1w.

User Interface
&
Activity

Ivy

Layout & View

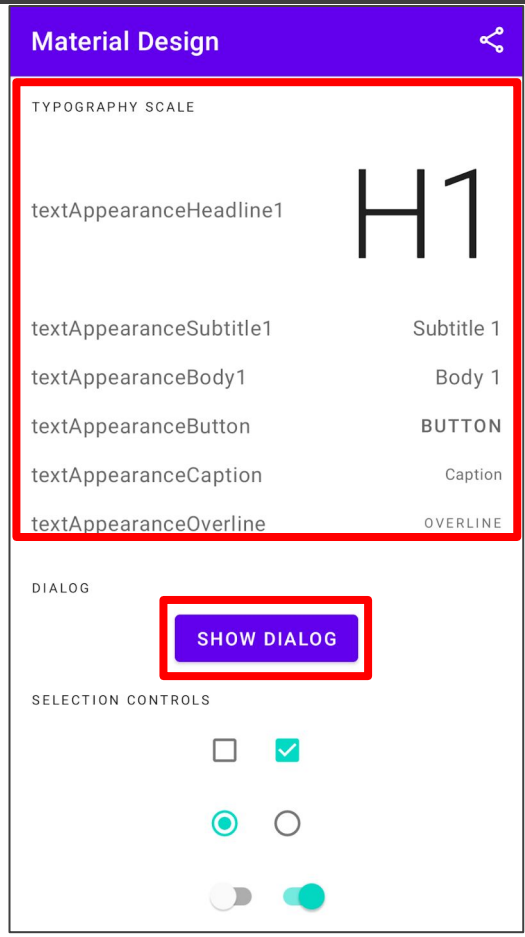
View : 기기 화면에 보이는 모든 것

- Widget : 사용자가 화면을 보면서 상호작용하는 View

	속성
TextView	android:layout_width
	android:layout_height
Button	android:text

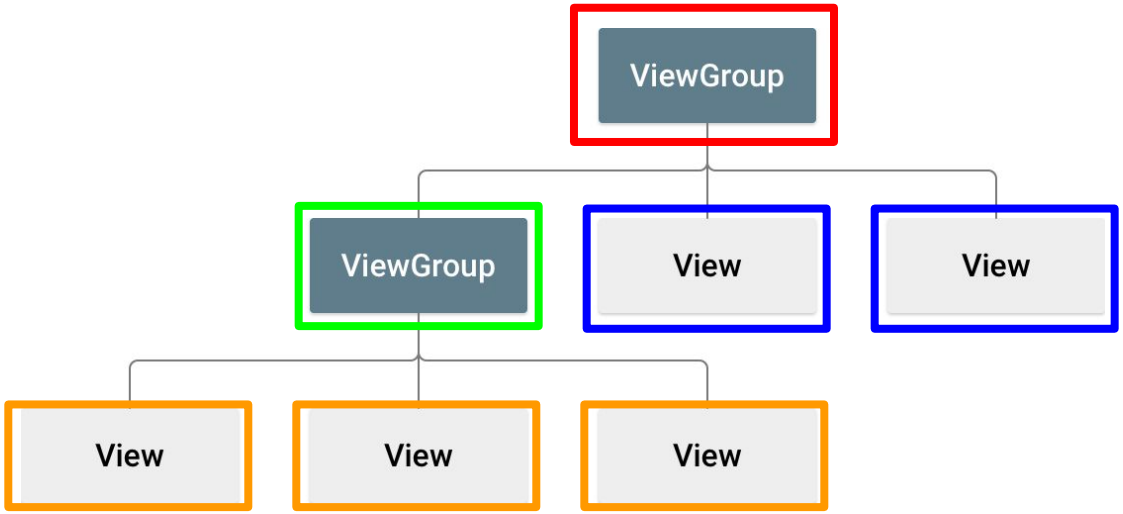
- 위와 같은 View객체를 일반적으로 Widget 이라고 합니다.

  TextView, Button과 같이 View의 subclass 중 하나일 수 있습니다.

