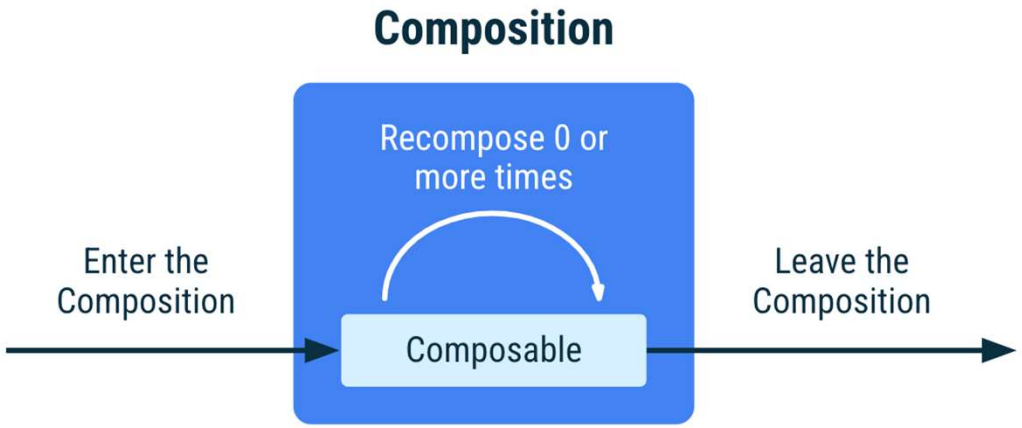
04

04-3. Layout Composable

Compose

Composable Lifecycle

- 액티비티, 프레그먼트 처럼 컴포저블도 라이프사이클을 가진다.
- 하지만 액티비티, 프레그먼트처럼 복잡하지는 않다.
  - Composition 진입 상태
  - Re-Composition 상태
  - Composition 종료 상태



Composable Lifecycle

Compose Three Phase

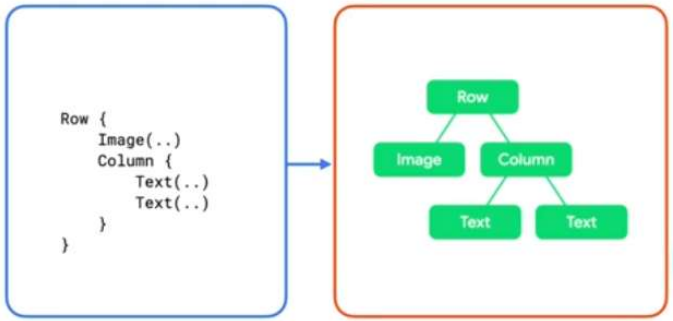
- Compose 는 Data 를 화면에 출력하기 위해서 Composition, Layout, Drawing 3단계를 거친다.

- Composition : What – 무엇을 그릴 것인지를 결정하는 단계
- Layout : Where – 어디에 그릴 것인지를 결정하는 단계
- Drawing : How – 어떻게 그릴 것인지를 결정하는 단계

Composable Lifecycle

Composition

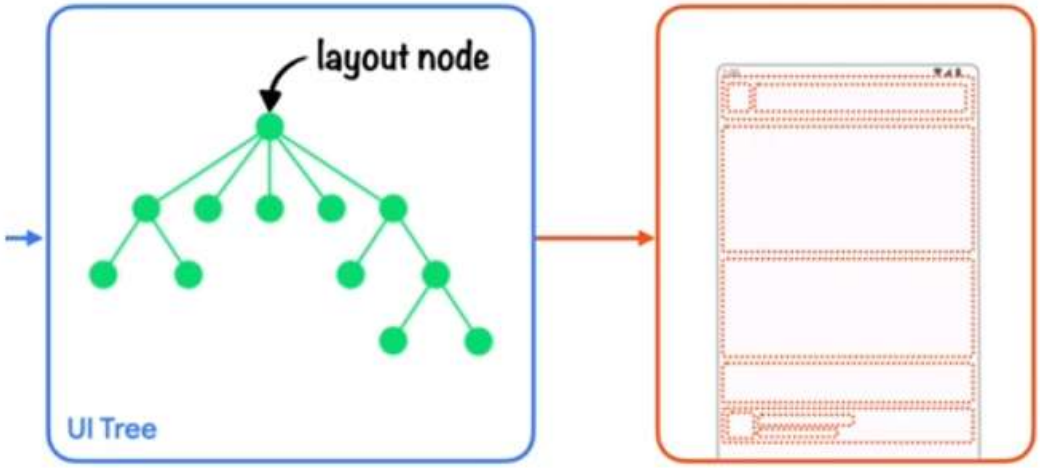
- 화면을 위한 Tree data structure 를 구성하는 단계이다.
- Composition 은 Composable 함수를 호출하여 단일 트리 구조를 완성하는 단계를 지칭한다.



