

11S3211 – Mobile Application Development



Android Development and Component (Styles and Themes)

Presented by:
Ranty Deviana Siahaan, S.Kom., M.Eng.

**Lecturers: HTO, RDS
- RPL 11S3211–
- Sem Genap 2023/2024 -**

Learning Outcome

- ❑ Mahasiswa memahami beberapa aspek penting dalam pengembangan android, seperti: *SharedPreferences and Data Saving, Device Compatibility in Android 12, Fragment Operations, dan Sending Data Between Screens.*
- ❑ Mahasiswa mampu memahami dan mengimplementasikan *styles dan themes* pada aplikasi android.

Outline

- ❑ **Feedback Quiz**
- ❑ *SharedPreferences and Data Saving*
- ❑ *Device Compatibility in Android 12*
- ❑ *Fragment Operations*
- ❑ *Sending Data Between Screens*
- ❑ **Styles and Themes**



Feedback Quiz

Soal 01

1	<code>data class Person(val name: String, val age: Int = 20)</code>	
2		
3	<code>fun main() {</code>	Apa hasil keluaran dari kode program tersebut?
4	<code>val person = Person("Lela")</code>	a. <u>Usia: 20</u>
5	<code>person.age = 23</code>	b. <u>Usia: 23</u>
6	<code>print("Usia: \${person.age}")</code>	c. 20
7	<code>}</code>	d. 23
		e. <u>Terdapat error pada kode program</u>

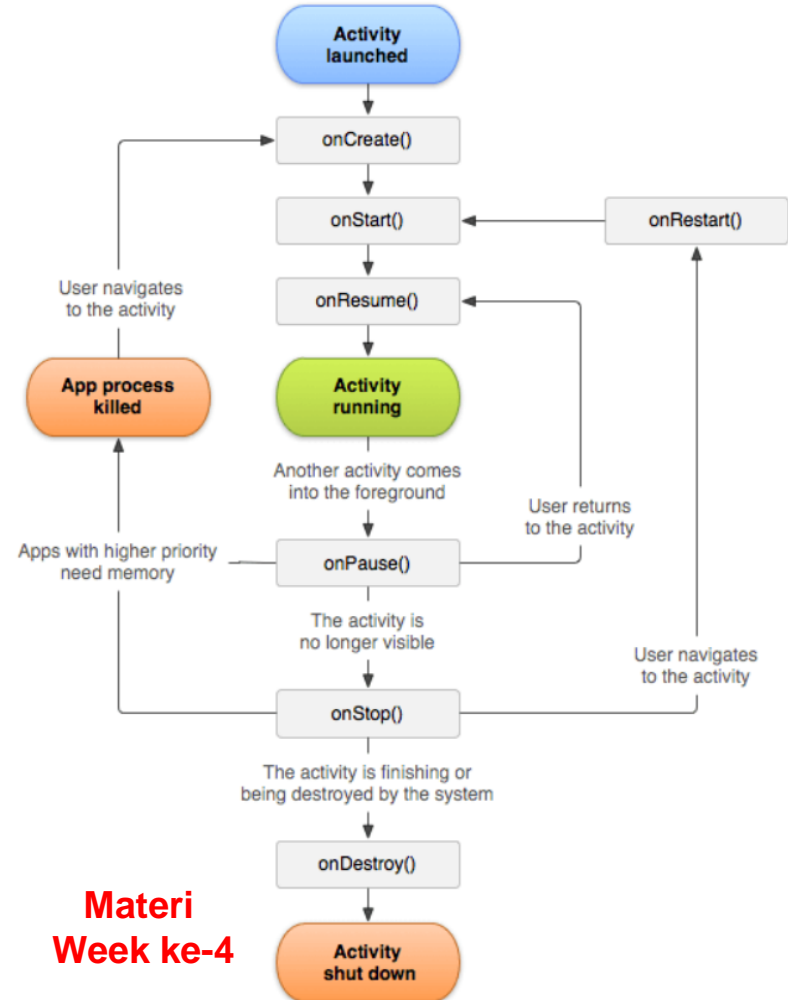
- ☐ **Output:** Menghasilkan kesalahan saat di-*compile*.
- ☐ **Tujuannya:** Mengubah nilai properti *age* dari objek *person* setelah objek diinisialisasi.
- ☐ **Kesalahan:** ingin mengubah nilai properti *age* setelah objek *person* dibuat, tentu tidak diizinkan karena properti tersebut dideklarasikan dengan *val* (*read only*).

Soal 02

```
1 class MainActivity : AppCompatActivity() {
2     override fun onCreate(savedInstanceState: Bundle?) {
3         super.onCreate(savedInstanceState)
4         setContentView(R.layout.activity_main)
5         Log.d("MainActivity", "onCreate()")
6     }
7     override fun onPause() {
8         super.onPause()
9         Log.d("MainActivity", "onPause()")
10    }
11    override fun onResume() {
12        super.onResume()
13        Log.d("MainActivity", "onResume()")
14    }
15    override fun onStart() {
16        super.onStart()
17        Log.d("MainActivity", "onStart()")
18    }
19    override fun onStop() {
20        super.onStop()
21        Log.d("MainActivity", "onStop()")
22    }
23    override fun onDestroy() {
24        super.onDestroy()
25        Log.d("MainActivity", "onDestroy()")
26    }
27 }
```

Urutan aktifitas yang tepat saat aplikasi dijalankan adalah...

- a. onCreate(), onPause(), onResume(), onStart(), onStop(), onDestroy()
- b. onCreate(), onResume(), onPause(), onStart(), onStop(), onDestroy()
- c. onCreate(), onStart(), onPause(), onResume(), onStop(), onDestroy()
- d. onCreate(), onStart(), onResume(), onPause(), onStop(), onDestroy()
- e. onCreate(), onStart(), onStop(), onResume(), onPause(), onDestroy()



**Materi
Week ke-4**

Soal 03

Yang perlu dilakukan sebelum menggunakan smartphone Android dalam Development Mode adalah...