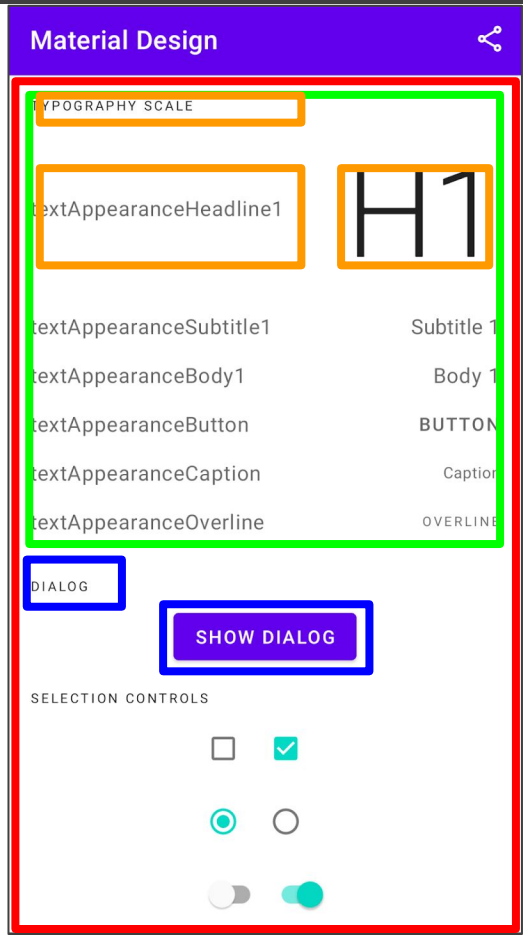


Layout

- UI 객체와, 화면에서의 위치와 같은 구조를 정의합니다.
- 다른 View 혹은 layout을 포함하는 Container 입니다.
- ViewGroup 객체를 일반적으로 'layout' 이라고 합니다.



UI layout을 정의하는 뷰 계층 구조



ConstraintLayout

- 가로, 세로의 제약조건을 추가해 View의 위치를 지정할 수 있습니다.

- 복잡한 UI에서도 중첩된 레이아웃을 최소화할 수 있습니다.

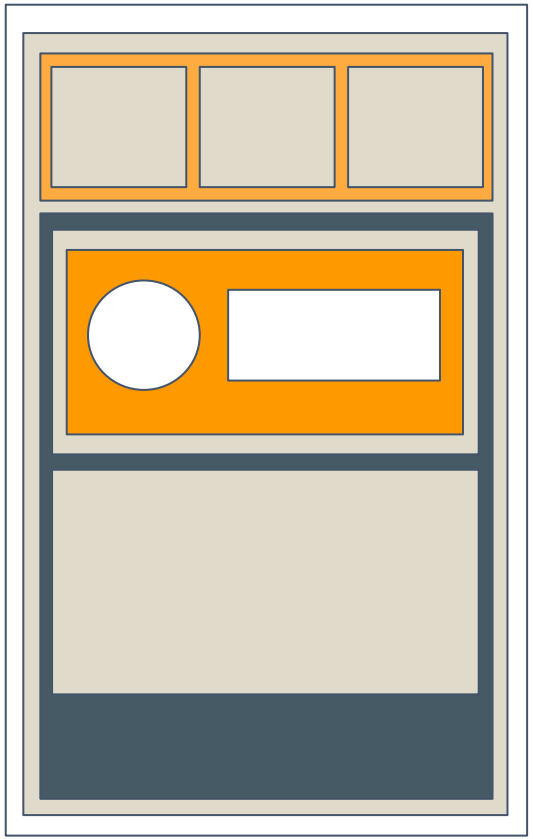
- ConstraintLayout 이전에는 LinearLayout, RelativeLayout 이 많이 쓰였지만, ConstraintLayout으로 모두 대체가 가능합니다.

LinearLayout

`LinearLayout`은 세로 또는 가로의 단일 방향으로 모든 하위 요소를 정렬하는 뷰 그룹입니다. `android:orientation` 속성을 사용하여 레이아웃 방향을 지정할 수 있습니다.

> ★ **참고:** 더 나은 성능과 도구 지원을 위해 ConstraintLayout을 사용하여 레이아웃을 빌드해야 합니다.

LinearLayout



```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.a
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <LinearLayout...>

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="160dp"
        android:orientation="vertical">

        <LinearLayout...>

        <LinearLayout...>

    </LinearLayout>
</LinearLayout>
```

과제 : RelativeLayout, FrameLayout

- 어떤 UI를 구성할 때 많이 쓰였던 Layout인지 간단히만 정리해봅니다.
- 이전 슬라이드의 LinearLayout 만으로 구현된 UI를 RelativeLayout과 조합하면 뷰 계층을 줄일 수 있을지 시도해봅니다.
- 왜 지금은 사용하지 않게 되었을지도 생각해 봅시다.

Style & Theme

Style

- View의 모양을 지정하는 속성의 모음입니다.
- font color, font size, background color 등
- Style 상속을 지원하며, 다른 스타일의 속성을 override도 가능합니다.