Composable Lifecycle

Layout

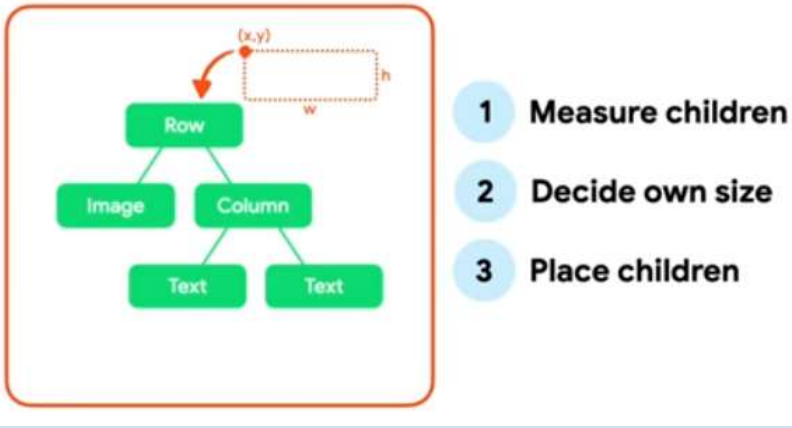
- Composition 에 의한 데이터 트리를 어디에 그릴 것인지를 결정하는 단계이다.
- Composition 단계에서 만들어진 data tree 를 이용해 각 노드의 사이즈, 출력 위치 등이 결정되는 단계이다.



Composable Lifecycle

Layout

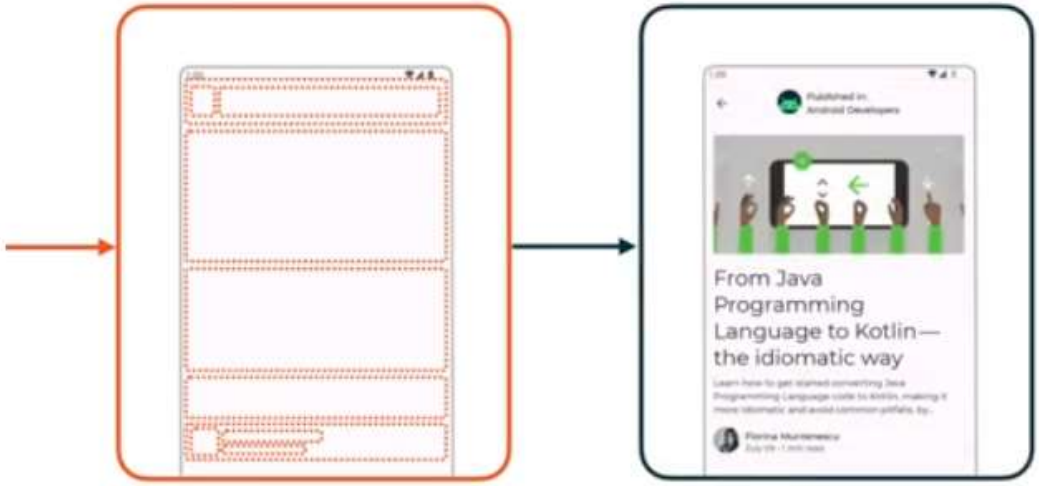
- Layout 단계에서 3가지 작업이 이루어진다.
  - Measure Children : 자식 노드가 있는지 판단하고 만약 있다면 자식노드의 Layout(사이즈, 위치등)을 먼저 판단한다.
  - Decide own size : 자식 노드에 대한 판단이 끝난 후 자식 노드의 결과를 참조하여 자신 노드의 Layout 을 결정한다.
  - Place Children : 자식 노드를 자신에 배치시키는 작업을 한다.
- 이 작업을 거치면서 각 노드들이 어느 위치(x, y) 에 어느 사이즈(width, height) 로 출력되어야 하는지가 결정되게 된다.



