifier.selectable(
            selected = selected,
            onClick = onClick
        ),
        verticalAlignment = Alignment.CenterVertically
    ) {
        RadioButton(selected = selected, onClick = onClick)
        Text(text = text, color = color)
    }
}
```

Side Effect

```
@Composable
fun EffectExample(lifecycleOwner: LifecycleOwner = LocalLifecycleOwner.current) {
    val snackbarHostState by remember {
        mutableStateOf(SnackbarHostState())
    }

    LaunchedEffect(key1 = snackbarHostState) {
        snackbarHostState.showSnackbar("Hello World!")
    }

    DisposableEffect(key1 = lifecycleOwner) {
        val observer = LifecycleEventObserver { _, event ->
            when (event) {
                Lifecycle.Event.ON_START -> {
                    Log.d("", "ON_START")
                }
                Lifecycle.Event.ON_RESUME -> {
                    Log.d("", "ON_RESUME")
                }
                Lifecycle.Event.ON_PAUSE -> {
                    Log.d("", "ON_PAUSE")
                }
                Lifecycle.Event.ON_STOP -> {
                    Log.d("", "ON_STOP")
                }
                else -> {
                    Log.d("", " ")
                }
            }
        }

        lifecycleOwner.lifecycle.addObserver(observer)

        onDispose {
            lifecycleOwner.lifecycle.removeObserver(observer)
        }
    }

    Scaffold(
        snackbarHost = { SnackbarHost(hostState = snackbarHostState) }
    ) { paddingValues ->
        Text(
            text = "",
            modifier = Modifier.padding(paddingValues)
        )
    }
}
```

CompositionLocal

```
val LocalElevation = compositionLocalOf { 8.dp }

@Composable
fun CompositionLocalExample() {
    CompositionLocalProvider(LocalElevation provides 12.dp) {
        Card(
            modifier = Modifier.padding(8.dp),
            elevation = CardDefaults.cardElevation(LocalElevation.current)
        ) {
            Column(
                modifier = Modifier.padding(16.dp),
            ) {
                Text(". ")
                CompositionLocalProvider(LocalContentColor provides Color.Red) {
                    Text(". ")
                    CompositionLocalProvider(LocalContentColor provides Color.Blue) {
                        Text(". ")
                    }
                    Text(". ")
                }
                Text(".")
            }
        }
    }
}
```



Navigation

```
@Composable
fun NavigationExample(
    modifier: Modifier = Modifier,
    navController: NavHostController = rememberNavController()
) {
    NavHost(navController = navController, startDestination = "Home", modifier = modifier) {
        composable("Home") {
            Column {
                Text(text = "Home")
                Button(onClick = {
                    navController.navigate("Playground")
                }) {
                    Text(text = "Playground  ")
                }
                Button(onClick = {
                    navController.navigate("Office")
                }) {
                    Text(text = "Office  ")
                }
                Button(onClick = {
                    navController.navigate("Home") {
                        launchSingleTop = true
                    }
                }) {
                    Text(text = "Home  ")
                }
                Button(onClick = {
                    navController.navigate("Argument/GIT")
                }) {
                    Text(text = "GIT  ")
                }
            }
        }
        composable("Office") {
            Column {
                Text(text = "Office")
                Button(onClick = {
                    navController.navigate("Playground")
                }) {
                    Text(text = "Playground  ")
                }
                Button(onClick = {
                    navController.navigate("Home")
                }) {
                    Text(text = "Home  ")
                }
            }
        }
        composable("Playground") {
            Column {
                Text(text = "Playground")
                Button(onClick = {
                    navController.navigate("Office")
                }) {
                    Text(text = "Office  ")
                }
                Button(onClick = {
                    navController.navigate("Home")
                }) {
                    Text(text = "Home  ")
                }
            }
        }

        composable("Argument/{userId}") {
            val userId = it.arguments?.getString("userId")
            Text("userId: $userId")
        }
    }
}
```

Unidirectional Data Flow

```
var name by remember { mutableStateOf("") }
OutlinedTextField(
        value = name,
        onValueChange = { name = it },
        label = { Text("Name") }
)
```

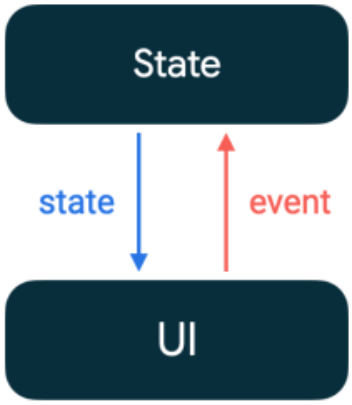
Compose 의 UI 는 수정될 수 없음.

UI 상태가 변결될 때만 변경된 UI 트리를 다시 만듬.

TextField 는 값을 받고 onValueChange 를 노출함.

상태를 노출하고 이벤트를 노출 → 단방향 데이터 흐름이 적합

- 이벤트, 상태



이벤트: UI의 일부에서 이벤트를 생성해 위로 전달. 뷰 모델에서처리하는 버튼 클릭, 사용자 세션의 만료됨을 표현하는 앱의 다른 레이어에서 전달되는 이벤트.
상태 갱신: 이벤트 핸들러가 상태를 바꿉니다.
상태 표시: 스테이트 홀더가 아래로 전달한 상태를 UI가 표시합니다.

- UDF 를 사용했을 때의 장점

테스트 가능성: 상태를 표시하는 UI와 상태를 분리하면, 둘을 분리해 쉽게 테스트할 수 있습니다.
상태 캡슐화: 상태가 한 곳에서만 갱신될 수 있으면, 컴포저블 상태를 위한 단일 정보 진실원(SSOT, single source of truth)가 될 수 있고, 일관되지 않는상태 때문에 발생하는 버그를 줄일 수 있다.
UI 일관성: StateFlow와 LiveData와 같은 관측가능한 상태 홀더를 사용해 UI에 상태 갱신을 즉각 반영할 수 있다.

- 단방향 데이터 흐름이 적합한 Compose

```
@Composable
fun Header(title: String, subtitle: String) {
        // Recomposes when title or subtitle have changed.
}
@Composable
fun Header(news: News) {
        // Recomposes when a new instance of News is passed in.
}
```

첫 번째 Header는 title이나 subtitle이 갱신되면 리컴포지션이 되는데,
아래는 News 중 일부가 변경되면 리컴포지션.
Header에서 title와 subtitle만 쓴다면 아래와 같이 짜는 것이 좋지 않음.