(1) 점 표기법

```
<style name="CustomButton">
    <item name="android:background">@color/purple_500</item>
    <item name="android:textColor">@color/white</item>
    <item name="android:textAllCaps">false</item>
</style>

<style name="CustomButton.Bold">
    <item name="android:textStyle">bold</item>
</style>
```

(2) parent 속성

```
<style name="CustomButton" parent="Widget.MaterialComponents.Button">
    <item name="android:backgro
    <item name="android:textCol
    <item name="android:textAllCaps">false</item>
</style>
```

Android 가 지원하는 앱 스타일

Theme

- 개별 View 뿐만 아니라 앱, Activity, View 계층구조에 전체적으로 적용되는 속성 모음
- status bar, window background와 같이 View가 아닌 요소에도 적용할 수 있습니다.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    <application
            android:allowBackup="false"
            android:supportsRtl="true"
            android:theme="@style/AppTheme">
```

```
<style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
    <!-- Customize your theme here. -->
    <item name="colorPrimary">@color/colorPrimary</item>
    <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
    <item name="colorAccent">@color/colorAccent</item>
    <item name="buttonStyle">@style/CustomButton.Bold</item>
```

Style을 Theme로 적용하기

- Style의 한계 : Widget은 하나의 Style만 적용할 수 있습니다.
- Theme 활용 : 개별 Widget 에 적용하지 않고도 앱 전체에 Style을 적용할 수 있습니다.

```
<style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
    <!-- Customize your theme here. -->
    <item name="colorPrimary">@color/colorPrimary</item>
    <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
    <item name="colorAccent">@color/colorAccent</item>
    <item name="buttonStyle">@style/CustomButton.Bold</item>
```

Activity

Activity

- Android 앱을 만드는 **주요 component** 중 하나 입니다.
- 사용자와 상호작용하기 위한 entry point 입니다.

  → 화면을 통해 앱이 상호작용할 수 있도록 기능을 기원합니다.

- UI를 포함하는 하나의 screen을 의미합니다.

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }
}
```

Activity - 사용자와 상호작용은 어떻게?

- Layout inflate
  - view hierarchy에 대응되는 객체로 생성하는 것을 의미합니다.
  - inflate 되면, layout 파일의 각 Widget이 자신의 속성에 정의된 대로 인스턴스를 생성합니다.
  - resource id를 인자로 받아서 해당 Widget 객체의 참조를 얻을 수 있게 됩니다.

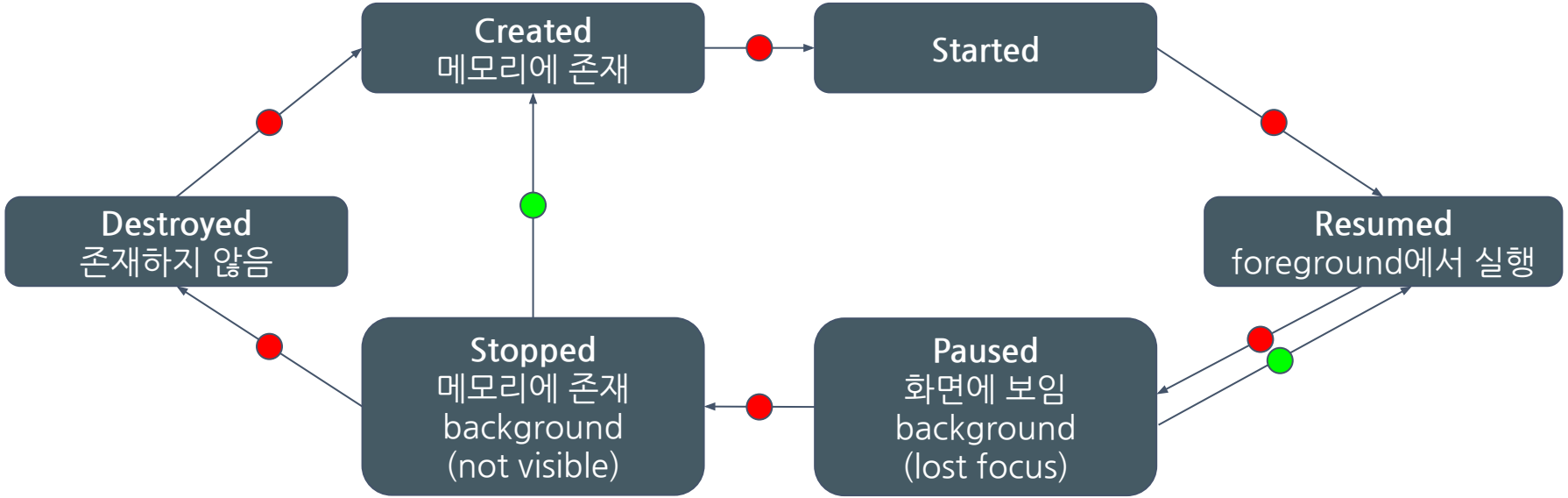
- 사용자와 상호작용 : ex. 버튼 클릭
  - 이벤트 기반으로 구동됩니다 = 이벤트 발생을 기다립니다

```
class MainActivity : AppCompatActivity() {
    private lateinit var btnYes: Button

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        btnYes = findViewById(R.id.btn_answer_yes)
        btnYes.setOnClickListener { view ->
            // TODO Yes 버튼을 클릭했을 때에 대한 처리
        }
```