Composable Lifecycle

Drawing

- 실제 그림을 그리는 단계



Modifier

모디파이어 이용하기

- 모디파이어란 Modifier 객체 타입으로 표현되는 컴포저블의 설정 정보
- 컴포저블을 위한 공통 설정은 모디파이어를 사용

- 컴포즈의 내장 컴포저블은 모두 매개변수로 modifier가 선언 되어 있습니다.
- 컴포저블을 설정하기 위한 모디파이어에는 100개 이상의 함수 가 존재
- 패딩을 설정하기 위한 padding(), 컴포저블의 테두리를 설정하기 위한 border()라는 함수 등

• modifier로 컴포저블 함수 설정

```
val modifier = Modifier
    .border(width = 10.dp, color = Color.Red)
    .padding(all = 30.dp)
    .background(Color.Yellow)
Text(
    text = "Hello World!",
    fontSize = 30.sp,
    modifier = modifier
)
```

▶실행 결과

Modifier

- then 함수를 이용해 다른 모디파이어를 조합할 수도 있습니다.
- 기본 설정을 위한 모디파이어 객체를 하나 준비하고 다른 모디파이어 객체에 새로운 설정을 추가하거나 기본 설정값을 변경

• then 함수로 모디파이어 추가

```
val modifier = Modifier
    .border(width = 10.dp, color = Color.Red)
    .padding(all = 30.dp)
    .background(Color.Yellow)

val secondModifier = Modifier.background(Color.Blue)
Text(
    text = "Hello World!",
    fontSize = 30.sp,
    modifier = modifier.then(secondModifier)
)
```

Modifier

속성 선언 순서

- Modifier에 다양한 속성을 선언할 때 선언 순서가 중요하다. 속성의 적용순서이다.
- 예를 들어 padding 과 clickable 의 위치를 변경한 경우 click 되는 영역에 차이가 있다.

- Clickable 이 먼저 선언되었다면 컴포저블 전체 영역이 click 된다. 하지만 padding 이 먼저 선언되었다면 padding 이 적용된 영역만 click 된다.

clickable -> padding

padding -> clickable

Modifier

background

```
Card(modifier = Modifier.background(Color.Blue)) {
    Text("Hello")
}
```

size/ width/ height

```
Modifier.background(Color.Blue)
//              .size(200.dp, 100.dp)
            .height(100.dp)
            .width(200.dp)
```

fillMaxSize()/ fillMaxWidth() / fillMaxHeight()

Modifier

requiredSize

- 사이즈 지정 시 부모의 사이즈보다 자식의 사이즈가 크게 지정되면 적용되지 않을 수 있다.
- 이때 자식의 사이즈를 requiredSize 로 지정하면 부모에 제약되지 않는 자식의 사이즈를 지정할 수 있다.

```
Row(modifier = Modifier.size(100.dp, 100.dp)){
    Card(modifier = Modifier.requiredSize(200.dp,
200.dp).background(Color.Red)){
        Text("hello")
    }
}
```

Modifier

padding

- Margin 은 따로 없다. padding 을 이용하거나 Spacer() 컴포저블을 이용해야 한다.

```
Modifier.
//              .padding(24.dp)
//              .padding(bottom = 24.dp, top = 24.dp, start = 24.dp, end = 24.dp)
              .padding(vertical = 24.dp, horizontal = 24.dp)
```

clickable

```
Card(modifier = Modifier.background(Color.Blue)
    .size(200.dp, 100.dp)
    .padding(24.dp)
    .clickable {
        Toast.makeText(context, "click", Toast.LENGTH_SHORT).show()
    }
) {
    Text("Hello")
}
```

Modifier

weight

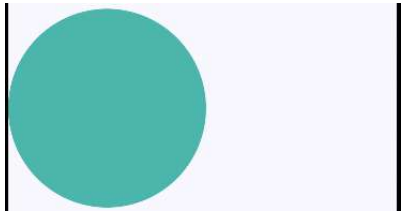
- Row / Column 의 modifier에만 적용

```
Row(modifier = Modifier.fillMaxWidth()){
    Card(modifier = Modifier.weight(2f).background(Color.Blue)){
        Text("hello")
    }
    Card(modifier = Modifier.weight(1f).background(Color.Red)){
        Text("hello")
    }
    Card(modifier = Modifier.weight(2f).background(Color.Green)){
        Text("hello")
    }
}
```

Modifier

clip

```
Box(
    modifier = Modifier
        .size(200.dp)
        .clip(CircleShape)
        .background(Color(0xFF4DB6AC))
```



Row, Column

Row와 Column 함수

- 컴포저블 여러 개를 등록하고 배치하기 위한 레이아웃
- Row, Column, Box, ConstraintLayout
- Row는 가로 방향 배 치에 관여하고 Column이 세로 방향 배치