- a. Menginstal semua aplikasi yang diperlukan.
- b. Melakukan root pada perangkat.
- c. Memeriksa persyaratan garansi.
- d. Mengatur akun Google.
- e. **Mengaktifkan USB Debugging.**

- ☐ Opsi A-D berkaitan dengan persiapan perangkat android itu sendiri untuk pengembangan.
- ☐ **USB Debugging** → fitur pada perangkat Android yang memungkinkan untuk menghubungkan perangkat ke komputer dan menggunakan **Android SDK (Software Development Kit)** untuk mengembangkan, menguji, dan debug aplikasi Android.

Sehingga, sebelum menggunakan perangkat Android untuk pengembangan, perlu mengaktifkan **USB Debugging** di perangkat kita. Karena memungkinkan komunikasi antara perangkat dan IDE (*Integrated Development Environment*) seperti Android Studio atau Eclipse.

SharedPreferences and Data Saving

SharedPreferences

- ❑ *SharedPreferences* adalah mekanisme penyimpanan data yang disediakan oleh Android.
- ❑ Dapat digunakan untuk menyimpan data primitif, seperti bilangan bulat, boolean, atau string.
- ❑ Data yang disimpan dalam *SharedPreferences* akan tetap tersedia bahkan setelah aplikasi ditutup dan dijalankan kembali.
- ❑ *SharedPreferences* juga dapat digunakan untuk mengakses preferensi pengguna, seperti *setting* aplikasi.

Data Saving

- ❑ *Data Saving* adalah istilah yang lebih umum yang mengacu pada berbagai teknik untuk menyimpan data pada aplikasi Android.
- ❑ Selain *SharedPreferences*, cara lain untuk menyimpan data pada aplikasi Android adalah:
 - *SQLite database*: untuk menyimpan data terstruktur dalam bentuk tabel.
 - *File*: untuk menyimpan data dalam bentuk file, seperti gambar atau dokumen teks.
 - *Network*: untuk menyimpan data pada *server* jarak jauh, seperti *Firebase* atau *server web*.

Device Compatibility in Android 12

Device Compatibility Android 12

- ❑ *Device Compatibility Android 12* mengacu pada kemampuan aplikasi atau sistem Android untuk berjalan dan berfungsi dengan baik pada perangkat tertentu yang menjalankan Android 12.
- ❑ Android 12 adalah versi terbaru dari sistem operasi Android, dirilis pada tahun 2021, dan dilengkapi dengan beberapa fitur dan perbaikan baru.
- ❑ Dalam hal kompatibilitas perangkat, Android 12 telah memperkenalkan beberapa perubahan untuk memastikan aplikasi dan sistem berjalan dengan lancar pada berbagai perangkat.

Device Compatibility Android 12 (Cont'd)

- ❑ Perubahan ini antara lain meliputi:
 - Dukungan yang ditingkatkan untuk perangkat lipat dan layar besar.
 - Fitur keamanan yang ditingkatkan.
 - Kompatibilitas melalui Google Play.
 - Kompatibilitas melalui *Android Compatibility Definition Document* (CDD) atau Dokumen Definisi Kompatibilitas Android.
 - Pengujian melalui *Android Compatibility Test Suite* (CTS).

Device Compatibility Android 12 (Cont'd)

- ❑ Kompatibilitas perangkat pada Android 12 adalah tentang memastikan bahwa aplikasi dan sistem dapat berjalan dengan baik pada berbagai perangkat, termasuk yang memiliki ukuran layar, rasio aspek, dan persyaratan keamanan yang berbeda.
- ❑ Yang dicapai melalui dukungan yang ditingkatkan untuk berbagai jenis perangkat, pengujian yang ketat, dan kepatuhan terhadap Dokumen Definisi Kompatibilitas Android.

Fragment Operations

Fragment

- ❑ *Fragment* adalah salah satu komponen utama dalam Android yang digunakan untuk membangun antarmuka pengguna yang fleksibel dan modular.
- ❑ Dalam Android Kotlin, kita dapat melakukan beberapa operasi pada *fragment* untuk memanipulasi tampilan dan perilaku antarmuka pengguna secara dinamis.

Fragment (Cont'd)

- ❏ Beberapa operasi *fragment* yang dapat dilakukan di Android:
 - Menambahkan *Fragment*: **add()**
 - Mengganti *Fragment*: **replace()**
 - Menghapus *Fragment*: **remove()**
 - Menyimpan Data *Fragment*: **onSaveInstanceState()**
 - Memulihkan Data: **onCreate()** (menggunakan objek *Bundle*)
 - Komunikasi antar *Fragment*: **findFragmentById()** dan **findFragmentByTag()**
 - Menampilkan Dialog: **DialogFragment**
- ❏ Dengan melakukan operasi tersebut, kita dapat memanipulasi tampilan dan perilaku antarmuka pengguna secara dinamis, dan membuat aplikasi Android yang lebih fleksibel dan modular.

Sending Data Between Screens

Sending Data Between Screens

- ❑ Dalam pengembangan aplikasi Android Kotlin, seringkali kita perlu mengirim data dari satu *Activity* ke *Activity* lainnya.
- ❑ Ada beberapa cara yang dapat dilakukan untuk mengirim data antar *Activity* di Android Kotlin, yaitu sebagai berikut:
 1. Menggunakan ***Intent***: Salah satu cara yang paling umum adalah dengan menggunakan ***Intent***. Kita dapat menambahkan data ekstra ke *Intent* menggunakan metode ***putExtra()*** dengan parameter *key-value*. Kemudian, di *Activity* tujuan, kita dapat mengambil data tersebut menggunakan metode ***getIntent().get<type>(key)***.

