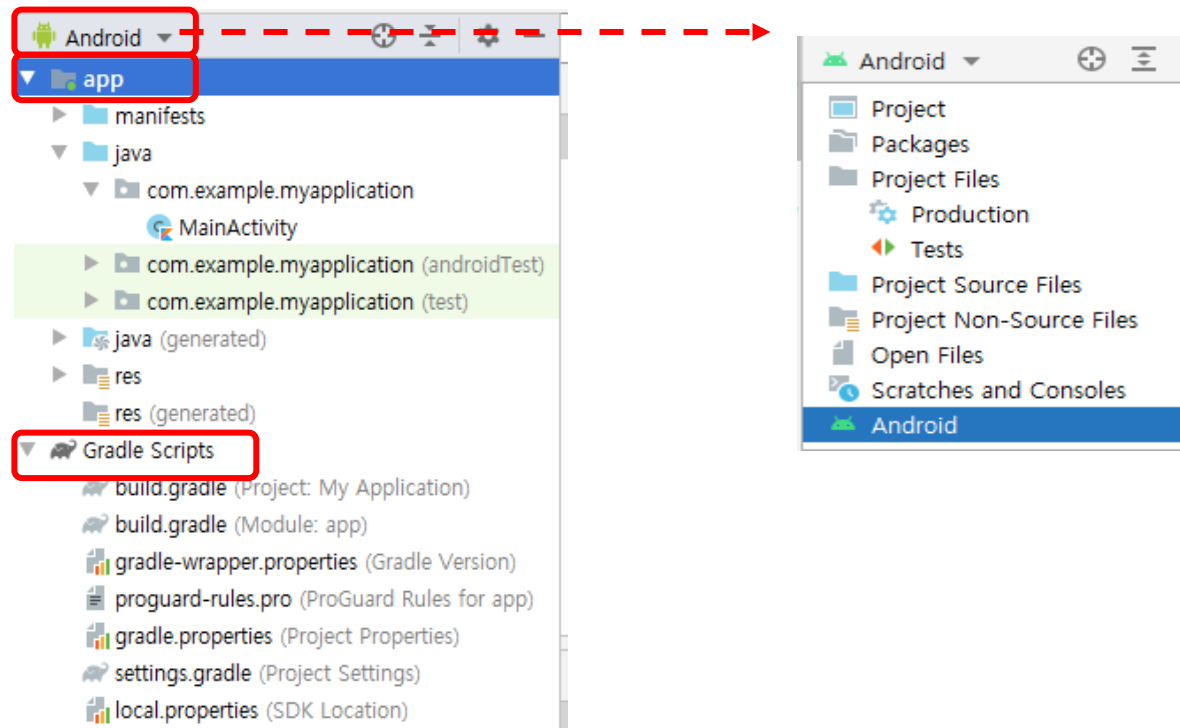


안드로이드 프로젝트 구조

프로젝트의 구성

- Android 표시 방식
 - app : 앱의 구조 및 소스와 관련된 파일이 위치하는 곳
 - Gradle Scripts : 빌드에 필요한 파일들이 위치하는 곳



AndroidManifest.xml

- 안드로이드 시스템이 앱을 실행할 때 필요한 기본적인 정보를 제공하는 역할

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.myapplication">
```

앱의 패키지 이름

<application

android:allowBackup="true"

android:icon="@mipmap/ic_launcher"

android:label="@string/app_name"

android:roundIcon="@mipmap/ic_launcher_round"

android:supportsRtl="true"

android:theme="@style/Theme.MyApplication"

<activity android:name=".MainActivity" android:exported="true">

<intent-filter>

<action android:name="android.intent.action.MAIN" />

<category android:name="android.intent.category.LAUNCHER" />

</intent-filter>

</activity>

</application>

</manifest>

런처 아이콘

런처 라벨
앱 이름

안드로이드 시스템에서 제공하는 백업 기능에 이 앱을 포함할지 여부
True : 앱을 삭제하고 다시 설치할 때 기존 사용하던 정보를 복원
False : 항상 새로 설치되고, 초기화된 상태로 실행

라운드 아이콘

이슬람 문화권의 Right To Left(RTL)을 지원하는지 여부

앱의 테마, 앱의 기본 색상과 모양 결정

첫화면

java+kotlin 폴더

- 안드로이드 앱을 구성하는 소스 파일들이 패키지 명으로 분류되어 있음



res 폴더


- Res는 resource의 약자로, 소스 코드를 제외한 기타 파일들이 위치
 - Drawable
 - 이미지(.png, .jpg, .gif), 나인패치 이미지(.9.png), 또는 XML
 - R.drawable.파일이름 or @drawable/파일이름
 - Layout
 - 액티비티 레이아웃을 정의하는 XML
 - R.layout.파일이름
 - Mipmap
 - 각기 다른 런처 아이콘 해상도에 대한 이미지 파일
 - R.mipmap.파일이름 or @mipmap/파일이름
 - Values
 - 코드와 레이아웃 모두에서 사용할 수 있는 공통 리소스를 정의한 XML
 - colors.xml : 컬러값 정의 → @color/name
 - strings.xml : 문자열 정의 → @string/name `stringResource(R.string.hello_world)`
 - themes.xml : 스타일 정의 → @style/name


Gradle Scripts


- Gradle
 - 복잡한 빌드 과정을 쉽게 적용하기 위한 안드로이드 공식 빌드 툴
 - 자신이 원하는 버전의 APK를 쉽게 생성하고, 여러가지 옵션을 담아서 APK를 만들 수 있음
- Settings.gradle, Build.gradle
 - Gradle이 빌드를 시작하기 위해서 필요한 설정 정보


▼ Gradle Scripts


 build.gradle (Project: My Application)


 build.gradle (Module: app)

 gradle-wrapper.properties (Gradle Version)

 proguard-rules.pro (ProGuard Rules for app)

 gradle.properties (Project Properties)

 settings.gradle (Project Settings)

 local.properties (SDK Location)

Settings.gradle

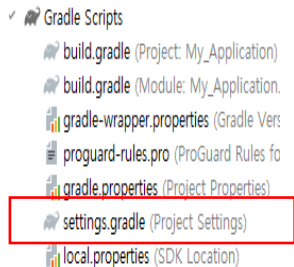
- 저장소 설정 정의 및 앱을 빌드할 때 포함해야 하는 모듈을 Gradle에 알려줌

```
pluginManagement {  
    repositories {  
        gradlePluginPortal()  
        google()  
        mavenCentral()  
    }  
}
```

Gradle이 Gradle 플러그인 및 관련된 dependencies를
탐색하거나 다운로드하기 위한 저장소