• Row 컴포저블 함수를 활용해 가로 방향으로 문자열을 배치

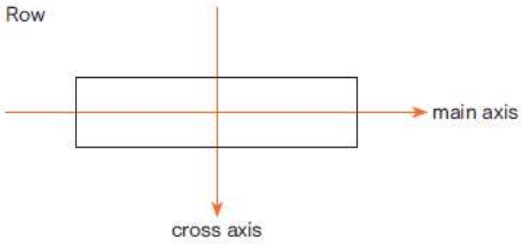
```
@Composable
fun MainScreen() {
    Column {
        Row(modifier = Modifier.background(Color.Yellow)) {
            MyText(data = "A")
            MyText(data = "B")
            MyText(data = "C")
        }
    }

}
```

Row, Column

정렬

- 배치되는 방향을 주축(main axis), 반대 방향을 반대축(cross axis)이라 부르며 Row 의 주축은 가로 방향이고, 반대축은 세로 방향입니다.
- Column의 경우 주축은 세로 방 향이고, 반대축은 가로 방향



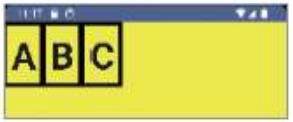
Row, Column

- 정렬은 Alignment 혹은 Arrangement 객체를 이용하는데 주축에 대한 배치는 Arrangement를, 반대축의 배치는 Alignment를 이용
- Row, Column이 차지하는 화면 영역에 추가되는 컴포저블은 기본으로 좌측 상단에 정렬

• Alignment나 Arragement를 사용하지 않은 경우

```
Row(modifier = Modifier
    .background(Color.Yellow)
    .fillMaxWidth()
    .height(300.dp)
) {
    MyText(data = "A")
    MyText(data = "B")
    MyText(data = "C")
}
```

▶ 실행 결과



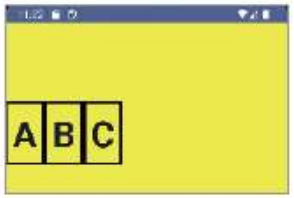
Row, Column

- Row에는 verticalAlignment 속성을 이용해 수직 방향의 정렬 위치를 설정합니다.
  - Alignment.Top: 상단 정렬
  - Alignment.CenterVertically: 세로 방향 중앙 정렬
  - Alignment.Bottom: 하단 정렬

• Row에 verticalAlignment 속성을 활용

```
Row(modifier = Modifier
    .background(Color.Yellow)
    .fillMaxWidth()
    .height(300.dp),
    verticalAlignment = Alignment.CenterVertically
) {
    MyText(data = "A")
    MyText(data = "B")
    MyText(data = "C")
}
```

▶ 실행 결과



Row, Column

- Column에서는 horizontalAlignment 속성을 이용해 수평 방향의 정렬 위치를 설정
  - Alignment.Start: 왼쪽 정렬
  - Alignment.CenterHorizontally: 가로 방향 중앙 정렬
  - Alignment.End: 오른쪽 정렬

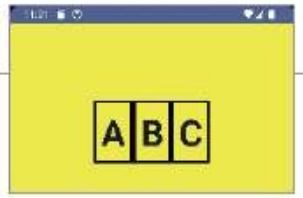
- Arrangement를 이용해 Row의 가로 방향을 정렬할 때 horizontalArrangement 속성을 이용합 니다.
- Column에서 세로 방향 정렬을 하고자 한다면 verticalArrangement를 이용하여 Arrangement.Top, Arrangement.Bottom 등을 지정
  - Arrangement.Start: 가로 방향 왼쪽 정렬
  - Arrangement.Center: 중앙 정렬
  - Arrangement.End: 가로 방향 오른쪽 정렬
  - Arrangement.Top: 세로 방향 윗쪽 정렬
  - Arrangement.Bottom: 세로 방향 아래 정렬
  - Arrangement.SpaceEvenly: 컴포저블 사이에 균등한 여백을 주고 정렬
  - Arrangement.SpaceBetween: 컴포저블 사이를 균등한 여백으로 지정. 양쪽 끝 컴포저블과 레이아 웃의 간격은 없음.
  - Arrangement.SpaceAround: 각 컴포저블 왼쪽, 오른쪽의 여백을 동일하게 지정해 정렬