Sending Data Between Screens (Cont'd)

2. Menggunakan ***Bundle***: Kita juga dapat menggunakan objek *Bundle* untuk mengirim data antar *Activity*. Kita dapat membuat objek *Bundle* dan menambahkan data ke dalamnya menggunakan metode *put<type>(key, value)*. Kemudian, kita dapat menempatkan *Bundle* ke dalam *Intent* menggunakan metode *putExtras()*.

Sending Data Between Screens (Cont'd)

3. Menggunakan ***Parcelable***: Jika kita perlu mengirim objek kompleks antar *Activity*, kita dapat mengimplementasikan *Parcelable* pada objek tersebut. *Parcelable* adalah antarmuka *marker* yang memungkinkan objek dikemas ke dalam *Bundle* dan dikirimkan antar *Activity*.

Sending Data Between Screens (Cont'd)

4. Menggunakan ***ViewModel***: Untuk pengiriman data yang lebih kompleks atau data yang perlu disimpan selama siklus hidup *Activity*, kita dapat menggunakan ***ViewModel***. ***ViewModel*** adalah objek yang dirancang untuk menyimpan dan mengelola data selama siklus hidup *Activity* dan *Fragment*.

Sending Data Between Screens (Cont'd)

- ❑ Untuk mengirim data antar *Activity* di Android Kotlin, kita perlu menentukan *Activity* asal dan *Activity* tujuan, dan kemudian memilih metode yang sesuai dengan kebutuhan aplikasi.
- ❑ Setelah itu, kita dapat menambahkan data ke *Intent* atau *Bundle*, dan menavigasi ke *Activity* tujuan menggunakan metode ***startActivity()*** atau ***startActivityForResult()***.
- ❑ Di *Activity* tujuan, kita dapat mengambil data yang dikirimkan dan menampilkannya di UI atau menyimpannya untuk penggunaan selanjutnya.

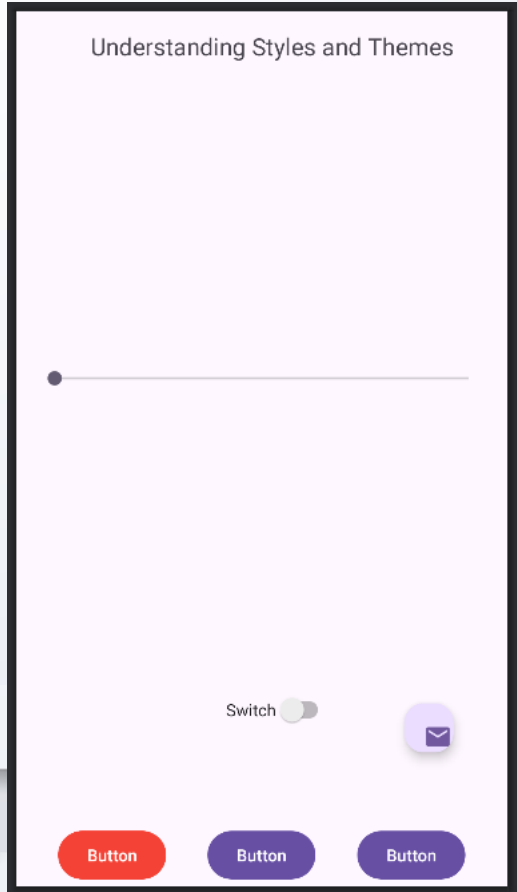


STYLES & THEMES

Intro

- ❑ **Styles** dan **Themes**: mengatur tampilan dan gaya secara terpusat, meningkatkan konsistensi dan kemudahan pemeliharaan pada aplikasi android.
- ❑ Untuk memisahkan detail desain aplikasi dari struktur dan perilaku antarmuka pengguna (UI). Sehingga, dapat **mengatur tampilan aplikasi secara konsisten** dan **memudahkan perubahan gaya atau tata letak secara global** dengan **hanya mengubah satu file konfigurasi**.
- ❑ Memungkinkan untuk menghindari pengulangan kode yang tidak perlu dan meningkatkan efisiensi pengembangan.

Intro (Cont'd)



- Kita dapat mengatur atribut pada *layout*.
- Kita dapat menerapkan *styles* pada sebuah *view*.
- Kita dapat menerapkan sebuah *theme* pada *layout*.
- Saat *create project* android, secara *default*, proyek tersebut akan menggunakan **Theme Material Design** yang dikembangkan oleh Google, yang menekankan pengalaman pengguna yang bersih, responsif, dan intuitif.
- Mencakup prinsip-prinsip seperti penggunaan warna, tipografi yang jelas, animasi yang halus, dan komponen UI yang konsisten.

Styles

- ❑ **Styles** → kumpulan *property*/atribut yang diterapkan ke elemen UI di XML *layout* file atau secara program di dalam kode Kotlin.
- ❑ Dengan **styles** dapat mengatur warna, ukuran *font*, latar belakang, dan atribut-atribut lainnya untuk *view* tertentu.
- ❑ **Style** didefinisikan dalam sebuah sumber daya XML yang terpisah dari XML yang menentukan *layout*.

```
<Style name="CustomFontStyle">
    <item name="android:layout_width">fill_parent</item>
    <item name="android:layout_height">wrap_content</item>
    <item name="android:capitalize">characters</item>
    <item name="android:typeface">monospace</item>
    <item name="android:textSize">12pt</item>
    <item name="android:textColor">#00F000</item>
</Style>
```

Styles (Cont'd)

- ❑ **Style** dalam Android mirip dengan CSS dalam desain *web*, yaitu memungkinkan kita memisahkan desain dari konten.
- ❑ Dengan menggunakan *style*, kita dapat memisahkan definisi tampilan dari struktur *layout*, membuat kode lebih bersih dan lebih mudah dikelola.
- ❑ Didefinisikan dalam file ***styles.xml*** yang terletak di dalam folder ***res/values*** pada proyek Android.

