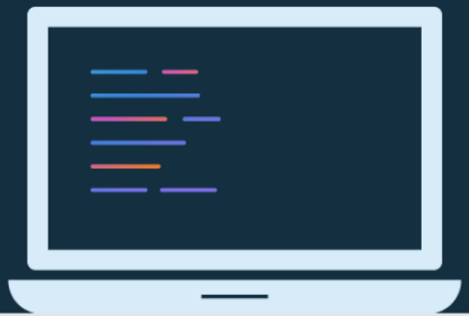
Lesson 13: App UI design



About this lesson

Lesson 13: App UI design

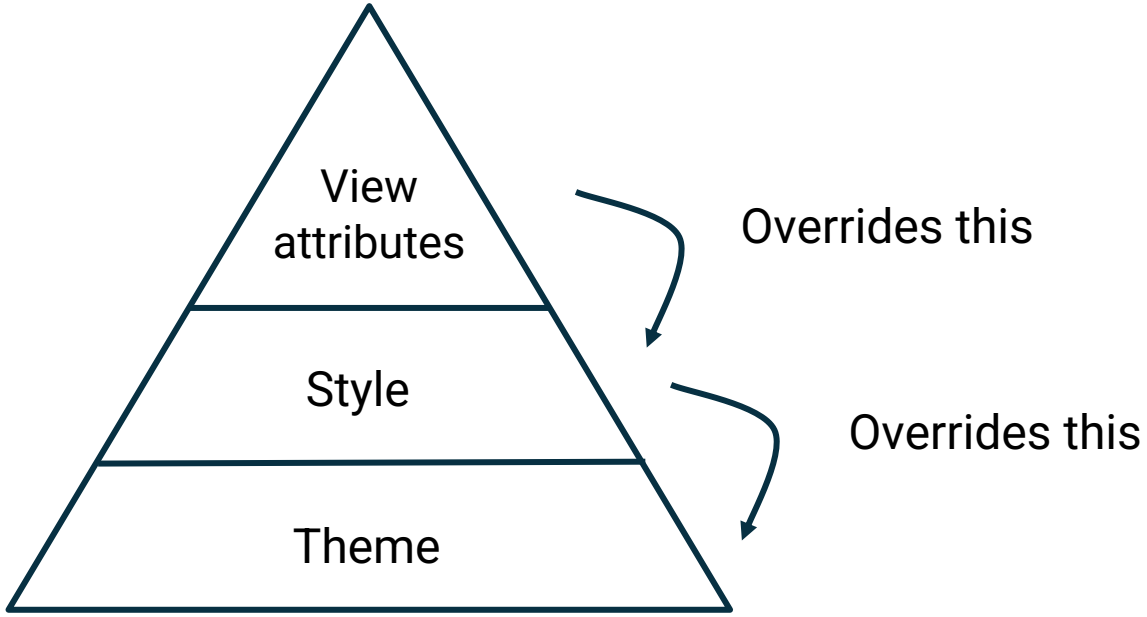
- Android styling
- Typography
- Material Design
- Material Components
- Localization
- Example apps
- Summary

Android styling

Android styling system

- Used to specify the visual design of your app
- Helps you maintain a consistent look across your app
- Hierarchical (you can inherit from parent styles and override specific attributes)

Precedence of each method of styling



Themes

- Collection of named resources, useful broadly across the app
- Named resources are known as theme attributes
- Examples:
  - Use a theme to define primary & secondary colors in the app
  - Use a theme to set the default font for all text within an activity

Declare a theme

In `res/values/themes.xml`:

```
<style name="Theme.MyApp" parent="Theme.MaterialComponents.Light">
    <item name="colorPrimary">@color/orange_500</item>
    <item name="colorPrimaryVariant">@color/orange_700</item>
    <item name="colorSecondary">@color/pink_200</item>
    <item name="colorSecondaryVariant">@color/pink_700</item>
    ...
</style>
```

Apply a theme

In `AndroidManifest.xml`:

```
<manifest ... >
    <application ... >
        <activity android:theme="@style/Theme.MyApp" ... >
        </activity>
    </application>
</manifest>
```

In layout file:

```
<ConstraintLayout …
    android:theme="@style/Theme.MyApp">
```

Refer to theme attribute in a layout

In layout file:

```
<LinearLayout …
    android:background="?attr/colorSurface">
```

Use `?attr/themeAttributeName` syntax.