Row, Column

• Row에 Alignment와 Arrangement 속성을 활용

```
Row(modifier = Modifier
    .background(Color.Yellow)
    .fillMaxWidth()
    .height(300.dp),
    verticalAlignment = Alignment.CenterVertically,
    horizontalArrangement = Arrangement.Center
) {
    (... 생략 ...)
}
```

▶ 실행 결과

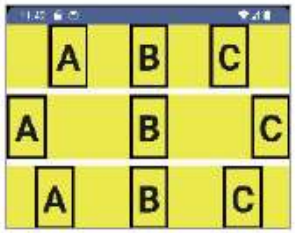


Row, Column

• Arrangement.SpaceEvenly, Arrangement.SpaceBetween, Arrangement.SpaceAround 사용 예

```
Row(modifier = Modifier
    horizontalArrangement = Arrangement.SpaceEvenly
) {
    (... 생략 ...)
}
Row(modifier = Modifier
    horizontalArrangement = Arrangement.SpaceBetween
) {
    (... 생략 ...)
}
Row(modifier = Modifier
    horizontalArrangement = Arrangement.SpaceAround
) {
    (... 생략 ...)
}
```

▶ 실행 결과



Row, Column

스코프 모디파이어

- Row, Column에 추가되는 여러 컴포저블 함수 가 동일 스코프
- 스코프 모디파이어는 Row, Column에 설정되는 것이 아니라 Row, Column에 추가되는 개별 컴포저블에 설정
- 자신이 포함된 스코프(Row 혹은 Column 영역) 내에 어떻게 배치되고, 어떤 크기를 가질지 를 지정하기 위해 사용되는 모디파이어 속성

```
Row {this: RowScope
    MyText(data = "A")
    MyText(data = "B")
    MyText(data = "C")
}
```

Row, Column

- 대표적인 스코프 모디파이어로 weight 함수
- weight()는 스코프 내에서 개별 컴포저블의 사이즈를 지정하기 위해서 사용
- size()는 동일 스코프에 지 정되는 다른 컴포저블을 고려하지 않고 자신의 사이즈를 결정
- weight()는 다른 컴포저블의 weight()값을 같이 이용해 사이즈를 결정

• weight 값에 따른 사이즈 지정

```
Row(modifier = Modifier.background(Color.Yellow)) {
    MyText(data = "A", modifier = Modifier.weight(1f))
    MyText(data = "B", modifier = Modifier.weight(2f))
    MyText(data = "C", modifier = Modifier.weight(1f))
}
```

▶ 실행 결과

Row, Column

- `fill` 매개변숫값을 true/ false로 지정할 수 있는데 `fill`이 true면 가중치에 의해 계산된 사이즈만큼 화면에 차지하게 되고 false면 가중치에 의해 계산된 사이즈가 무시됩니다.

• fill에 의한 사이즈 적용 여부

```
Row(modifier = Modifier.background(Color.Yellow)) {
    MyText(data = "A", modifier = Modifier.weight(weight = 1f, fill = true))
    MyText(data = "B", modifier = Modifier.weight(weight = 2f, fill = true))
    MyText(data = "C", modifier = Modifier.weight(weight = 1f, fill = false))
}
```

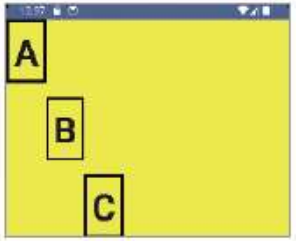
▶ 실행 결과

Row, Column

- align 함수로 지정되는 모디파이어는 개별 컴포저블에 설정해 동일 스코프 내에 컴포저블이 어느 위치에 배치될 것인지를 결정
- Alignment.CenterHorizontally, Alignment.Start, Alignment.End, Alignment.CenterVertically, Alignment.Top, Alignment.Bottom을 적용해 개별 컴포저블이 동일 스코프에서 어느 위치에 포함되어야 하는 지를 지정

• align 함수를 이용해 모디파이어 지정

```
Row(modifier = Modifier.background(Color.Yellow).height(300.dp).fillMaxWidth()) {
    MyText(data = "A", modifier = Modifier.align(Alignment.Top))
    MyText(data = "B", modifier = Modifier.align(Alignment.CenterVertically))
    MyText(data = "C", modifier = Modifier.align(Alignment.Bottom))
}
```