Contoh penerapan *Styles*

- ❑ Membuat ***style*** untuk mengatur tampilan teks pada *TextView* dalam aplikasi android.
- ❑ Awalnya, properti-properti seperti warna *font* dan jenis huruf ditentukan langsung dalam XML *layout*.
- ❑ Namun, dengan menggunakan *style*, properti-properti tersebut dipindahkan ke dalam sebuah *style* terpisah.
- ❑ Sehingga, tampilan XML menjadi lebih ringkas dan lebih fokus pada konten.

Contoh penerapan *Styles* (Cont'd)

```
<TextView  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:textColor="#00FF00"  
    android:typeface="monospace"  
    android:text="@string/hello" />
```



```
<TextView  
    style="@style/CodeFont"  
    android:text="@string/hello" />
```

Langkah-langkah Mendefinisikan *Styles*

1. **Menyimpan File XML:** Simpan file XML yang berisi definisi *styles* di direktori ***res/values/*** dari proyek. Nama file XML tersebut bebas, namun harus memiliki ekstensi ***.xml*** dan disimpan dalam folder ***res/values/***.
2. **Menentukan Root Node:** *Root* node dari file XML tersebut haruslah ***<resources>***. Ini menandakan bahwa file tersebut berisi definisi *resource* untuk aplikasi.
3. **Menambahkan Element *<style>*:** Untuk setiap *style* yang ingin di buat, tambahkan sebuah elemen ***<style>*** ke dalam file tersebut dengan sebuah nama yang secara unik mengidentifikasi *style* tersebut (atribut ini wajib ada).

Langkah-langkah Mendefinisikan *Styles*

4. Menambahkan **Element *<item>***: Untuk setiap properti dari *style* tersebut, tambahkan sebuah elemen ***<item>*** dengan sebuah nama yang mendeklarasikan properti *style* dan sebuah nilai yang sesuai dengan properti tersebut (atribut ini wajib ada). Nilai dari item ini bergantung pada properti *style* yang bersangkutan.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="CodeFont" parent="@android:style/TextAppearance.Medium">
        <item name="android:layout_width">fill_parent</item>
        <item name="android:layout_height">wrap_content</item>
        <item name="android:textColor">#00FF00</item>
        <item name="android:typeface">monospace</item>
    </style>
</resources>
```


Contoh *Styles* pada File XML

- ❑ Setiap elemen ***child*** yang termasuk dalam elemen ***<resources>*** ini akan dianggap sebagai sumber daya aplikasi.
- ❑ Ketika proyek dikompilasi, elemen-elemen ini akan diubah menjadi objek sumber daya aplikasi yang dapat diakses melalui nilai atribut ***name*** dari elemen ***<style>***.

Contoh *Styles* pada File XML (*Cont'd*)