Examples: `?attr/colorPrimary`
`?attr/colorPrimaryVariant`

Styles

- A style is a collection of view attributes, specific to a type of view
- Use a style to create a collection of reusable styling information, such as font size or colors
- Good for declaring small sets of common designs used throughout your app

Declare a style

In `res/values/styles.xml`:

```
<style name="DescriptionStyle">
    <item name="android:textColor">#00FF00</item>
    <item name="android:textSize">16sp</item>
    ...
</style>
```

Apply a style

On a view in a layout file:

```
<TextView
    style="@style/DescriptionStyle"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"

    android:text="@string/description_text" />
```

Refer to theme attribute in a style

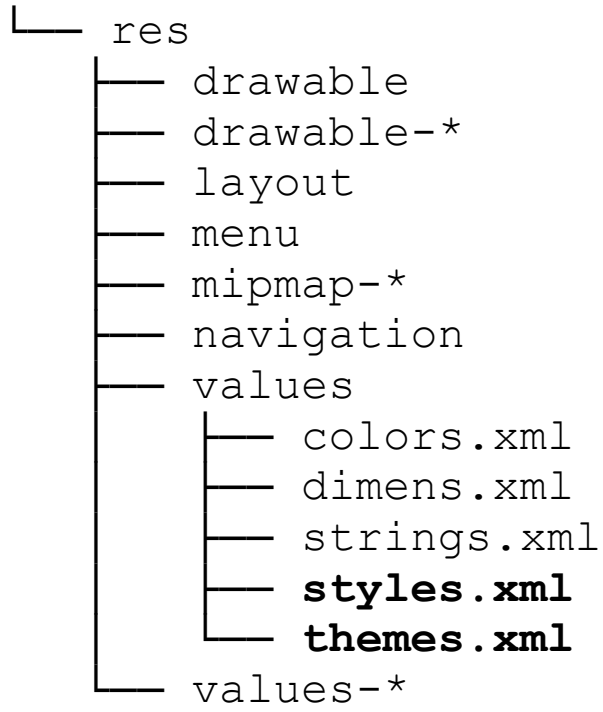
In `res/values/styles.xml`:

```
<style name="DescriptionStyle">
    <item name="android:textColor">?attr/colorOnSurface</item>
    <item name="android:textSize">16sp</item>
    ...
</style>
```

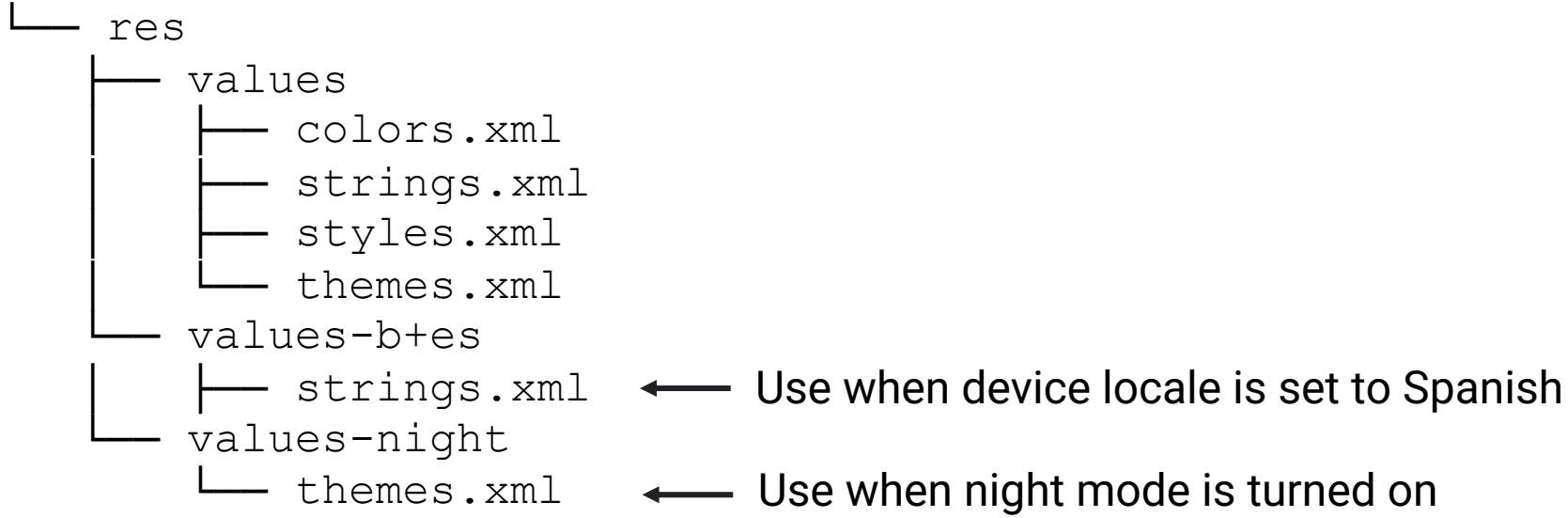
View attributes

- Use view attributes to set attributes explicitly for each view
- You can use every property that can be set via styles or themes
- Use for custom or one-off designs such as margins, paddings, or constraints

Resources directory



Provide alternative resources



Color resources

A way to name and standardize colors throughout your app

In `res/values/colors.xml`:

```
<resources>
    <color name="purple_200">#FFBB86FC</color>
    <color name="purple_500">#FF6200EE</color>
    <color name="purple_700">#FF3700B3</color>
    <color name="teal_200">#FF03DAC5</color>
    <color name="teal_700">#FF018786</color>
    ...
</resources>
```

Specified as hexadecimal colors in form of `#AARRGGBB`

Dimension resources

A way to name and standardize dimension values in your layouts

- Declare your dimension values in `res/values/dimens.xml`:

```
<resources>
    <dimen name="top_margin">16dp</dimen>
</resources>
```