Box

Box

- Box에 추가되는 컴포저블을 동일한 위치에 겹쳐서 출력

• Box 함수의 레이아웃 사용

```
Box {
    MyText(data = "A")
    MyText(data = "B")
}
```

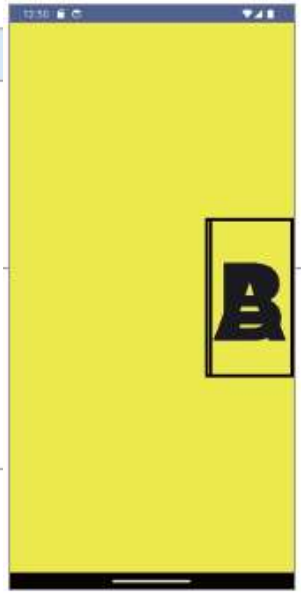
▶ 실행 결과

Box

- Box 함수에 추가되는 컴포저블의 정렬은 contentAlignment를 이용하여 Alignment.TopStart, Alignment.TopCenter, Alignment.TopEnd, Alignment.CenterStart, Alignment.Center, Alignment.CenterEnd, Alignment.BottomStart, Alignment.BottomCenter, Alignment.BottomEnd 등을 사용해 정렬

• Box 함수에 추가되는 컴포저블 정렬

```
Box(
    modifier = Modifier
        .background(color = Color.Yellow),
    contentAlignment = Alignment.CenterEnd
) {
    MyText(data = "A")
    MyText(data = "B")
}
```

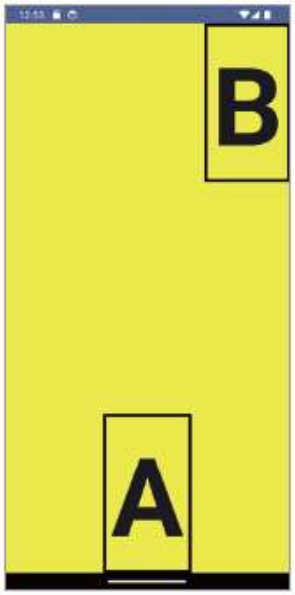


Box

- BoxScope 내에 추가되는 컴포저블에 align 모디파이어를 사 용하면 개별 컴포저블의 위치를 지정할 수도 있습니다.

• align 모디파이어 사용

```
Box(
    modifier = Modifier
        .background(color = Color.Yellow)
) {
    MyText(data = "A", modifier = Modifier.align(Alignment.BottomCenter))
    MyText(data = "B", modifier = Modifier.align(Alignment.TopEnd))
}
```



Box

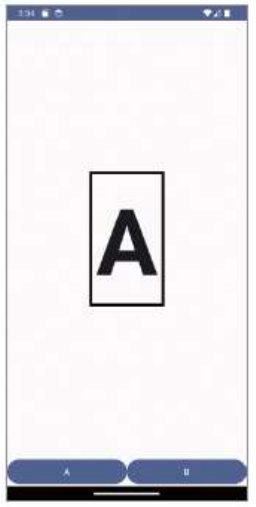
- 화면에 나타나거나 사라지게 하는 AnimatedVisibility 컴포저블이 제공

• AnimatedVisibility 함수를 활용해 한 박스만 노출

```
var aVisible by remember {
    mutableStateOf(mutableListOf(true, false))
}
Column {
    Box(modifier = Modifier.weight(1f).fillMaxWidth()) {
        this@Column.AnimatedVisibility(
            visible = aVisible.get(0), enter = fadeIn(), exit = fadeOut(),
            modifier = Modifier.align(Alignment.Center)
        ) {
            MyText(data = "A")
        }

        this@Column.AnimatedVisibility(
            visible = aVisible.get(1), enter = fadeIn(), exit = fadeOut(),
            modifier = Modifier.align(Alignment.Center)
        ) {
            MyText(data = "B")
        }
    }
    Row {
        Button(modifier = Modifier.weight(1f), onClick = {
            aVisible = mutableListOf(true, false)
        }) {
            Text("A")
        }
        Button(modifier = Modifier.weight(1f),onClick = {
            aVisible = mutableListOf(false, true)
        }) {
            Text("B")
        }
    }
}
```

▶ 실행 결과



ConstraintLayout

- ConstraintLayout은 다양한 조건으로 컴포저블을 배치하기 위한 레이아웃

• ConstraintLayout 사용 선언

```
implementation("androidx.constraintlayout:constraintlayout-compose:1.1.0")
```

- ConstraintLayout에는 부모 혹은 다른 컴포저블을 기준으로 컴포저블의 배치 조건을 명시
- 다른 컴포저블을 지정하기 위해서는 컴포저블을 식별할 수 있는 식별자가 있어야 하는데 이것을 참조라고 합니다.
- 참조 선언은 이와 같이 createRef()를 이용
- 여러 개의 참조를 선언하고 싶으면 createRefs()를 이용