```
1 <resources>
2 <style name="AppTheme" parent="Theme.MaterialComponents.Light">
3   <item name="colorPrimary">@color/colorPrimary</item>
4   <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
5   <item name="colorAccent">@color/colorAccent</item>
6 </style>
7 <color name="colorPrimary">#3F51B5</color>
8 <color name="colorPrimaryDark">#303F9F</color>
9 <color name="colorAccent">#FF4081</color>
10 </resources>
```

Terlihat sebuah **<style>** dengan nama **AppTheme** didefinisikan dengan tiga atribut: **colorPrimary**, **colorPrimaryDark**, dan **colorAccent**. Nilai-nilai dari atribut-atribut ini diambil dari sumber daya warna yang didefinisikan di bawahnya.

Themes

- ❑ **Themes** → kumpulan *property* atau *styles* yang diterapkan ke seluruh atau sebagian aplikasi, aktivitas, atau hirarki tampilan.
- ❑ Untuk mengatur tata letak umum, skema warna, gaya teks, dan atribut-atribut lainnya yang diterapkan secara konsisten ke berbagai bagian aplikasi.
- ❑ **Themes** dapat juga menerapkan **styles** pada elemen *non-view* seperti **status bar** dan **windows background**.

Themes (Cont'd)

- ❑ Penggunaan ***themes*** membantu menjaga konsistensi tampilan UI di seluruh aplikasi dan memungkinkan untuk dengan cepat mengubah penampilan keseluruhan aplikasi dengan mengganti ***theme*** yang aktif.
- ❑ Kita dapat mendefinisikan ***themes*** dalam file ***styles.xml*** atau ***themes.xml*** yang terletak di dalam folder ***res/values*** pada proyek Android.

Theme (Cont'd) – Baseline Material Color Theme

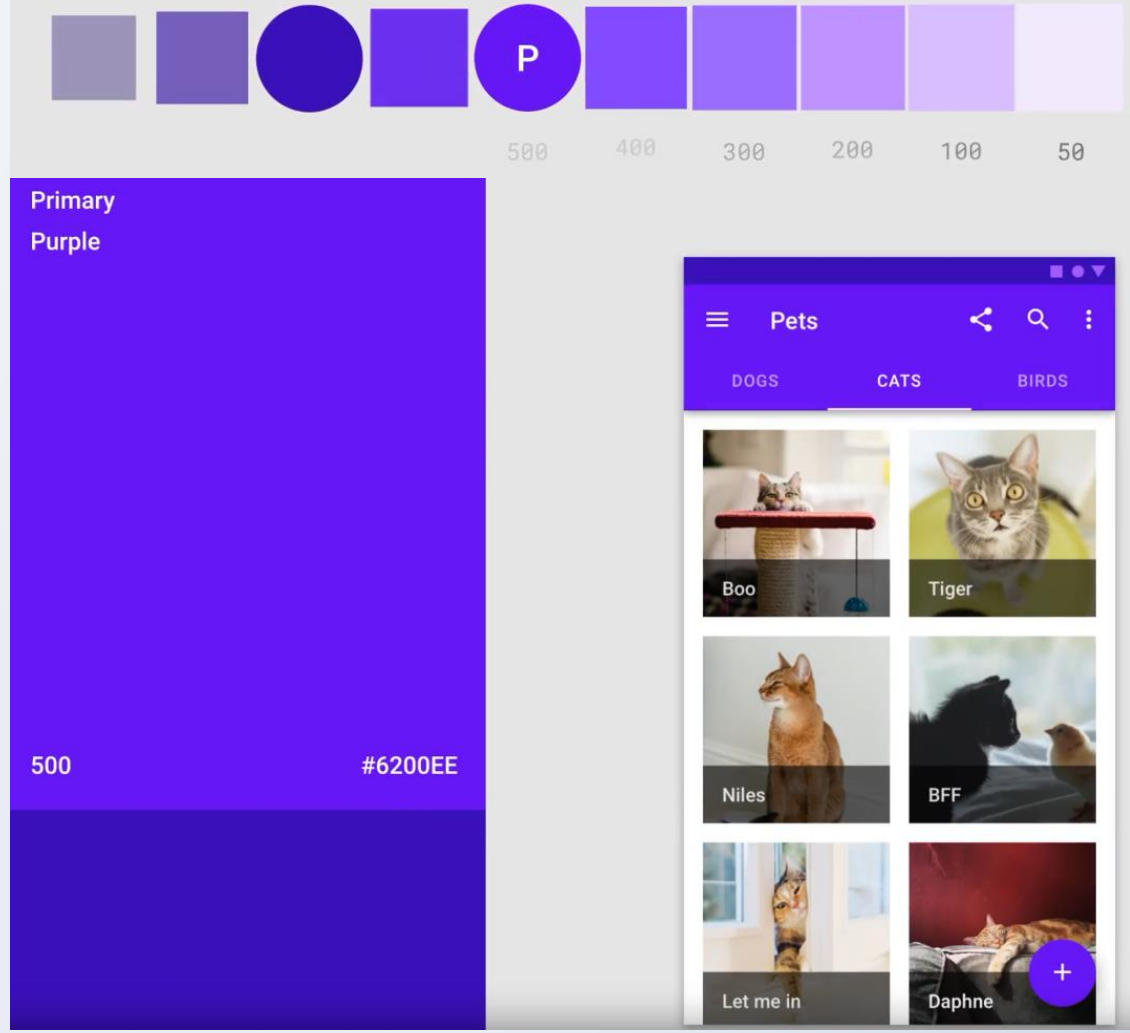
Primary 1 #6200EE	Primary Variant 2 #3700B3	Secondary 3 #03DAC6	Secondary Variant 4 #018786
Background 5 #FFFFFF	Surface 6 #FFFFFF	Error 7 #B00020	
On Primary 8 #FFFFFF		On Secondary 9 #000000	