Activity 상태 - 화면의 생성부터 소멸까지



용어

- **Focus**

  Activity가 사용자와 상호작용할 수 있을 때, focus를 가지고 있다고 말합니다.

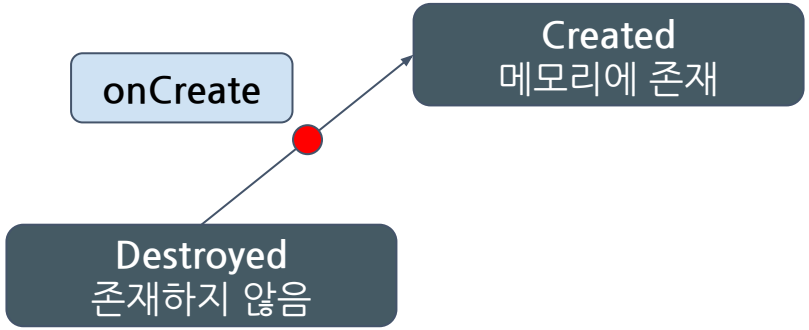
- **Foreground**

  Activity가 화면에 보이고 있는 경우

- **Background**

  Activity 가 화면에 보이지 않는 경우를 의미합니다.

Activity 상태와 생명주기



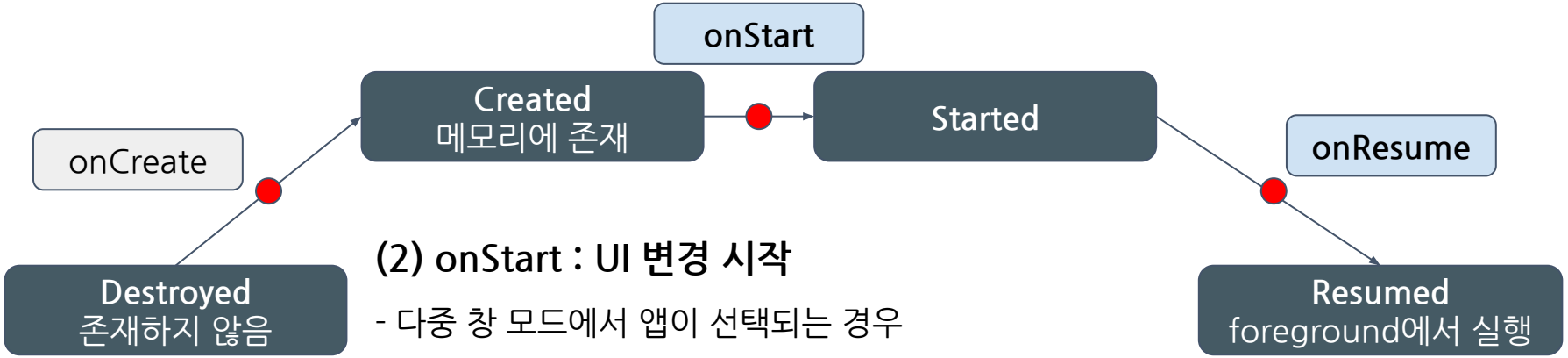
(1) onCreate : UI 준비

- App을 처음 실행할 때 호출된다

[주로 처리하는 작업]

- Widget을 inflate해 View 객체로 생성한 후 화면에 보여준다
- 사용자와의 상호 작용을 처리하기 위해 Widget에 listener(이벤트 응답 처리 객체)를 설정한다.
- 외부의 모델 데이터를 연결한다.