• 참조 선언

```
val text1 = createRef()
val (button1, button2) = createRefs()
```

ConstraintLayout

- ConstraintLayout : 참조 정보를 활용해 제약조건으로 배치하는 컴포저블
- ConstraintSet : 제약 조건을 별도로 관리하기 위한 객체, UI 구성과 제약 조건을 분리해서 개발하는 경우에 이용, 필수는 아님

- createRef(), createRefs() 와 createRefFor(), createRefsFor() 모두 참조를 선언하는 면에서는 동일하다.
- createRef(), createRefs() 의 경우 ConstraintLayoutScope 내에서 사용
- createRefFor(), createRefsFor() 은 ConstraintSetScope 에서 사용

ConstraintLayout

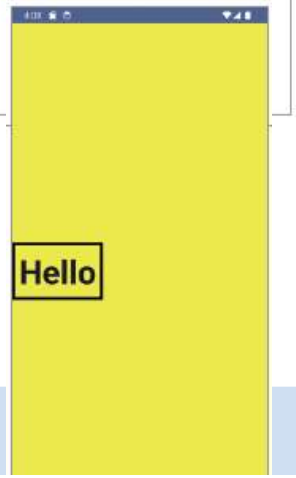
- 선언된 참조를 특정 컴포저블에 지정하려면 다음과 같이 constrainAs 모디파이어를 이용합 니다.
- 위치에 관한 조 건은 "나의 위치.linkTo(대상의 위치)" 형태로 작성합니다.

• constrainAs로 선언된 참조를 특정 컴포저블에 지정

```
ConstraintLayout(
        Modifier
            .size(150.dp, 150.dp)
            .background(Color.Yellow)) {
        //.....
        MyText(data="Hello", modifier = Modifier.constrainAs(text1){
                top.linkTo(parent.top, margin = 10.dp)
                bottom.linkTo(parent.bottom, margin = 10.dp)
        })

}
```

▶ 실행 결과



ConstraintLayout

- 다른 컴포저블을 기준으로 컴포저블의 위치를 지정하고 싶다면 기준이 되는 컴 포저블에 미리 선언한 참조를 이용

• 선언된 참조를 사용해 다른 컴포저블 기준으로 위치 지정

```
Button(onClick = { /*TODO*/ }, modifier = Modifier.constrainAs(button1){
    top.linkTo(text1.bottom, margin=10.dp)
    start.linkTo(text1.end, margin = 10.dp)
}) {
    Text(text = "button1")
}
Button(onClick = { /*TODO*/ }, modifier = Modifier.constrainAs(button2){
    centerVerticallyTo(parent)
    centerHorizontallyTo(button1)
}) {
    Text(text = "button2")
}
```

▶실행 결과

ConstraintLayout - ConstraintSet

- ConstraintLayout 을 사용할 때 ConstraintLayout 과 제약조건 선언을 분리해서 작성
- ConstraintSet 을 이용해 제약조건을 선언하고 ConstraintSet 에 선언된 제약조건을 ConstraintLayout 에서 이용
- 코드의 간결함 혹은 동적 제약조건 지정등을 구현할 수 있게 된다.

- ConstraintSet 에서 참조를 선언할 때 createRefFor(), createRefsFor() 를 이용
- createRefFor(), createRefsFor() 의 매개변수에 참조의 별칭을 문자열로 지정하고 ConstraintLayout 에서는 그 별칭으로 참조를 이용

ConstraintLayout - ConstraintSet

```
private fun myConstraint(): ConstraintSet {
    return ConstraintSet {
        val button = createRefFor("button2")
        val text = createRefFor("text2")

        constrain(button) {
            top.linkTo(parent.top, margin = 10.dp)
            start.linkTo(parent.start, margin = 10.dp)
        }
        constrain(text) {
            top.linkTo(button.bottom, margin = 10.dp)
            start.linkTo(button.end, margin = 10.dp)
        }
    }
}
```

```
val constraints1 = myConstraint()
ConstraintLayout(
    constraints1,
    modifier = Modifier.background(Color.Green)
){
    Button(onClick = {}, modifier = Modifier.layoutId("button2")) {
        Text("Button1")
    }
    Text("Hello", modifier = Modifier.layoutId("text2"))
}
```

ConstraintLayout - Guideline

- 화면에 부모를 기준으로 임의 선을 긋고 그 선을 기준으로 제약 조건을 지정하는 방법이다.
- 가로 가이드라인은 top, bottom 기준이며 세로 가이드라인은 start, end 기준이다.