On Background 10 #000000	On Surface 11 #000000	On Error 12 #FFFFFF	

This includes default colors for:

- Primary and secondary colors
- Variants of primary and secondary colors
- Additional UI colors, such as colors for backgrounds, surfaces, errors, typography, and iconography.

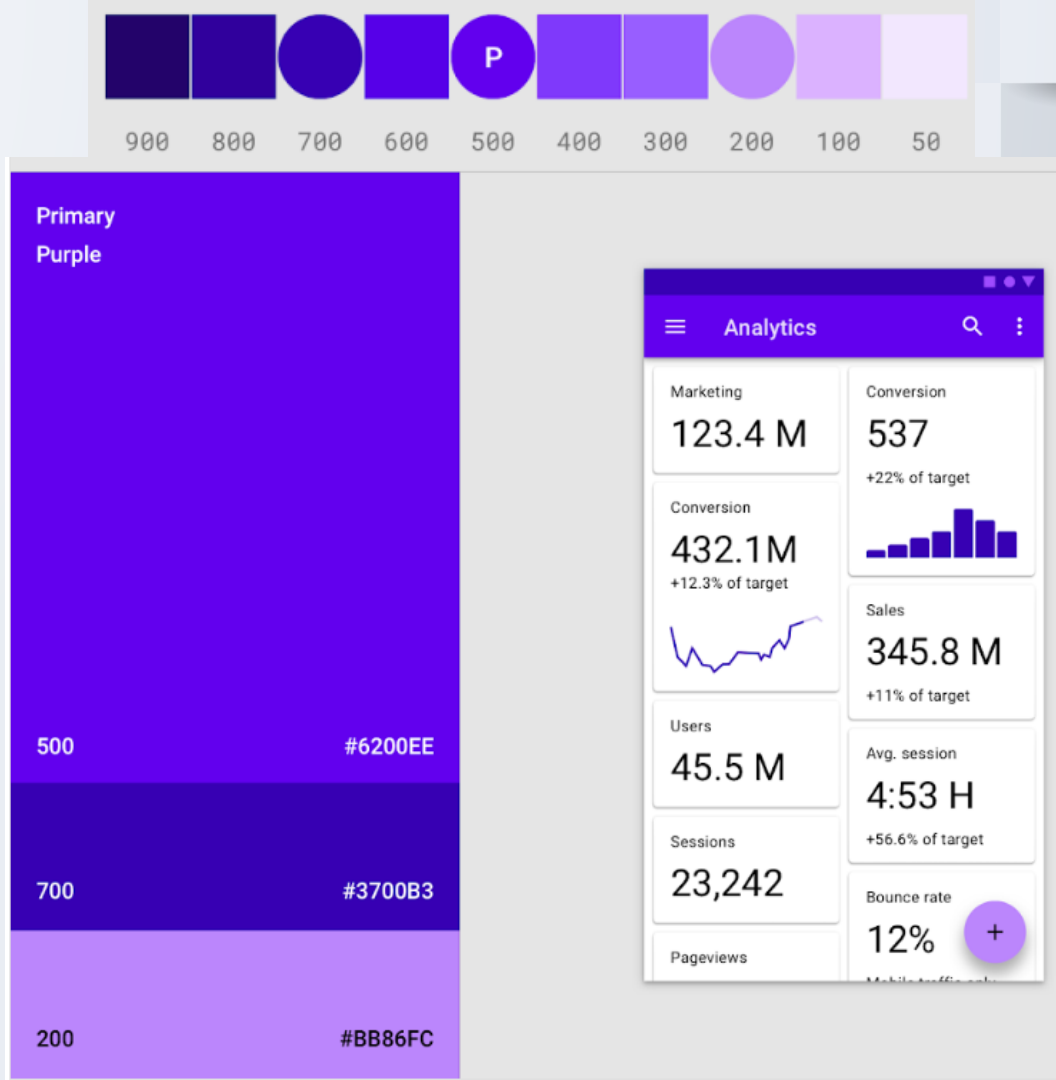
Theme (Cont'd) – Baseline Material Color Theme

Top app bar menggunakan variasi warna *primer* yang terang dan gelap untuk membedakannya dari sistem bar-nya.



Theme (Cont'd) – Baseline Material Color Theme

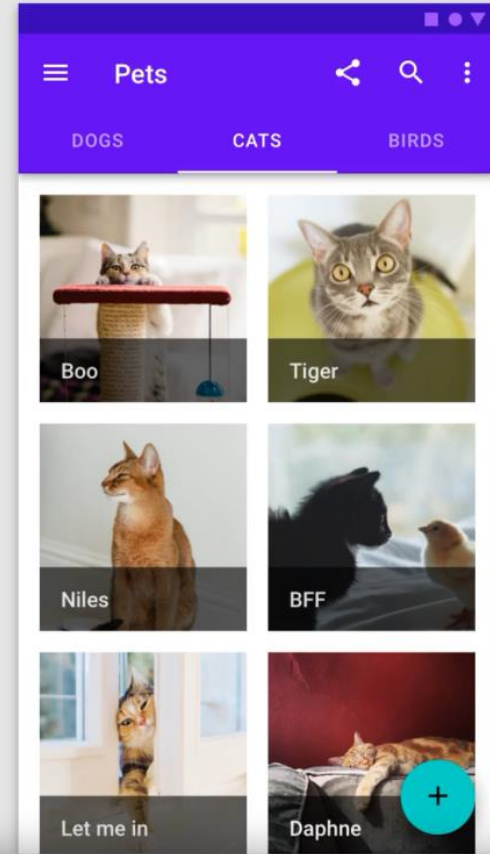
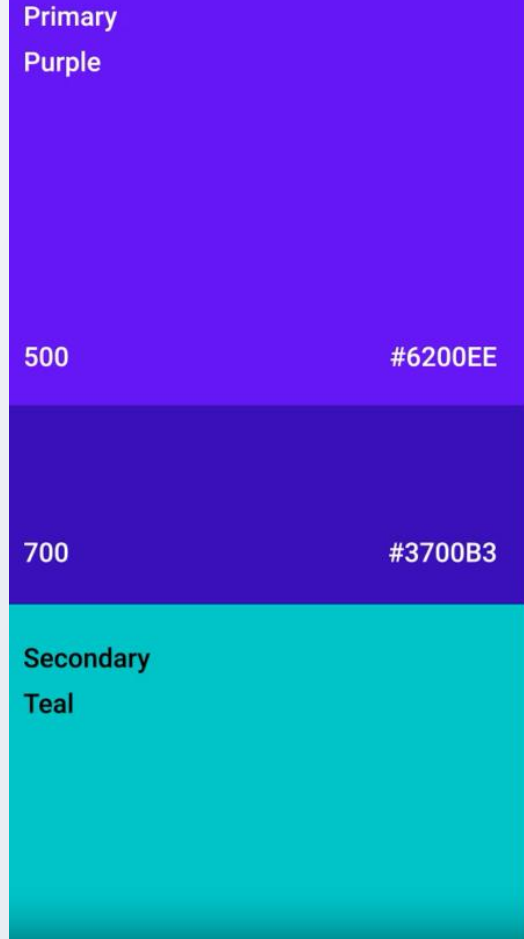
Menggunakan *primary color* dan
dua *primary variants*.



Theme (Cont'd) – Baseline Material Color Theme

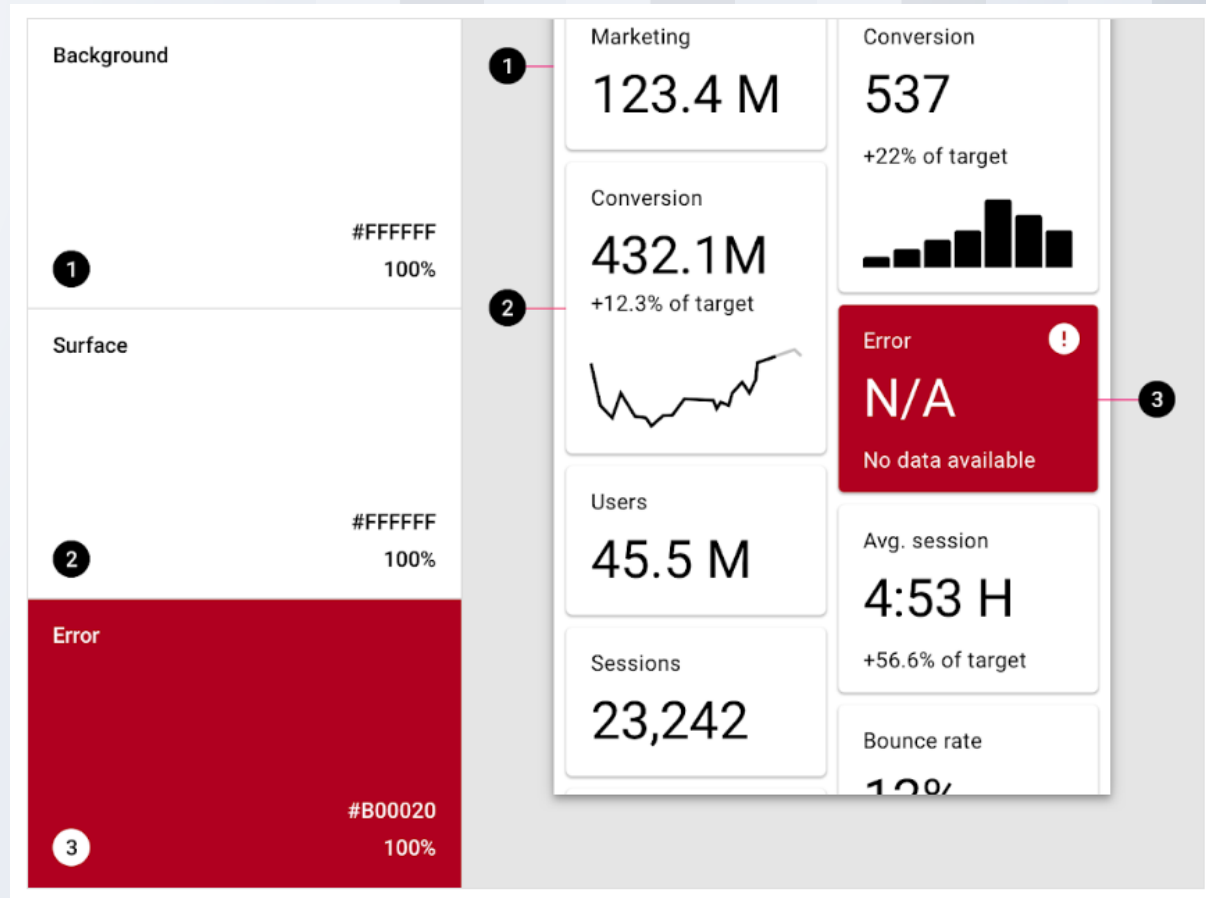
Dark and light variants untuk
warna primer dan sekunder.

- Floating action buttons
- Selection controls, like sliders and switches
- Highlighting selected text
- Progress bars
- Links and headlines



Theme (Cont'd) – Baseline Material Color Theme

Baseline colors untuk
background, surface, dan
error color.



Theme (Cont'd) – Baseline Material Color Theme

Untuk menjaga keterlihatan elemen dan keterbacaan teks, kita dapat menyesuaikan skema warna yang berbeda untuk *theme light* dan *dark*.



Theme (Cont'd) – Material Design Palette

material design palette

PALETTES ICONS COL up More awesome designs on UpLabs → Learn to code

RED PINK PURPLE DEEP PURPLE

INDIGO BLUE LIGHT BLUE CYAN TEAL

GREEN LIGHT GREEN LIME YELLOW AMBER

ORANGE DEEP ORANGE BROWN GREY BLUE GREY

Palette preview

Full Palette colors below

Daily Design Showcase
Visit UpLabs


Daily Resources for Designers & Developers

Your Palette ^

DOWNLOAD

#303F9F	#C5CAE9	#3F51B5	#FFFFFF
DARK PRIMARY COLOR	LIGHT PRIMARY COLOR	PRIMARY COLOR	