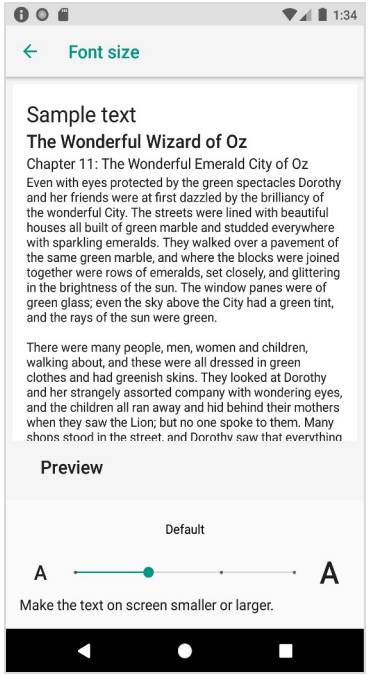
- Refer to them as `@dimen/<name>` in layouts or `R.dimen.<name>` in code:

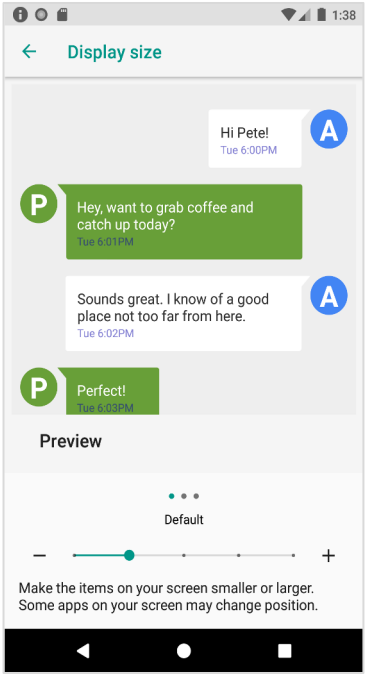
```
<TextView …
    android:layout_marginTop="@dimen/top_margin" />
```

Typography

Scale-independent pixels (sp)

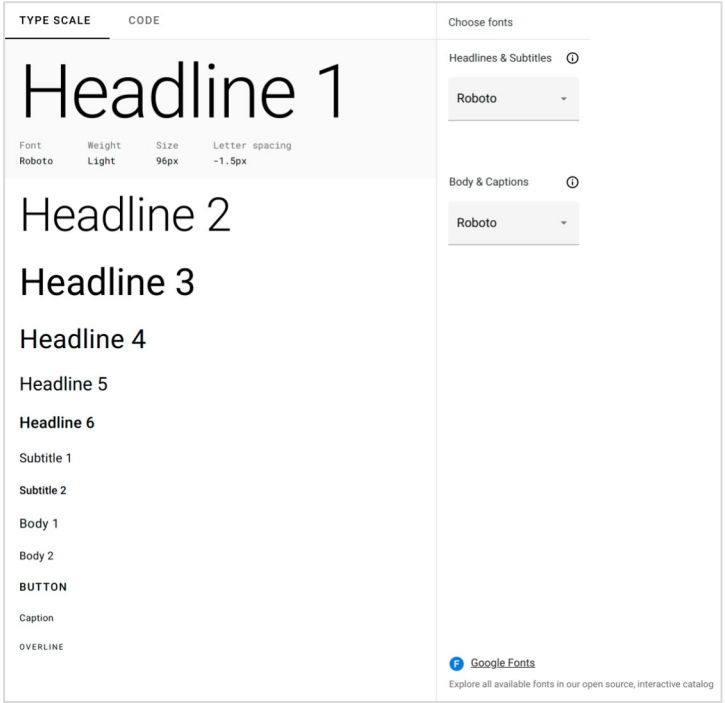
- The textual equivalent to density-independent pixels (dp)
- Specify text sizes in sp
  (takes into account user preferences)
- Users can adjust Font and Display sizes in the Settings app (after Display)





Type scale

- A set of styles designed to work together in a cohesive manner for your app and content
- Contains reusable categories of text with intended purpose for each (for example, headline, subtitle, caption)



TextAppearance

A `TextAppearance` style often alters one or more of these attributes:

- typeface (`android:fontFamily`)
- weight (`android:textStyle`)
- text size (`android:textSize`)
- capitalization (`android:textAllCaps`)
- letter spacing (`android:letterSpacing`)

Examples using TextAppearance

```
<TextView
    ...
    android:textAppearance="@style/TextAppearance.MaterialComponents.Headline1"
    android:text="@string/title" />


<TextView
    ...
    android:textAppearance="@style/TextAppearance.MaterialComponents.Body1"
    android:text="@string/body_text" />
```

Customize your own TextAppearance

```
<style name="TextAppearance.MyApp.Headline1"
        parent="TextAppearance.MaterialComponents.Headline1">
    ...
    <item name="android:textStyle">normal</item>
    <item name="android:textAllCaps">false</item>
    <item name="android:textSize">64sp</item>
    <item name="android:letterSpacing">0</item>
    ...
</style>
```