```
dependencyResolutionManagement {  
    repositoriesMode.set(RepositoriesMode.FAIL_ON_PROJECT_REPOS)  
    repositories {  
        google()  
        mavenCentral()  
    }  
}  
rootProject.name = "My Application"  
include ':app'
```

프로젝트의 모든 모듈에서 사용할 저장소 및 dependencies



Build.gradle (Project)

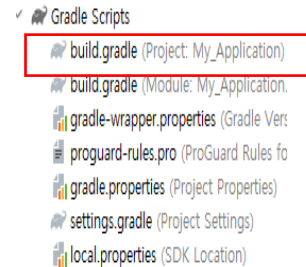
- 프로젝트의 모든 모듈에 적용되는 종속 항목 정의

```
plugins {
```

```
    id("com.android.application") version "8.2.2" apply false
```

```
    id("org.jetbrains.kotlin.android") version "1.9.0" apply false
```

```
}
```



Build.gradle (Module)

- 모듈 빌드 파일로, 앱에 직접 적용될 빌드 구성을 정의

```
android {  
    compileSdkVersion 32  
  
    defaultConfig {  
        applicationId "com.example.myapplication"  
        minSdkVersion 29  
        targetSdkVersion 32  
        versionCode 1  
        versionName "1.0"  
        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"  
    }  
    buildTypes {  
        release {  
            isMinifyEnabled false  
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'  
        }  
    }  
}
```

컴파일에 사용할 API 버전 지정

패키지 네임

최소 지원 API

가장 최적화된 버전 → 가장 안정적으로 동작하도록 테스트

앱의 버전 정보, 업데이트되면 값을 올려서 정보 표시

배포용 apk를 생성할 때 적용하는 옵션

난독화 적용 여부
난독화는 내부 코드를 쉽게 분석할 수 없게 만들어 주는 기능

Build.gradle (Module)

- 모듈 빌드 파일로, 앱에 직접 적용될 빌드 구성을 정의

프로젝트에서 쓰이는 종속성을 정의

dependencies {

```
implementation("androidx.core:core-ktx:1.12.0")  
implementation("androidx.lifecycle:lifecycle-runtime-ktx:2.7.0")  
implementation("androidx.activity:activity-compose:1.8.2")  
implementation(platform("androidx.compose:compose-bom:2023.08.00"))  
implementation("androidx.compose.ui:ui")  
implementation("androidx.compose.ui:ui-graphics")  
implementation("androidx.compose.ui:ui-tooling-preview")  
implementation("androidx.compose.material3:material3")
```

...

```
implementation(libs.androidx.core.ktx)  
implementation(libs.androidx.lifecycle.runtime.ktx)  
implementation(libs.androidx.activity.compose)  
implementation(platform(libs.androidx.compose.bom))  
implementation(libs.androidx.ui)  
implementation(libs.androidx.ui.graphics)  
implementation(libs.androidx.ui.tooling.preview)  
implementation(libs.androidx.material3)  
implementation(libs.androidx.compose.material)
```

안드로이드 Jetpack Compose 소개

<https://developer.android.com/jetpack/compose>

Android Jetpack Compose

- Jetpack Compose란?

- 안드로이드 UI를 빌드하기 위한 새로운 선언적 UI 프레임워크
- 기존의 XML 기반 레이아웃 구성 대신 Kotlin 언어를 사용하여 UI 설계
- UI 설계를 위해 안드로이드에서 권장하는 최신도구
- <https://developer.android.com/jetpack/compose>

- 장점 및 접근방식

- 간결한 코드
 - 간결하고 가독성 높은 코드를 제공하고, 복잡한 UI도 적은 코드로 구현 가능
- 실시간 미리보기
 - UI 코드를 변경할 때마다 실시간으로 변경 내용을 미리 볼 수 있음
- 상태 관리의 용이성
 - 상태를 보다 쉽게 관리하고 업데이트 할 수 있어 유지보수가 용이함

기존 UI 개발 방식 vs. Compose

- 기존 UI 개발 방식 (명령형프로그래밍)
 - XML 파일을 통한 UI 레이아웃 정의
 - UI와 동작 로직이 분리되어 있음
 - 가독성이 낮고 복잡한 구조로 유지보수가 어려움
- Compose (선언형프로그래밍)
 - Ui 구조를 Top-level 및 @Composable 함수로 구성
 - 코드의 가독성이 증가, 높은 생산성을 제공

```
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical">

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello, XML Layout" />

  <Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Click me"
    android:onClick="handleButtonClick" />
</LinearLayout>
```

```
@Composable
fun ComposeExample() {
  Column {
    Text("Hello, Jetpack Compose!")
    Button(onClick = { /* Handle button click */ }) {
      Text("Click me")
    }
  }
}
```

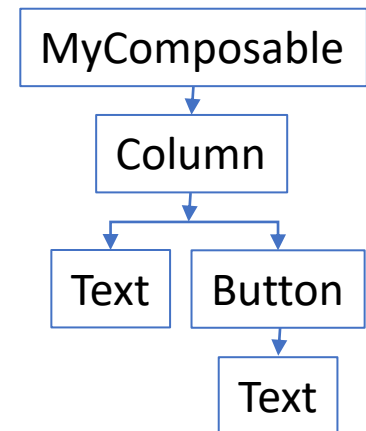
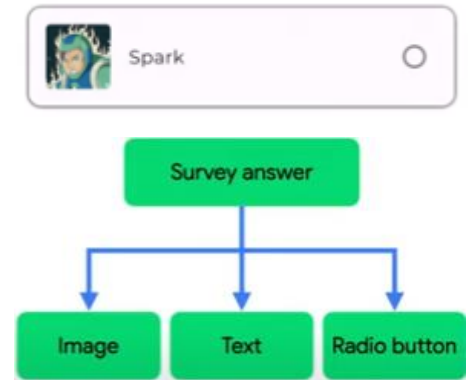
Jetpack Compose의 기본 구조

- @Composable 함수
 - UI를 생성하는 함수
 - @Composable 어노테이션 사용
- Top-level 구조를 활용하여 구성
 - 여러 컴포저블 함수들이 서로 상호작용하여 UI를 구성