#03A9F4	#212121	#757575	#BDBDBD
ACCENT COLOR	PRIMARY TEXT	SECONDARY TEXT	

Theme (Cont'd) – Material Design Component

 MATERIAL DESIGN

Components

App bars: bottom

App bars: top

Backdrop

Banners

Bottom navigation

Buttons

Buttons: floating action button

Cards

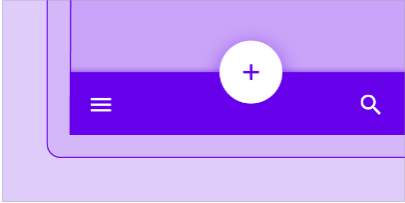
Checkboxes

Chips

Data tables

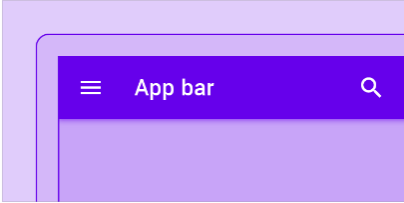
Date pickers

All ComponentsAndroidWebFlutteriOS



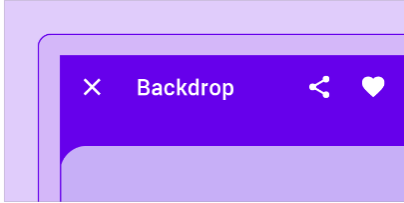
App bars: bottom

A bottom app bar displays navigation and key actions at the bottom of mobile screens



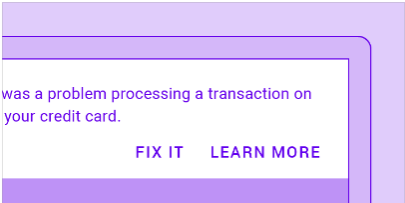
App bars: top

The top app bar displays information and actions relating to the current screen



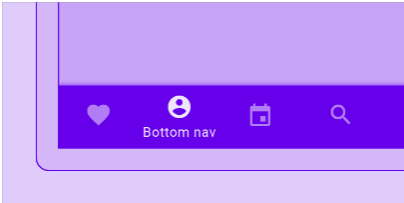
Backdrop

A backdrop appears behind all other surfaces in an app, displaying contextual and actionable content



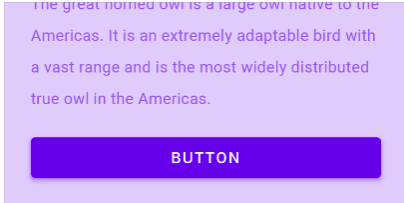
Banners

A banner displays a prominent message and related optional actions



Bottom navigation

Bottom navigation bars allow movement between primary destinations in an app



Buttons

Buttons allow users to take actions, and make choices, with a single tap

Inheritance in Styles and Themes

- Dalam Android, *inheritance* atau pewarisan memungkinkan kita untuk mewarisi properti-properti dari *style* atau tema yang sudah ada sebelumnya, sehingga kita dapat memodifikasinya atau menambahkan properti-properti baru sesuai kebutuhan.

Cara *Inheritance* pada *Styles* dan *Themes*

- Menggunakan Atribut ***Parent***: Atribut *parent* dalam elemen ***<style>*** adalah opsional dan menentukan ID sumber daya dari *style* lain yang ingin diwarisi propertinya. Kita dapat mengganti atau menambah properti-properti yang diwarisi jika diperlukan.