Use a custom TextAppearance in a theme

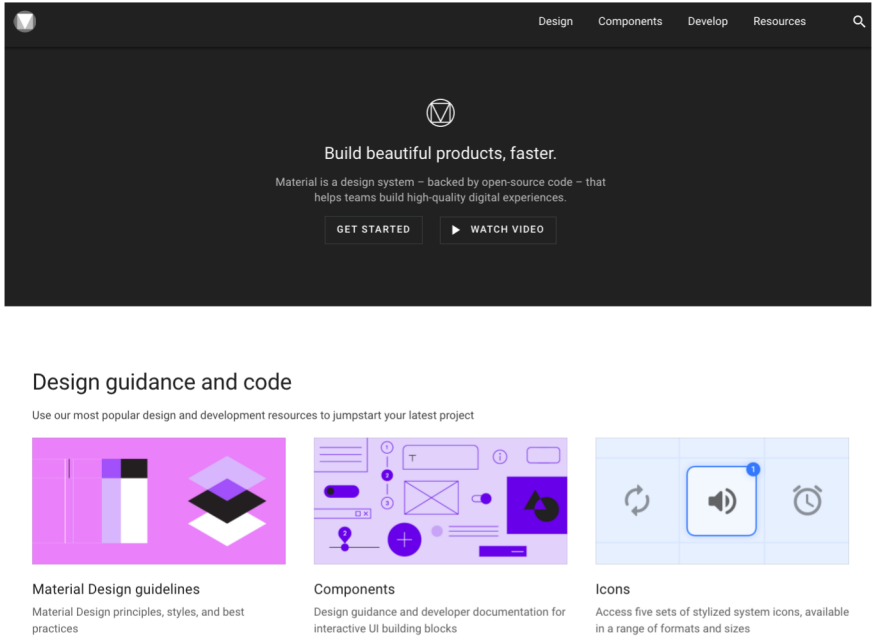
```
<style name="Theme.MyApp" parent="Theme.MaterialComponents.Light">
    ...
    <item name="textAppearanceHeadline1">@style/TextAppearance.MyApp.Headline1</item>
    ...
</style>
```

Material Design

Intro to Material

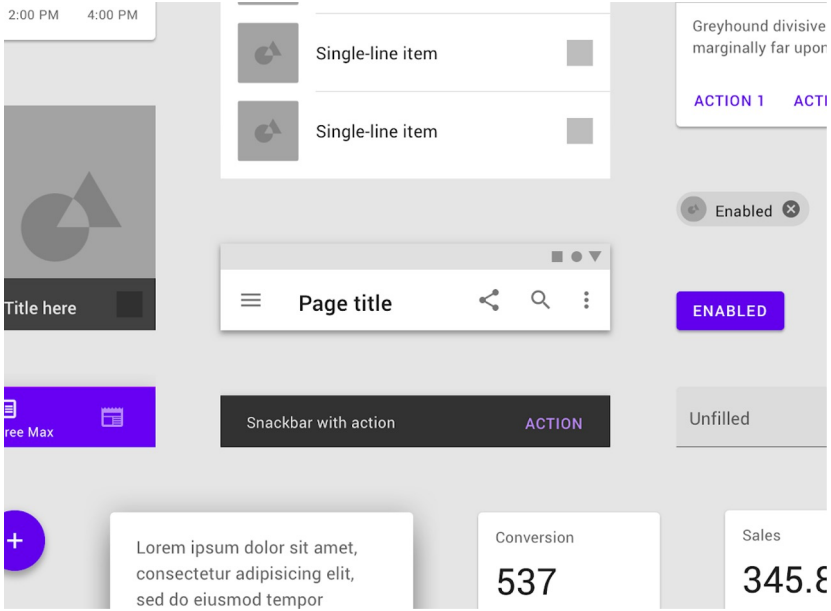
Adaptable system of guidelines, components, and tools that support best practices for UI design

[Material Design homepage]