```
@Composable
fun MyApp() {
    MyComposable()
    AnotherComposable()
}
```

- 각 함수내에 컴포저블 함수 및 레이아웃 구성요소 배치

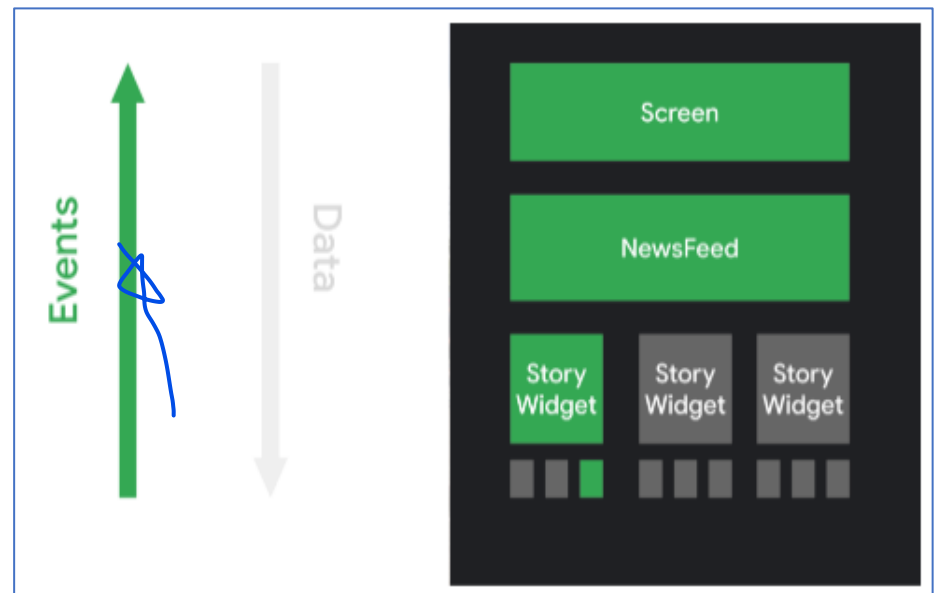
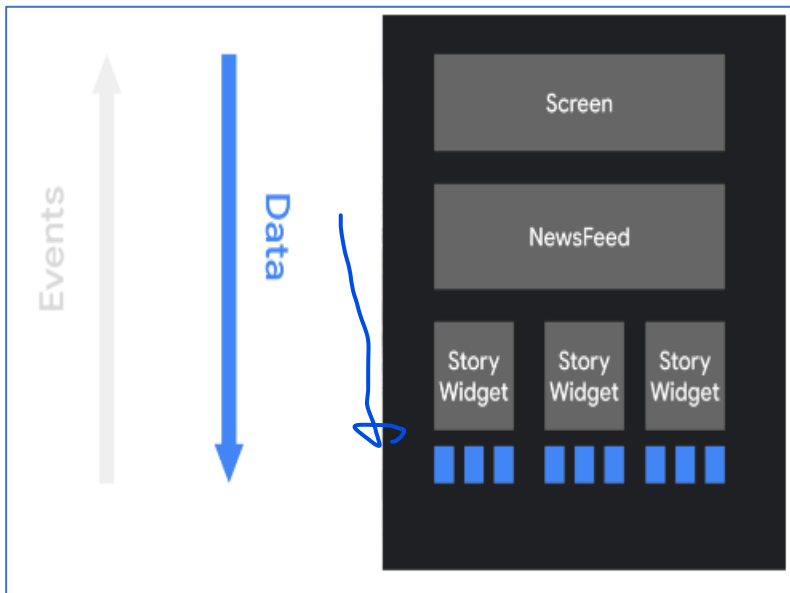
```
@Composable
fun MyComposable() {
    Column {
        Text("Hello")
        Button(onClick = { /* Handle button click */ }) {
            Text("Click me")
        }
    }
}
```



Compose의 특징

• 선언형 UI 프레임워크

- 함수의 매개변수로 데이터를 전달하여 UI 업데이트
- 이벤트가 발생하여, 앱의 상태가 변경되면 UI 요소가 새로운 데이터를 이용하여 다시 그려짐 (**Recomposition**)



Compose의 특징

- 간단한 컴포저블 함수의 집합으로 사용자 인터페이스 빌드



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

- 동적 콘텐츠

- 함수이므로 여러 번 호출이 가능하고, 제어구문을 통해 동적 콘텐츠를 쉽게 구성할 수 있음

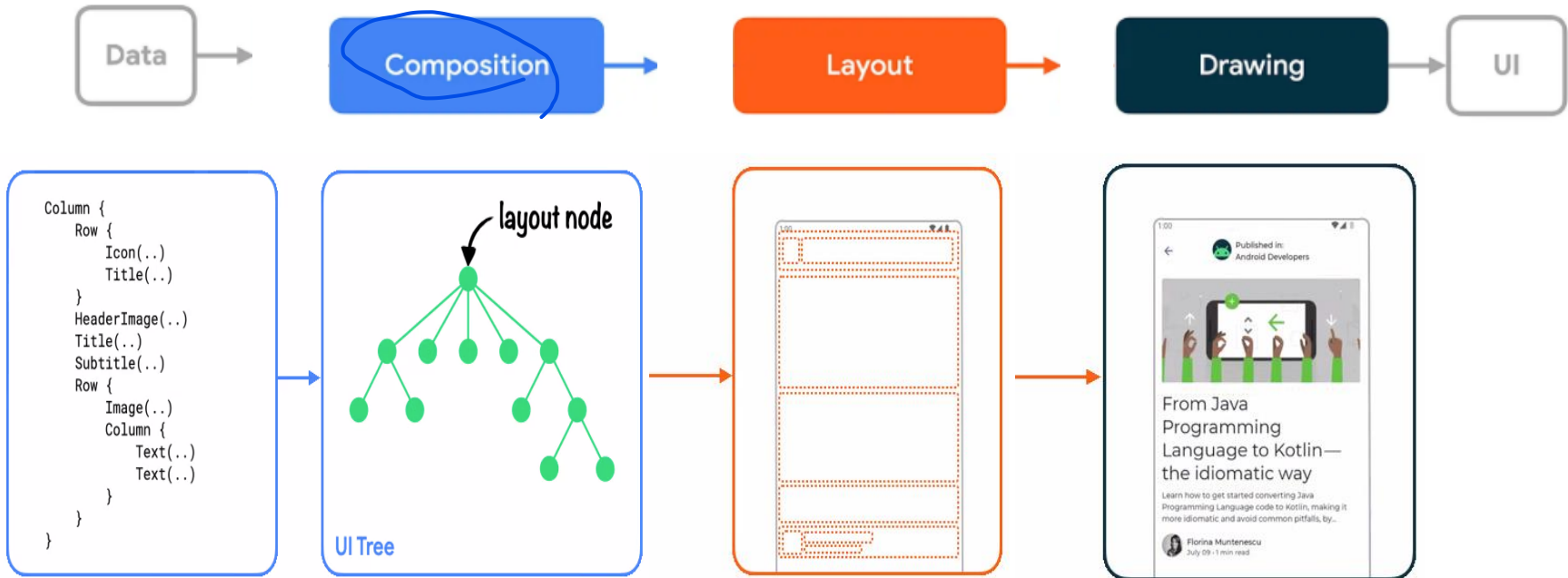
```
@Composable
fun Greeting(names: List<String>) {
    for (name in names) {
        Text("Hello $name")
    }
}
```


Composable 함수

- @Composable 어노테이션이 있어야 함
 - Compose 컴파일러에게 UI를 만드는 함수라고 알림
- 매개변수를 통해 데이터를 수신
- 반환하는 값이 없음
 - 화면 상태를 설명하므로 아무것도 반환할 필요 없음
- 다른 Composable 함수를 호출해서 UI 계층 구조를 생성
- 빠르고, side-effect가 없음
 - 여러 번 호출되어도 동일한 방식으로 작동
 - 전역변수를 사용하지 않음

UI 생성 과정

- Composable로 부터 UI가 만들어지는 과정



- Recomposition

- 상태가 변경되면 UI 요소가 새로운 데이터를 이용하여 다시 그려지는 과정

Compose의 구성요소

- **Foundation 컴포넌트**

- 기본적인 사용자 인터페이스 기능을 제공하는 컴포넌트
 - Button, Text, Image, BaseTextField, LazyColumn, LazyRow . . .

- **Layout 컴포넌트**

- 컴포넌트를 화면에 배치하고, 배치된 컴포넌트들이 상호 동작하는 방법을 정의하는 컴포넌트
 - Column, Row, Box, ConstraintLayout . . .

- **Material design 컴포넌트**

- 구글이 제공하는 머티리얼 테마 가이드라인을 만족하도록 디자인된 컴포넌트
 - AlertDialog, Button, Card, CheckBox, TextField, RadioButton . . .

* Composable Function을 줄여서 “Composable (컴포저블)” 이라 부름

용어정리

- 컴포즈 (Compose)
 - Jetpack Compose로 안드로이드 UI를 선언적으로 작성하기 위한 프레임워크
 - UI를 구축하고 관리하는데 사용되는 API 및 라이브러리 모임
- 컴포저블 (Composable)
 - 컴포즈에서 UI를 만들기 위해 사용되는 함수
 - 각 컴포저블 함수는 특정 UI 요소를 정의하고, 컴포저블을 조합하여 전체 UI를 구성함
- 컴포지션 (Composition)
 - 여러 컴포저블 함수들을 조합하여 만든 전체 UI 구조를 의미하는 용어
 - 컴포저블 함수를 조합하여 UI를 선언적으로 정의하는 프로세스를 의미
- 리컴포지션 (Recomposition)
 - 컴포즈가 컴포저블의 상태가 변경될 때 자동으로 UI를 갱신하는 프로세스

기본 Composable UI 컴포넌트

Contents

- 기본 컴포넌트
 - Layout 컴포저블
 - Column / Row / Box
 - Text, TextField, Switch
- Modifier
- Recomposition
 - 상태 / 상태 호이스팅

Custom Tip 계산기 만들기

- Compose 사용시 알아야 하는 Android 기본 사항
 - <https://developer.android.com/courses/android-basics-compose/course>
 - **Unit2 : Interacting with UI and state**
 - <https://developer.android.com/codelabs/basic-android-compose-calculate-tip#0>
- 참고 기능 (Live Templates)
 - Android Studio에서 생산성을 높이기 위해 제공되는 기능
 - Comp : 컴포저블 함수를 만들어 주는 기능
 - WC : Column 구문 만들어 주는 기능
 - WR : ROW 구문 만들어 주는 기능
 - Prev : Preview 어노테이션 완성 기능

Calculate Tip

Bill Amount

Tip Percentage

Round up tip? ☐

Tip Amount: \$0.00

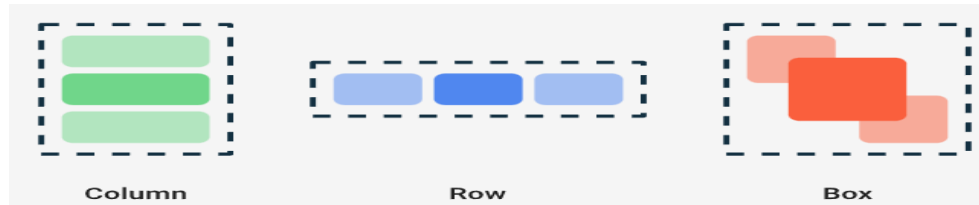
Column/Row/Box 컴포저블

- Layout 컴포저블로 UI 컴포넌트들을 배치하는 기능 수행
 - 배치 방식이 지정되지 않으며, 원치 않는 형태로 화면 구성됨