Activity 상태와 생명주기



(2) onStart : UI 변경 시작

- 다중 창 모드에서 앱이 선택되는 경우

[주로 처리하는 작업]

- 애니메이션, 데이터 갱신 시작

Activity gains focus

(3) onResume : 현재 상호작용하는 Activity

- 오버뷰 화면에서 Activity를 선택하는 경우
- 하나의 Activity만 Resumed 상태일 수 있다

일반적으로 앱을 실행하면, onCreate → onStart → onResume은 순차적으로 빠르게 호출됩니다

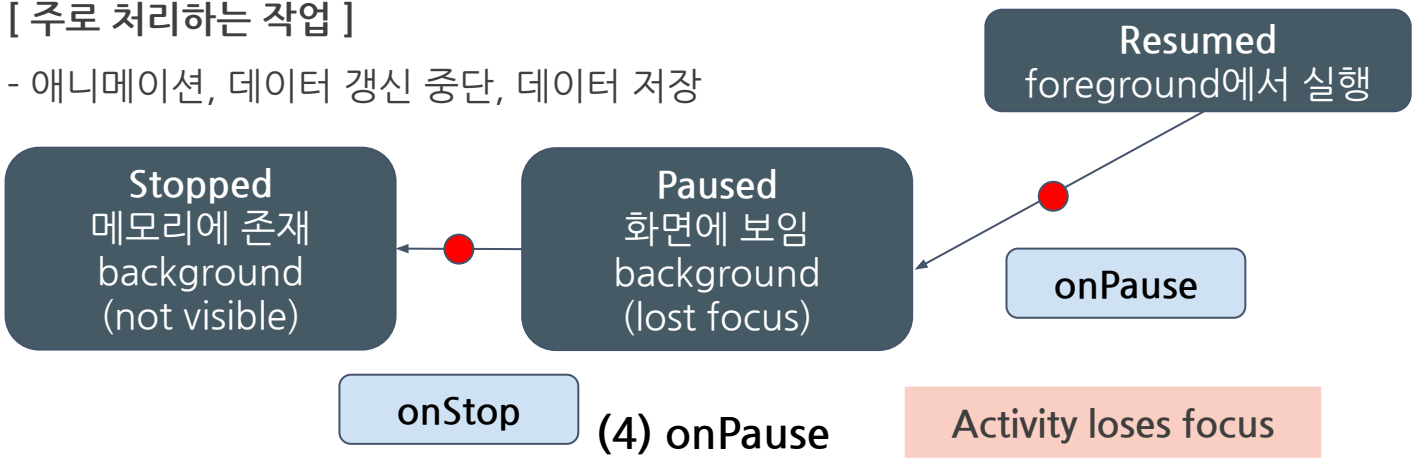
Activity 상태와 생명주기

(5) onStop : UI 변경 중단

- 홈버튼을 눌러 다른 앱을 사용할 때
- onStop 호출 이후에 Activity는 종료시킬 대상이 됩니다

[주로 처리하는 작업]

- 애니메이션, 데이터 갱신 중단, 데이터 저장

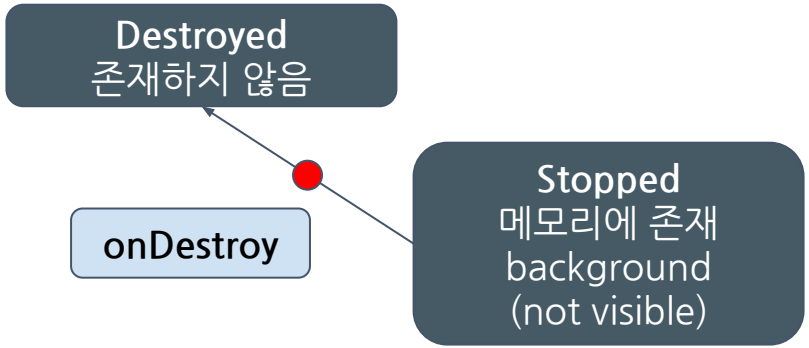


(4) onPause

Activity loses focus

- 사용자가 다중창 모드를 시작해서 다른 앱을 사용중인 경우

Activity 상태와 생명주기



(6) onDestroy : Activity 소멸

- back 버튼을 탭해서 앱을 종료시킬 때
- 오버뷰 화면에서 해당 앱을 목록에서 제거할 때

[주로 처리하는 작업]

- 이전에 해제되지 않은 모든 리소스를 해제합니다.

Activity 상태 변경에 따라 생명주기 콜백이 호출됩니다.
생명주기를 이해하고 적시에 알맞은 작업을 해야 합니다.

Activity 상태와 생명주기