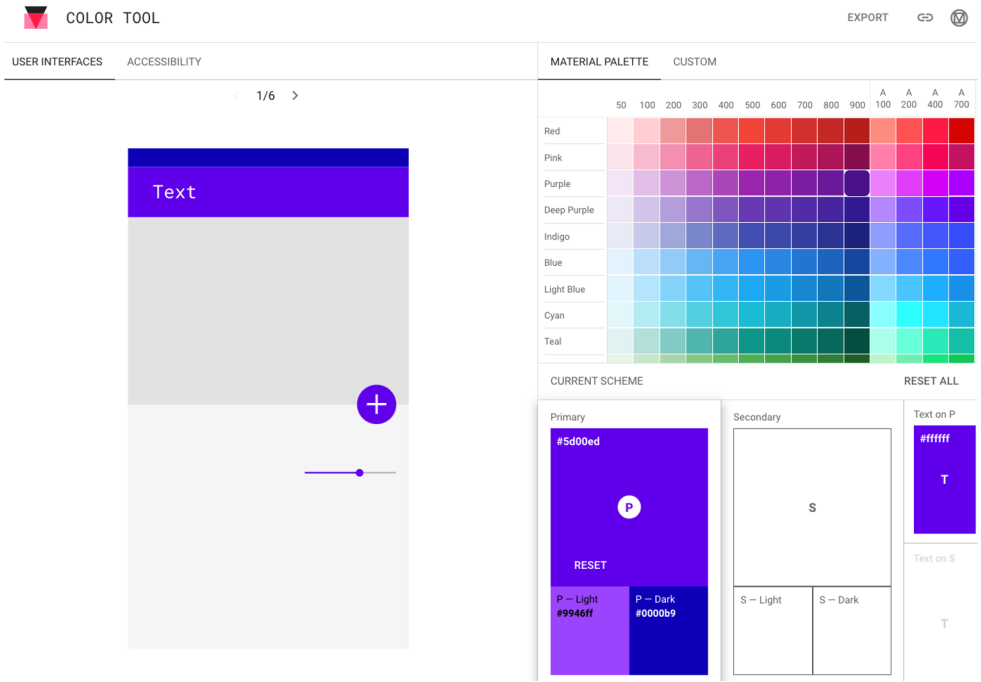


Material Components

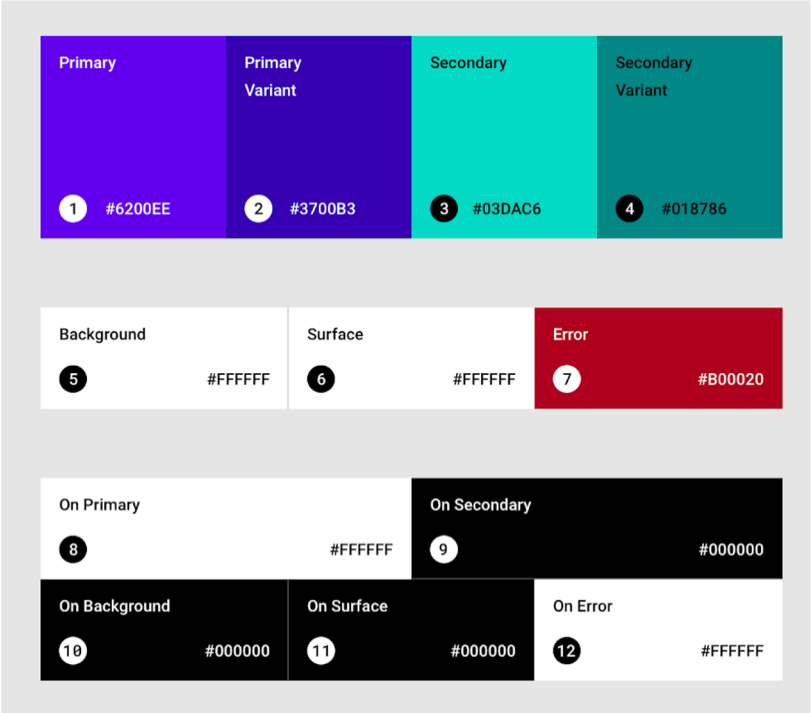
Interactive building blocks
for creating a user interface



Material color tool



Baseline Material color theme



Material Components for Android Library

```
implementation 'com.google.android.material:material:<version>'
```

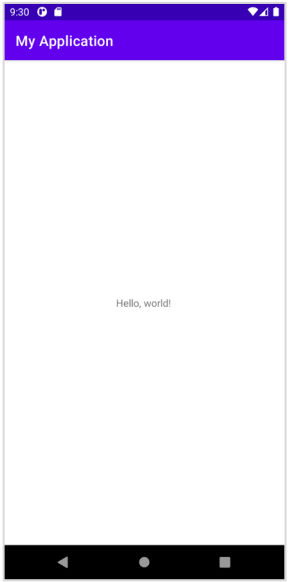
Material Themes

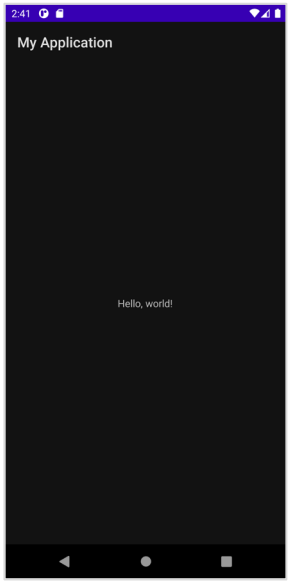
- `Theme.MaterialComponents`
- `Theme.MaterialComponents.NoActionBar`
- `Theme.MaterialComponents.Light`
- `Theme.MaterialComponents.Light.NoActionBar`
- `Theme.MaterialComponents.Light.DarkActionBar`
- `Theme.MaterialComponents.DayNight`
- `Theme.MaterialComponents.DayNight.NoActionBar`
- `Theme.MaterialComponents.DayNight.DarkActionBar`