```
@Composable
fun ArtistCard() {
    Text("Alfred Sisley")
    Text("3 minutes ago")
}
```



- Standard Layout Components
 - Column / Row / Box



```
@Composable
fun ArtistCardColumn() {
    Column {
        Text("Alfred Sisley")
        Text("3 minutes ago")
    }
}
```

```
@Composable
fun ArtistCardRow(artist: Artist) {
    Row(verticalAlignment = Alignment.CenterVertically) {
        Image(bitmap = artist.image, contentDescription = "Artist image")
        Column {
            Text(artist.name)
            Text(artist.lastSeenOnline)
        }
    }
}
```

```
@Composable
fun ArtistAvatar(artist: Artist) {
    Box {
        Image(bitmap = artist.image, contentDescription = "Artist image")
        Icon(Icons.Filled.Check, contentDescription = "Check mark")
    }
}
```

Alfred Sisley
3 minutes ago



Alfred Sisley
3 minutes ago



Column/Row/Box Layout

- Column

```
@Composable
inline fun Column(
    modifier: Modifier = Modifier,
    verticalArrangement: Arrangement.Vertical = Arrangement.Top,
    horizontalAlignment: Alignment.Horizontal = Alignment.Start,
    content: @Composable ColumnScope.() -> Unit
): Unit
```

```
@Composable
fun ArtistCardColumn() {
    Column {
        Text("Alfred Sisley")
        Text("3 minutes ago")
    }
}
```

- Row

```
@Composable
inline fun Row(
    modifier: Modifier = Modifier,
    horizontalArrangement: Arrangement.Horizontal = Arrangement.Start,
    verticalAlignment: Alignment.Vertical = Alignment.Top,
    content: @Composable RowScope.() -> Unit
): Unit
```

- Box

```
@Composable
inline fun Box(
    modifier: Modifier = Modifier,
    contentAlignment: Alignment = Alignment.TopStart,
    propagateMinConstraints: Boolean = false,
    content: @Composable BoxScope.() -> Unit
): Unit
```

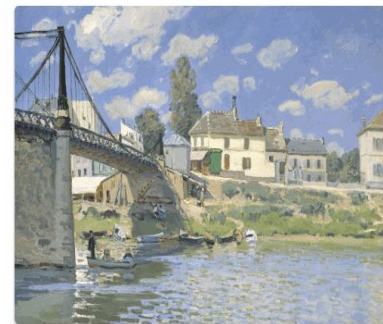
Modifier

- 컴포저블에 다양한 기능 제공하는 컴포즈 내장 객체
 - 컴포저블의 크기, 레이아웃, 동작, 모양 변경
 - 클릭, 스크롤, 드래그, 확대/축소 등 높은 수준의 상호작용 추가
- **Modifier의 순서가 중요**
 - Modifier의 함수를 호출하는 순서가 최종 결과에 영향을 줌