  - createGuidelineFromStart(0.1f)
  - createGuidelineFromEnd(16.dp)
  - createGuidelineFromTop(0.1f)
  - createGuidelineFromBottom(0.1f)

- 가이드라인을 각 함수로 선언하면서 매개변수에 0.5f 처럼 값을 지정하면 % 개념이다. 즉 부모의 start 에서 50% 지점에 세로 가이드 라인 선을 선언하겠다는 의미이다.
- 100.dp 처럼 매개변수를 지정하면 부모의 top 에서 100dp 떨어진 지점에 가이드라인을 지정하겠다는 의미이다.

ConstraintLayout - Guideline

```
ConstraintLayout(
    modifier = Modifier
        .background(Color.Yellow)
        .size(300.dp, 200.dp)
) {
    val guidelineStart = createGuidelineFromStart(0.5f)
    val guidelineTop = createGuidelineFromTop(100.dp)

    val (button1) = createRefs()

    Button(onClick = {}, modifier = Modifier.constrainAs(button1) {
        top.linkTo(guidelineTop, margin = 0.dp)
        start.linkTo(guidelineStart, margin = 0.dp)
    }) {
        Text("Button1")
    }

}
```

ConstraintLayout - Barrier

- 가상의 선을 만들기 위함은 가이드 라인과 동일하다.
- 가이드 라인은 부모를 기준으로 % 혹은 숫치 값을 이용해 가이드 라인을 지정함으로 정적이다.
- 배리어를 이용하면 다른 컴포저블을 지준으로 가상의 선을 구성할 수 있게 된다.
- 기준이 되는 컴포저블의 사이즈 변화에 따라 동적인 선이 만들어 질 수 있으며 여러 컴포저블을 묶어서 선을 만들 수도 있다.

```
val barrier1 = createEndBarrier(text1, text2)

    Button(onClick = {}, modifier = Modifier.constrainAs(button) {
        top.linkTo(text2.bottom, margin = 10.dp)
        start.linkTo(barrier1, margin = 10.dp)
    }) {
        Text("Button")
    }
```

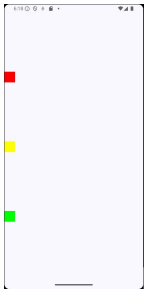
ConstraintLayout - Chain

- 체인은 가로 혹은 세로 축으로 컴포저블을 그룹핑 하여 여백을 나누기 위해 사용된다.

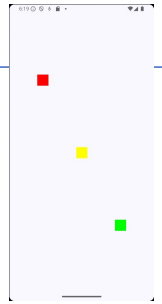
  - createVerticalChain()
  - createHorizontalChain()

```
createHorizontalChain(box1, box2, box3)
```


```
createVerticalChain(box1, box2, box3)
```



```
createHorizontalChain(box1, box2, box3)
createVerticalChain(box1, box2, box3)
```

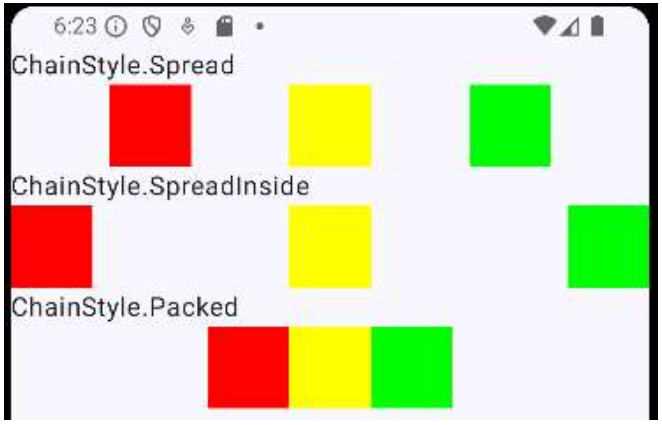


ConstraintLayout - Chain

- 체인을 선언하면서 여백을 어떻게 나눌 것인지를 ChainStyle 로 지정해 주어야 한다.

  - ChainStyle.Spread
  - ChainStyle.SpreadInside
  - ChainStyle.Packed

```
createHorizontalChain(box1, box2, box3, chainStyle = ChainStyle.Spread)
```



감사합니다

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

윌리엄 셰익스피어
William Shakespeare