Material theme example

`Theme.MaterialComponents.DayNight.DarkActionBar`

Light mode



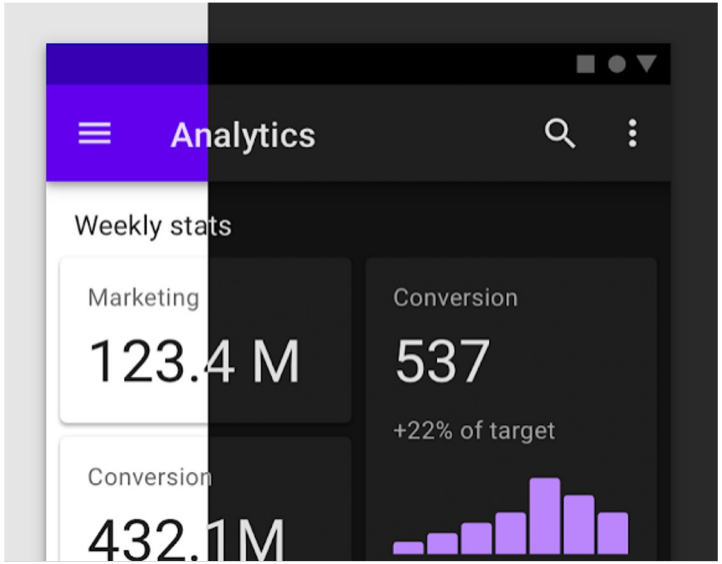


Dark mode

Dark theme

A low-light UI that displays mostly dark surfaces

- Replaces light-tinted surfaces and dark text with dark-tinted surfaces and light text
- Makes it easier for anyone to use a device in lower-light environments
- Improves visibility for users with low vision and those sensitive to bright light
- Can significantly reduce power usage (depending on the device)



Support dark theme

In `values/themes.xml`:

```
<style name="AppTheme" parent="Theme.MaterialComponents.DayNight">
    <item name="colorPrimary">@color/orange_500</item>
    ...
```

In `values-night/themes.xml`:

```
<style name="AppTheme" parent="Theme.MaterialComponents.DayNight">
    <item name="colorPrimary">@color/orange_200</item>
    ...
```

Material Components

Material Components

Component library provided for Android and design guidelines

- Text fields
- Buttons
- Menus
- Cards
- Chips
- App bars (top and bottom)
- Floating Action Button (FAB)
- Navigation Drawer
- Bottom navigation
- Snackbar

...and more!

Text field

- Composed of `TextInputLayout` with child view `TextInputEditText`
- Shows a floating label or a text hint before the user enters text
- Two types:

Filled text field

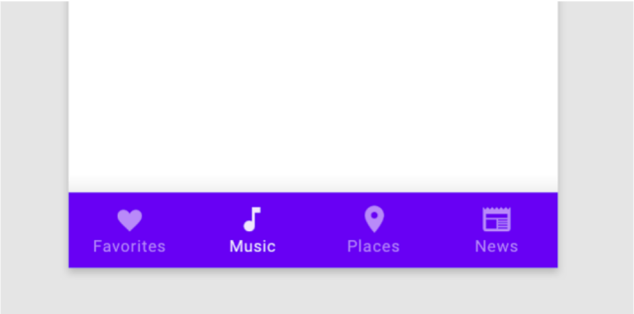
Outlined text field

Text field example

```
<com.google.android.material.textfield.TextInputLayout
    android:id="@+id/textField"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="@string/label"
    style="@style/Widget.MaterialComponents.TextInputLayout.OutlinedBox">

    <com.google.android.material.textfield.TextInputEditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

</com.google.android.material.textfield.TextInputLayout>
```

Bottom navigation

- Allows movement between top level destinations in your app
- Alternate design pattern to a navigation drawer
- Limited to 5 locations max



Bottom navigation example

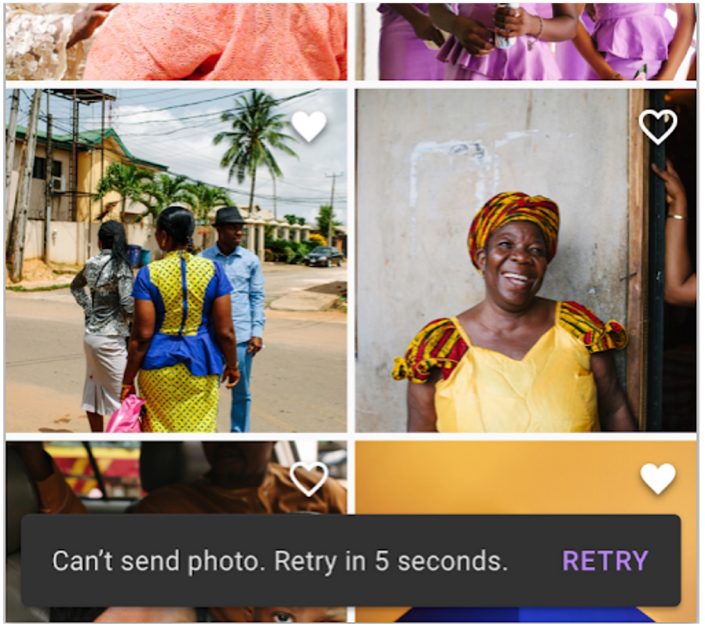
```
<LinearLayout …>

    ...

    <com.google.android.material.bottomnavigation.BottomNavigationView
        android:id="@+id/bottom_navigation"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:menu="@menu/bottom_navigation_menu" />

</LinearLayout>
```