```
@Composable
fun ArtistCard(/*...*/) {
    val padding = 16.dp
    Column(
        Modifier
            .clickable(onClick = onClick)
            .padding(padding)
            .fillMaxWidth()
    ) {
        // rest of the implementation
    }
}
```



```
@Composable
fun ArtistCard(/*...*/) {
    val padding = 16.dp
    Column(
        Modifier
            .padding(padding)
            .clickable(onClick = onClick)
            .fillMaxWidth()
    ) {
        // rest of the implementation
    }
}
```



Modifier

- Modifier 만들기

```
val modifier = Modifier
```

```
val modifier = Modifier.padding(all=10.dp)
```

```
val modifier = Modifier
```

```
.padding(all=10.dp)
```

```
.border(width=2.dp, color=Color.Black)
```

```
Text("Greenjoa",
```

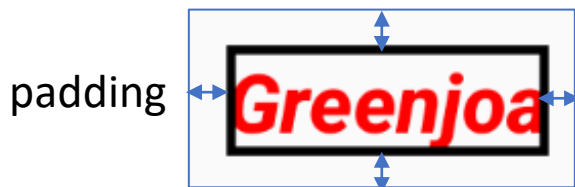
```
    modifier = modifier,
```

```
    fontSize = 20.sp,
```

```
    fontStyle = FontStyle.Italic,
```

```
    fontWeight = FontWeight.ExtraBold,
```

```
    color = Color.Red)
```



*아래에서 위 순서로 적용



- 컴포넌트의 크기는 dp 사용

- Density-independent Pixel
- 1인치에 들어가는 픽셀을 나타냄
- 화면의 해상도가 달라도 동일한 비율로 보여주기 위한 단위

```
val modifier = Modifier
```

```
.border(width=2.dp, color=Color.Black)
```

```
.padding(all=10.dp)
```

Modifier

- 컴포저블은 Modifier를 매개변수로 받아서 사용
 - Modifier 매개변수는 선택적이어야 하므로, 디폴트 Modifier 인스턴스 지정

```
@Composable
fun TextExample(modifier: Modifier=Modifier){
    Text("Greenjoa",
        modifier = modifier,
        fontSize = 20.sp,
        fontStyle = FontStyle.Italic,
        fontWeight = FontWeight.ExtraBold,
        color = Color.Red)
}
```

```
val modifier = Modifier
    .border(width=2.dp, color=Color.Black)
    .padding(all=10.dp)
```

```
TextExample(modifier)
```

```
@Composable
fun Greeting(nameList:List<String>,
             modifier: Modifier = Modifier)
{
    Column(modifier=Modifier.padding(all = 10.dp)) {
        for (name in nameList) {
            Text(
                text = "Hello $name!",
                modifier = modifier
            )
        }
    }
}
```

Modifier

- Built-in Modifier 메소드

- Background : 배경 색상 지정
- Clickable : 클릭했을 때 호출되는 핸들러 지정
- Clip : 콘텐츠를 지정한 크기로 자름
- **FillMaxHeight** : Composable의 높이를 부모가 허용하는 최대값에 맞춤
- **FillMaxSize** : Composable의 높이와 폭을 부모가 허용하는 최대값에 맞춤
- **FillMaxWidth** : Composable의 폭을 부모가 허용하는 최대값에 맞춤
- Layout : 커스텀 레이아웃을 구현
- Offset : 현재 위치에서 x, y축 방향으로 지정한 거리만큼 이동
- **Padding** : 주변에 공백을 추가
- Rotate : Composable의 중심점을 기준으로 지정한 숫자(각도)만큼 회전
- Scale : 지정한 비율만큼 컴포저블의 크기를 확대 및 축소
- Scrollable : 스크롤 기능 활성화
- **Size** : 높이와 폭을 지정할 때 이용. 크기를 지정하지 않으면 콘텐츠에 맞춰 Composable의 크기가 결정(wrapping)됨

Modifier

- Modifier 조합

- 동일한 컴포지블에 둘 이상의 Modifier 객체 이용하는 경우 “then” 으로 조합

val combinedModifier =

firstModifier.**then**(secondModifier).**then**(thridModifier)

@Composable

```
fun TextExample(modifier: Modifier=Modifier){  
    val rotateModifier = Modifier.rotate(10.0f)
```

```
    Text("Greenjoa",  
        modifier = modifier.then(rotateModifier),  
        fontSize = 20.sp,  
        fontStyle = FontStyle.Italic,  
        fontWeight = FontWeight.ExtraBold,  
        color = Color.Red)  
}
```

Column/Row/Box 실습

- Custom Composable 함수 만들기
 - **@Composable 어노테이션 추가하여 생성**
 - Composable 함수는 반환하는 데이터가 없음
 - Composable 함수에서는 Composable 함수와 표준함수 호출 가능
 - 표준함수에서는 Composable 함수 호출 불가
 - 함수이름은 대문자로 시작

Text

[androidx.compose.material3.Text](#)

• 텍스트를 출력하는 컴포넌트

- Material Design guidelines에 따라 제공되는 Composable 함수
 - 기본 요소로 BasicText Composable 함수도 제공되고 있지만, Text 사용 권고

```
@Composable
fun Text(
    text: String,
    modifier: Modifier = Modifier,
    color: Color = Color.Unspecified,
    fontSize: TextUnit = TextUnit.Unspecified,
    fontStyle: FontStyle? = null,
    fontWeight: FontWeight? = null,
    fontFamily: FontFamily? = null,
    letterSpacing: TextUnit = TextUnit.Unspecified,
    textDecoration: TextDecoration? = null,
    textAlign: TextAlign? = null,
    lineHeight: TextUnit = TextUnit.Unspecified,
    overflow: TextOverflow = TextOverflow.Clip,
    softWrap: Boolean = true,
    maxLines: Int = Int.MAX_VALUE,
    minLines: Int = 1,
    onTextLayout: ((TextLayoutResult) -> Unit)? = null,
    style: TextStyle = LocalTextStyle.current
): Unit
```

Parameters

text: String	The text to be displayed.
modifier: Modifier = Modifier	Modifier to apply to this layout node.
color: Color = Color.Unspecified	Color to apply to the text. If Color.Unspecified, and style has no color set, this will be LocalContentColor.
fontSize: TextUnit = TextUnit.Unspecified	The size of glyphs to use when painting the text. See TextStyle.fontSize.
fontStyle: FontStyle? = null	The typeface variant to use when drawing the letters (e.g., italic). See TextStyle.fontStyle.
fontWeight: FontWeight? = null	The typeface thickness to use when painting the text (e.g., FontWeight.Bold).
fontFamily: FontFamily? = null	The font family to be used when rendering the text. See TextStyle.fontFamily.
letterSpacing: TextUnit = TextUnit.Unspecified	The amount of space to add between each letter. See TextStyle.letterSpacing.
textDecoration: TextDecoration? = null	The decorations to paint on the text (e.g., an underline). See TextStyle.textDecoration.
textAlign: TextAlign? = null	The alignment of the text within the lines of the paragraph. See

Text

@Composable