```
<style name="GreenText"  
parent="@android:style/TextAppearance">  
    <item name="android:textColor">#00FF00</item>  
</style>
```

Cara *Inheritance* pada *Styles* dan *Themes*

- **Penggunaan Nama *Style*:** Cara lain adalah dengan menambahkan awalan nama *style* yang ingin diwarisi di depan nama *style* baru yang ingin kita buat, dipisahkan dengan tanda titik (.).

```
<style name="CodeFont.Red">  
    <item name="android:textColor">#FF0000</item>  
</style>
```

Cara *Inheritance* pada *Styles* dan *Themes*

- ❑ **Referensi *Style* Baru:** Kita dapat merujuk ke *style* baru yang sudah di buat dengan cara menggunakan sintaks `@style/nama_style_baru`.
- ❑ **Pewarisan Bertingkat:** Kita dapat meneruskan pewarisan seperti ini sebanyak yang diinginkan dengan menyambungkan nama-nama *style* menggunakan tanda titik.

```
<style name="CodeFont.Red.Big">  
    <item name="android:textSize">30sp</item>  
</style>
```


Cara *Inheritance* pada *Styles* dan *Themes*

- **Penerapan pada *View*:** Jika kita menerapkan sebuah *style* pada sebuah *View* yang tidak mendukung semua properti dari *style* tersebut, *View* tersebut akan mengaplikasikan hanya properti-properti yang didukungnya dan mengabaikan yang lainnya.

Penggunaan *Styles* and *Themes* pada UI

- ❑ ***Styles and Themes*** dapat diterapkan pada tampilan individual (*View*) dengan menambahkan atribut *style* ke elemen *View* dalam XML *layout*.
- ❑ Atau, dapat diterapkan pada seluruh *Activity* atau aplikasi dengan menambahkan atribut ***android:theme*** ke elemen ***<activity>*** atau ***<application>*** dalam file Android ***manifest***.
- ❑ *Style* juga dapat diterapkan sebagai tema (*theme*) agar berlaku untuk semua elemen *View* dengan menerapkannya sebagai tema.

Properti *Style* yang Hanya Dapat Diterapkan sebagai Tema

- ❑ Beberapa properti *style* tidak didukung oleh elemen *View* mana pun dan hanya dapat diterapkan sebagai tema.
- ❑ Properti *style* ini berlaku untuk seluruh jendela (*window*) dan tidak untuk jenis *View* tertentu.
- ❑ Contoh properti *style* untuk tema termasuk menyembunyikan judul aplikasi, menyembunyikan status bar, atau mengubah latar belakang jendela.
- ❑ Properti gaya semacam ini tidak terkait dengan objek *View*. Untuk menemukan properti gaya semacam ini, lihat referensi ***R.attr*** dari internet untuk atribut yang dimulai dengan *window*.

Penerapan Theme pada *Activity* atau Aplikasi

- Untuk menetapkan *theme* untuk semua aktivitas aplikasi kita, buka file ***AndroidManifest.xml*** dan sunting tag ***<application>*** untuk menyertakan atribut ***android:theme*** dengan nama gaya (*style*) yang diinginkan.

```
<application android:theme="@style/CustomTheme">
```

- Jika kita ingin menerapkan tema hanya pada satu *Activity* di aplikasi, tambahkan atribut ***android:theme*** ke tag ***<activity>*** tersebut.

Penerapan *Theme* yang Telah Didefinisikan Sebelumnya

- ❑ Android menyediakan banyak tema yang telah ditentukan sebelumnya yang dapat digunakan untuk menghindari menulisnya sendiri.

```
<activity android:theme="@android:style/Theme.Dialog">
```

- ❑ Misalnya, kita dapat menggunakan tema **Dialog** untuk membuat **Activity** kita muncul seperti kotak dialog.
- ❑ Atau jika kita ingin latar belakangnya transparan, kita dapat menggunakan tema **Translucent**.

```
<activity android:theme="@android:style/Theme.Translucent">
```

Penerapan *Theme* yang Telah Didefinisikan Sebelumnya

- Jika kita menyukai sebuah tema, namun ingin menyesuaikannya, kita bisa menambahkan tema tersebut sebagai induk (*parent*) dari tema kustom kita.

```
<color name="custom_theme_color">#b0b0ff</color>
<style name="CustomTheme" parent="android:Theme.Light">
    <item name="android:windowBackground">@color/custom_theme_color</item>
    <item name="android:colorBackground">@color/custom_theme_color</item>
</style>
```

Menggunakan *Styles* dan *Theme* Platform

- ❑ **Platform Android** menyediakan koleksi besar *styles* dan *themes* yang dapat kita gunakan dalam aplikasi.
- ❑ Kita dapat menemukan referensi dari semua gaya yang tersedia dalam kelas **R.style** dari internet.
- ❑ Untuk menggunakan gaya yang terdaftar di sana, gantilah semua garis bawah (_) dalam nama gaya dengan titik (.) Misalnya, Kita dapat menerapkan tema **Theme_NoTitleBar** dengan **@android:style/Theme.NoTitleBar**.

References

- ❑ <https://kotlinlang.org>.
- ❑ <https://www.w3schools.com/kotlin/index.php>.
- ❑ <https://developer.android.com/develop/ui/views/theming/themes>.
- ❑ <https://apipuro.del.ac.id/v1/file/ae0f447287fe9341fbda2a5a6544e252>.
- ❑ <https://www.materialpalette.com/>.
- ❑ <https://m2.material.io/design/color/the-color-system.html#color-theme-creation>.

Selamat Belajar 😊