Bottom navigation listener

```
bottomNav.setOnNavigationItemSelectedListener { item ->
    when(item.itemId) {
        R.id.item1 -> {
            // Respond to navigation item 1 click
            true
        }
        R.id.item2 -> {
            true
        }
        else -> false
    }
}
```

Snackbar

- Display short messages within the app
- Messages have a duration (`SHORT`, `LONG`, or `INDEFINITE`)
- May contain an optional action
- Works best in a `CoordinatorLayout`
- Shown at bottom of enclosing container



Snackbar examples

Show a simple message:

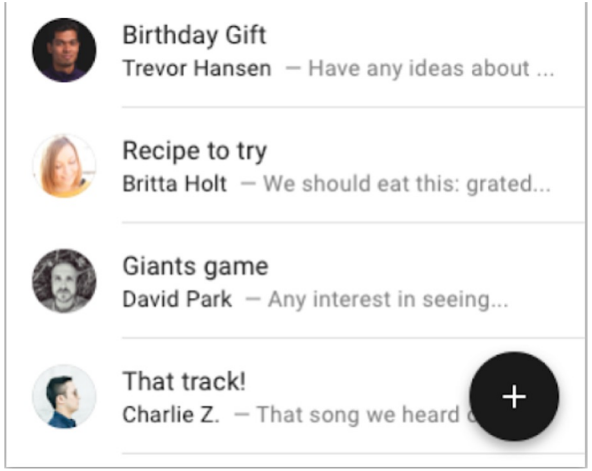
```
Snackbar.make(view, R.string.text_label, Snackbar.LENGTH_SHORT)
    .show()
```

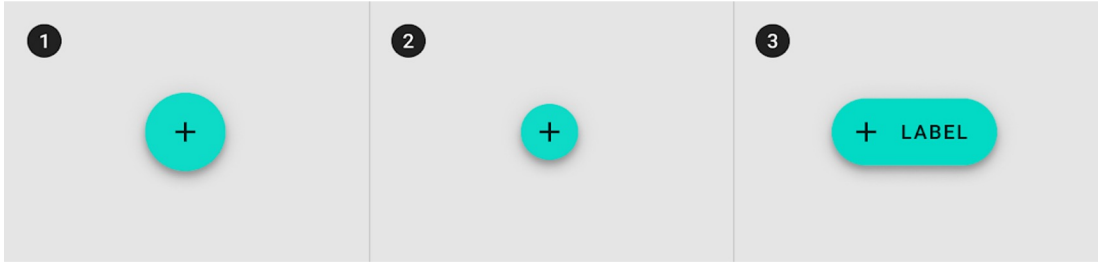
Add an action to a Snackbar:

```
Snackbar.make(view, R.string.text_label, Snackbar.LENGTH_LONG)
    .setAction(R.string.action_text) {
        // Responds to click on the action
    }
    .show()
```

Floating Action Button (FAB)

- Perform the most-common action for a screen (for example, creating a new email)
- Come in multiple sizes (regular, mini, and extended)





CoordinatorLayout

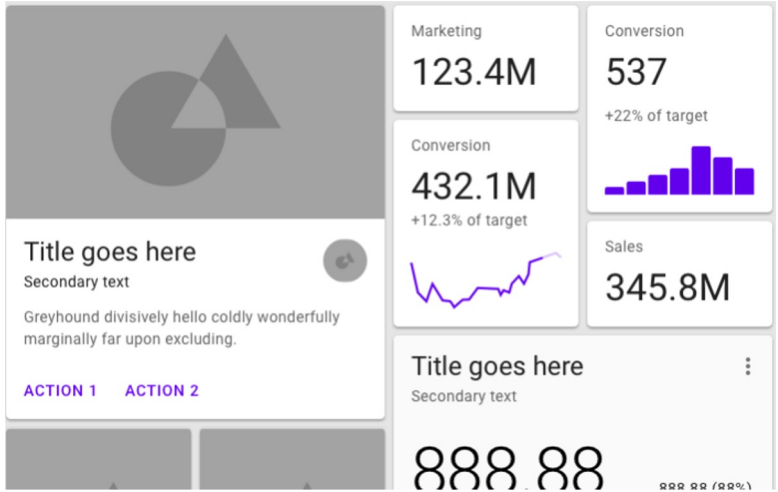
- Acts as top-level container in an app
- Manages interaction of its child views, such as gestures
- Recommended for use with views like a Snackbar or FAB

FAB example

```
<androidx.coordinatorlayout.widget.CoordinatorLayout ...>
    ....
    <com.google.android.material.floatingactionbutton.FloatingActionButton
        android:id="@+id/floating_action_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="bottom|end"
        android:layout_margin="16dp"
        android:contentDescription="@string/fab_content_desc"
        app:fabSize="normal" <!-- or mini or auto -->
        app:srcCompat="@drawable/ic_plus"/>

</androidx.coordinatorlayout.widget.CoordinatorLayout>
```

Cards

- A card holds content and actions for a single item.
- Cards are often arranged in a list, grid, or dashboard.
- Use `MaterialCardView`.