```
fun TextExample(){  
    Text("Greenjoa",  
        fontSize = 20.sp,  
        fontStyle = FontStyle.Italic,  
        fontWeight = FontWeight.ExtraBold,  
        color = Color.Red)  
}
```



Greenjoa

*String 리소스에서 불러오기

```
Text(text=stringResource(id = R.string.greenjoa))
```

- 텍스트의 크기 **sp(scaled independent pixel)** 단위
 - 글꼴 크기에 따라 텍스트 크기 지정됨
 - Int에서 확장함수로 제공됨
 - 20.sp

문자열 리소스 추가

- 문자열 리소스

- res > values > strings.xml

```
<resources>
    <string name="app_name">Tip Time</string>
    <string name="calculate_tip">Calculate Tip</string>
    <string name="bill_amount">Bill Amount</string>
    <string name="how_was_the_service">Tip Percentage</string>
    <string name="round_up_tip">Round up tip?</string>
    <string name="tip_amount">Tip Amount: %s</string>
</resources>
```

- strings.xml 파일 상단의 editor 이용

Edit translations for all locales in the translations editor. [Open editor](#)

Add Key

×

Key:


Default Value:

Resource Folder:

OK

Cancel

Calculate Tip

 Bill Amount

% Tip Percentage

Round up tip?



Tip Amount: \$0.00

TextField

[androidx.compose.material3.TextField](#)

- 사용자의 텍스트 입력을 위한 머티리얼 디자인이 적용된 컴포넌트

```
@Composable
fun TextField(
    value: String,
    onChange: (String) -> Unit,
    modifier: Modifier = Modifier,
    enabled: Boolean = true,
    readOnly: Boolean = false,
    textStyle: TextStyle = LocalTextStyle.current,
    label: (@Composable () -> Unit)? = null,
    placeholder: (@Composable () -> Unit)? = null,
    leadingIcon: (@Composable () -> Unit)? = null,
    trailingIcon: (@Composable () -> Unit)? = null,
    isError: Boolean = false,
    visualTransformation: VisualTransformation = VisualTransformation.None,
    keyboardOptions: KeyboardOptions = KeyboardOptions.Default,
    keyboardActions: KeyboardActions = KeyboardActions(),
    singleLine: Boolean = false,
    maxLines: Int = if (singleLine) 1 else Int.MAX_VALUE,
    minLines: Int = 1,
    interactionSource: MutableInteractionSource? = null,
    shape: Shape = MaterialTheme.shapes.small.copy(bottomEnd = ZeroCornerSize, bottomStart = ZeroCornerSize),
    colors: TextFieldColors = TextFieldDefaults.textFieldColors()
): Unit
```

TextField

- TextField / OutlinedTextField

```
@Composable
fun SimpleFilledTextFieldSample() {
    var text by remember { mutableStateOf("Hello") }

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

Label
Hello

```
@Composable
fun SimpleOutlinedTextFieldSample() {
    var text by remember { mutableStateOf("") }

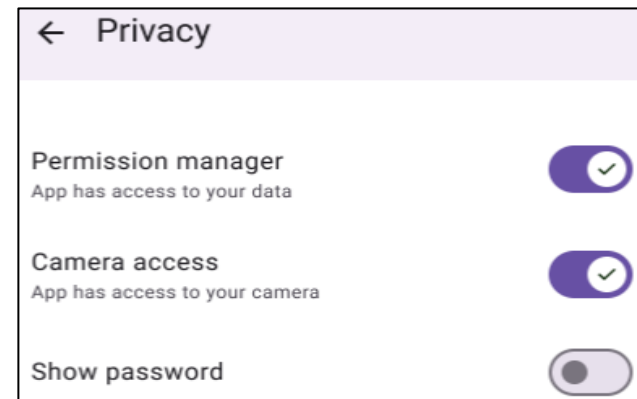
    OutlinedTextField(
        value = text,
        onValueChange = { text = it },
        label = { Text("Label") }
    )
}
```

Label
Hello Compose

Switch

[androidx.compose.material3.Switch](#)


- On/Off 선택을 처리하는 컴포넌트
 - 주로 설정할 때 사용




```
@Composable
fun Switch(
    checked: Boolean,
    onCheckedChange: ((Boolean) -> Unit)?,
    modifier: Modifier = Modifier,
    thumbContent: (@Composable () -> Unit)? = null,
    enabled: Boolean = true,
    colors: SwitchColors = SwitchDefaults.colors(),
    interactionSource: MutableInteractionSource? = null
): Unit
```

예제. 화면 디자인

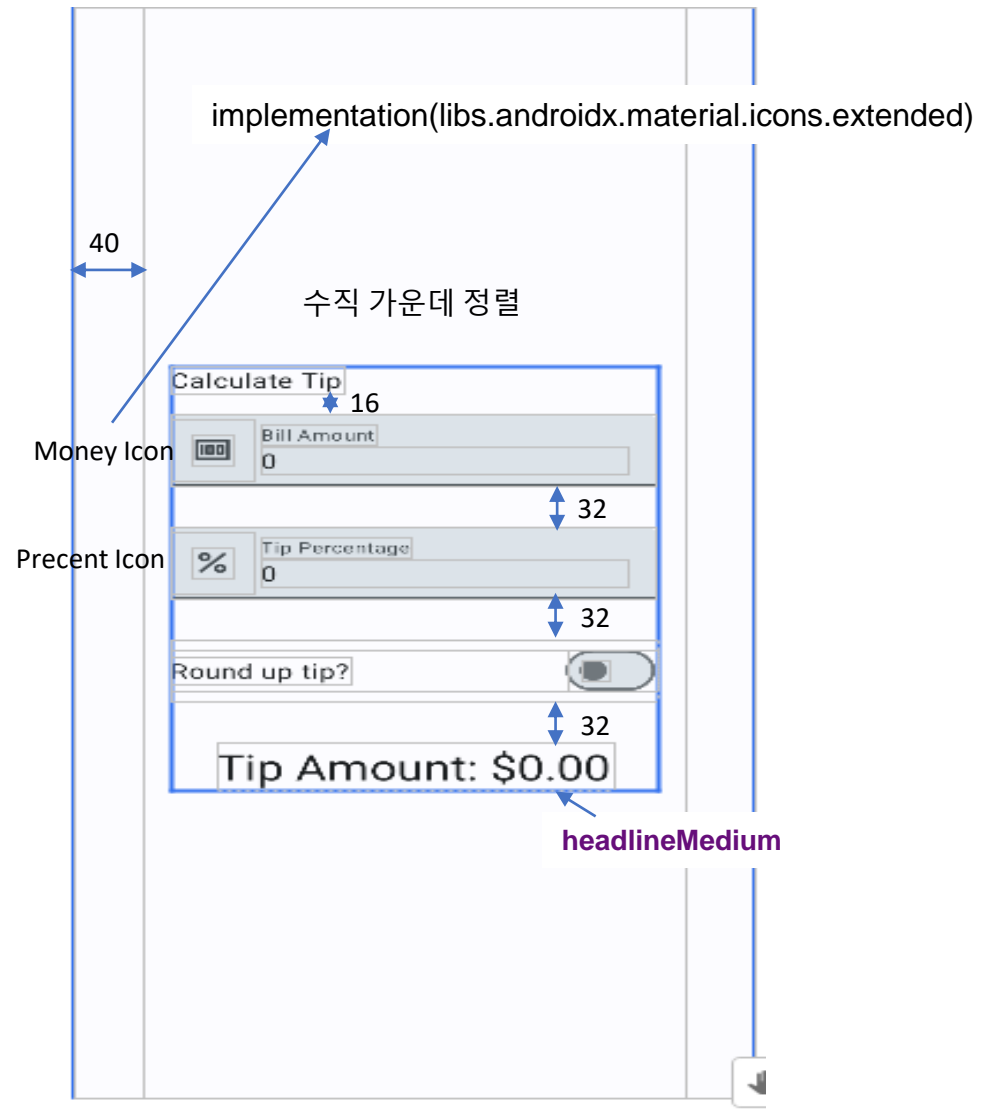
Calculate Tip

 Bill Amount

 Tip Percentage

Round up tip? ☐

Tip Amount: \$0.00



실습 1. 화면 디자인 변경하기

- 기본 화면 구성요소 변경하기
 - Text
 - **TextField (키보드 옵션 변경)**
 - Bill Amount : ImeAction Next
 - Tip Percentage : ImeAction Done / hide
 - Switch

* 배포한 스타일 참고 할 것

Calculate Tip



Bill Amount



Tip Percentage

Round up tip?



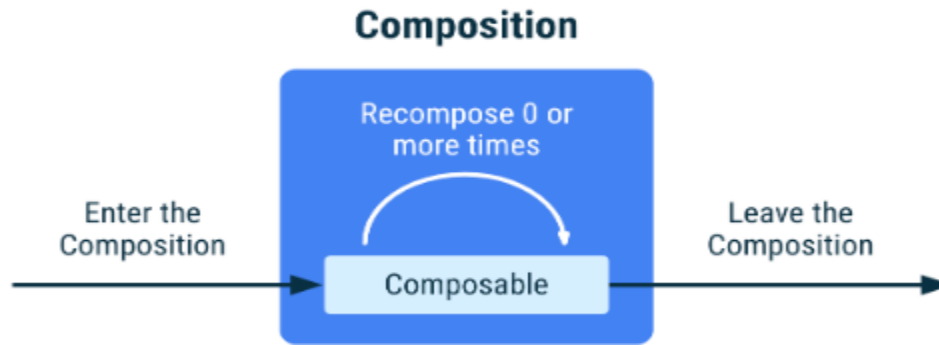
Tip Amount: \$0.00

상태(State)

- 상태(State)
 - 시간에 따라 변경될 수 있는 값
 - 이벤트에 대한 응답으로 상태가 업데이트 됨
 - 기존 Kotlin 표준 변수와의 차이점
 - 컴포저블에서 상태 변수에 할당된 값은 기억되어야 함
 - 상태 변수의 변경이 컴포지션의 계층 트리 전체에 영향을 주게 됨
 - Compose에서는 컴포저블(UI 컴포넌트)에 대한 변수를 별도로 생성하지 않으므로 컴포저블의 상태 관리가 중요

리컴포지션(Recomposition)

- 컴포저블의 생명주기
 - 컴포지션은 컴포저블을 통해서 생성되고, 리컴포지션을 통해서만 업데이트됨
 - 리컴포지션은 일반적으로 **상태(State<T>) 객체가 변경**되면 트리거 됨



리컴포지션(Recomposition)

- 상태가 변경될 때 컴포지션을 업데이트하기 위해 컴포저블을 재 실행하는 것
 - Composable 함수는 다른 함수들을 호출하면 계층적 구조를 생성
 - 부모 함수에서 선언된 상태 변수들은 모든 자식 Composable에 반영되어 해당 상태가 전달 됨
- 컴포저블 함수의 계층 안에서 상태값이 변경될 때 발생
 - 컴포즈의 상태 변화를 감지하면 해당 상태값의 변화에 영향을 받는 모든 함수를 재구성함
 - 즉, **해당 함수들을 다시 호출하고, 새로운 상태값을 전달하는 함**

상태(State) 객체 생성

- 상태(State)

- `State<T>` : 변경할 수 없는 객체
- `MutableState<T>` : 변경가능한 객체

```
interface State<out T> {  
    val value: T  
}
```

```
interface MutableState<T> : State<T> {  
    override var value: T  
}
```

- `mutableStateOf(value)` 함수로 생성

- `value(default)`값으로 초기화된 `MutableState` 객체 반환
- 상태가 변해 리컴포지션되면 다시 초기화 됨

`var name = mutableStateOf("")` // 오류표시

→ 상태를 유지하는 remember와 같이 사용됨

상태(State) 저장하기

- **remember 함수**

- Recomposition 할 때, 다시 생성 되지 않도록 보장하는 기능
- mutableStateOf 와 함께 사용

```
val mutableState = remember { mutableStateOf(default) }  
var value by remember { mutableStateOf(default) }  
val (value, setValue) = remember { mutableStateOf(default) }
```

- **rememberSaveable 함수**

- Activity의 재생성, 다크/라이트모드, 회전 등 변경될 때도 상태 유지

```
var text1 : MutableState<String> = rememberSaveable { mutableStateOf( value: "" ) }  
var text2 : String by rememberSaveable { mutableStateOf( value: "" ) }  
val (text3 : String , setText3 : (String) -> Unit ) = rememberSaveable { mutableStateOf( value: "" ) }
```

예) TextField의 상태값 접근

- 사용형태 비교

```
@Composable
fun MyTextField(){
    var textState = remember { mutableStateOf("") }
    val onTextChange = {text:String ->
        textState.value = text
    }
    TextField(
        value = textState.value,
        onValueChange = onTextChange
    )
}
```

```
@Composable
fun MyTextField(){
    var textState by remember { mutableStateOf("") }
    val onTextChange = {text:String ->
        textState = text
    }
    TextField(
        value = textState,
        onValueChange = onTextChange
    )
}
```