MaterialCardView example

```
<com.google.android.material.card.MaterialCardView
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="8dp">

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical">
        <ImageView .../>
        <TextView .../>
    </LinearLayout>

</com.google.android.material.card.MaterialCardView>
```

Localization

Localize your app

- Separate the localized aspects of your app (for example, text, audio files, currency, and numbers) as much as possible from the core Kotlin functionality of the app.

  Example: Extract the user facing strings into `strings.xml`.

- When a user runs your app, the Android system selects which resources to load based on the device's locale.
- If locale-specific resources are not found, Android falls back to default resources you defined.

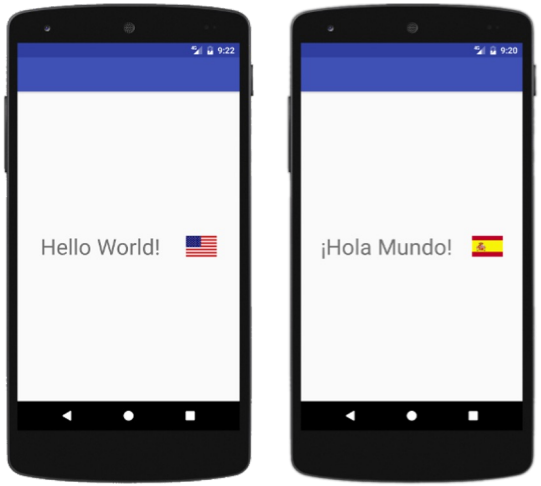
Support different languages and cultures

- Decide which locales to support.
- Create locale-specific directories in `res` directory:

  `<resource type>-b+<language code>`
  `[+<country code>]`

  Examples: `layout-b+en+US`
  `values-b+es`

- Provide locale-specific resources (such as strings and drawables) in those directories.



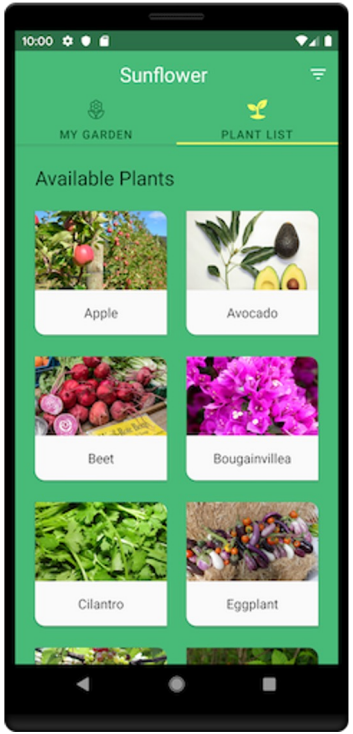
Support languages that use RTL scripts

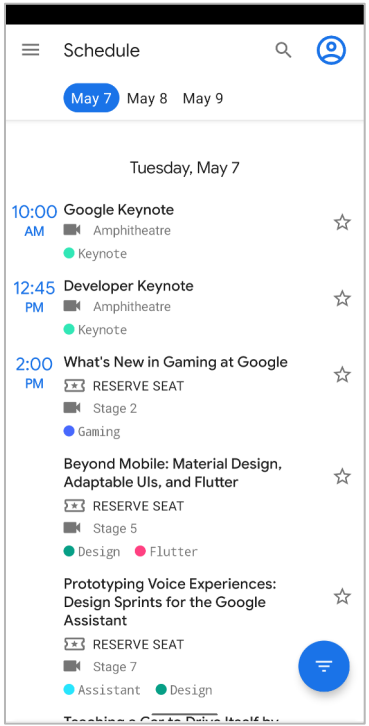
- Users can choose a language that uses right-to-left (RTL) scripts.
- Add `android:supportsRtl="true"` to app tag in manifest.
- Convert `left` and `right` to `start` and `end`, respectively, in your layout files (change `android:paddingLeft` to `android:paddingStart`).
- Localize strings and format text in messages.
- Optionally, use `-ldrtl` resource qualifier to provide alternate resources.

Example apps

Check out other apps

Sunflower app





Google I/O app

Summary

Summary

In Lesson 13, you learned how to:

- Customize the visual look of your app using styles and themes
- Choose from predefined type scales for the text in your app (or create your own text appearance)
- Select theme colors for your app using Material color tool
- Use Material Components library to speed up UI development
- Localize your app to support different languages and cultures

Learn more

- Material Design
- Material Components
- Tools for picking colors
- Dark theme
- Localize your app
- Blog posts: Themes vs Styles, Common Theme Attributes, Prefer Theme Attributes, Themes Overlay
- Sample code: Sunflower app, Google I/O app, Android GitHub repo

Pathway

Practice what you've learned by completing the pathway:

Lesson 13: App UI Design