* `getValue()` 와 `setValue()` 함수 import 해야 함

예) TextField의 상태값 접근

- 사용형태 비교

`@Composable`

`fun` MyTextField(){

`var (textValue, setText) = remember { mutableStateOf("") }`

`val` onTextChange = {text:String ->

`setText`(text)

}

`TextField`(

value = `textValue`,

onValueChange = onTextChange

)

}

예제. 상태 추가하기

- 팁 계산함수

```
import android.icu.text.NumberFormat
```

```
fun calculateTip(amount: Double,  
                tipPercent: Double,  
                roundUp: Boolean): Any {
```

```
    var tip = tipPercent / 100 * amount
```

```
    if (roundUp) {
```

```
        tip = kotlin.math.ceil(tip)
```

```
    }
```

```
    return NumberFormat.getCurrencyInstance().format(tip)
```

```
}
```

Calculate Tip



Bill Amount



Tip Percentage

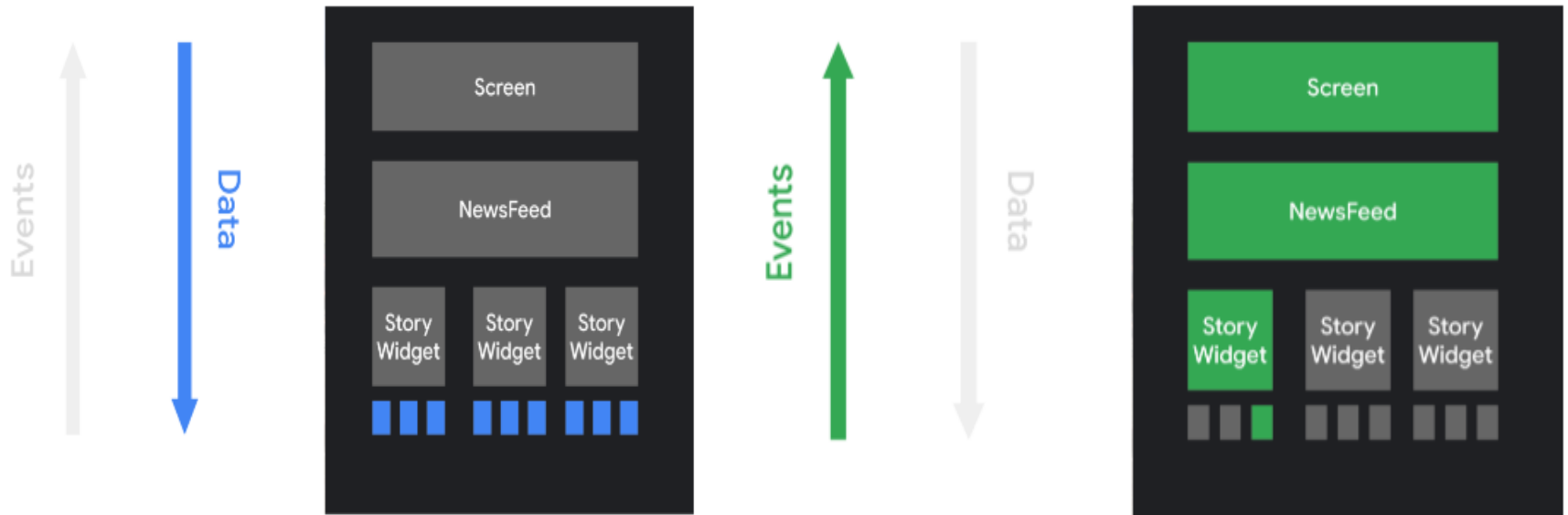
Round up tip?



Tip Amount: \$0.00

단방향 데이터 흐름

- 한 컴포저블에서 저장된 상태는 자식 컴포저블 함수들에서 직접 변경되어서는 안됨
- State(데이터는)는 아래로 흐르고, 이벤트는 위로 향하도록 설계



상태 호이스팅 (State Hoisting)

- 컴포저블을 Sateless로 만들기 위해 상태를 컴포저블의 호출자로 옮기는 패턴
 - Stateful
 - 상태를 소유하는 컴포저블
 - 상태를 제어할 필요 없고, 상태를 직접 관리하지 않는 경우 유용
 - 컴포저블의 재사용성이 떨어지고, 테스트가 어려운 경향이 있음
 - Sateless
 - 상태를 소유하지 않는 컴포저블
- 상태 변수와 이벤트 핸들러의 위치를 호출자로 바꾸는 것
 - `value: T`: 표시할 현재 값
 - `onValueChange: (T) -> Unit`: `T`가 제안된 새 값인 경우 값을 변경하도록 요청하는 이벤트

상태 호이스팅 (State Hoisting)

```
@Composable
fun MyTextField(){
    var textState : String by remember { mutableStateOf( value: "" ) }
    val onTextChange : (String) -> Unit = {text:String ->
        | textState = text
    }
    TextField(
        value = textState,
        onValueChange = onTextChange
    )
}
```

```
@Composable
fun MyTextField(textState:String, onTextChange:(String)->Unit){
    TextField(
        | value = textState,
        onValueChange = onTextChange
    )
}
```

```
@Composable
fun MyTextFieldDemo(){
    var textState : String by remember { mutableStateOf( value: "" ) }
    val onTextChange : (String) -> Unit = {text:String ->
        | textState = text
    }
    MyTextField(textState, onTextChange)
}
```


예제. 상태호이스팅 추가하기


- Stateless 컴포저블 만들기

EditNumberField

RoundTheTipRow

Calculate Tip

 Bill Amount

 Tip Percentage

Round up tip? ☐

Tip Amount: \$0.00

과제 2. BMI 계산하기 (10점)

- 몸무게와 키 정보를 입력하면 BMI를 계산하고, BMI 정도에 따라, 텍스트 출력 (스위치 버튼에 따라 키 입력단위 결정 m or cm)
 - BMI : 몸무게 / (키)²
 - 18.5 미만 : 저체중
 - 25 미만 : 정상
 - 30 미만 : 과체중
 - 그 이상 : 비만
- 제출물
 - 보고서(실습보고서)
 - 프로젝트 압축파일
 - 동영상 (앱 실행)

키 입력 단위 미터(m) ? ☐

키(cm)

몸무게(kg)

BMI 체크

키 입력 단위 미터(m) ? ☒

키(m)

1.6

몸무게(kg)

50

정상

키 입력 단위 미터(m) ? ☐

키(cm)

160

몸무게(kg)

50

정상

수고하셨습니다.