User Interface

Fundamentals

- The **layout** defines the structure for an app's user interface.
- All elements in the layout are built using:
  - `View` (widget) – draws something the user can see and interact with
  - `ViewGroup` (layout) – an invisible container that defines the layout structure for `View` and other `ViewGroup` objects
- Two (2) ways to create a layout:
  - **Declare UI elements in XML**. The presentation of the app can be separated from the code that controls its behavior.
  - **Instantiate layout elements at runtime**. The `View` and `ViewGroup` objects can be created and their properties can be manipulated programmatically.
- To create a view/widget:
  1. Define a view/widget in the layout file and assign it a unique ID.

```
<Button android:id="@+id/my_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/my_button_text"/>
```

  2. Create an instance of the view object and capture it from the layout (typically in the `onCreate()` method)

```
Button myButton = (Button) findViewById(R.id.my_button);
```

- To manipulate the property of a widget in Java code:

```
TextView myText = new TextView(this);
myText.setText("Display this text!");
//The this keyword is a context.
```

- **Context** is an interface to global information about an application environment. To get the context:

```
Context context = getApplicationContext();
```

- All view groups include a **width** (`layout_width`) and **height** (`layout_height`), and each view is required to define them.
- Although the width and height can be specified with exact measurements, these two (2) constants are more often used:
  - `wrap_content` – sets the size of the view to the dimensions required by its content.
  - `match_parent` – sets the view to be as big as its parent view group will allow.
- The most common layout types are the following:
  - **Constraint Layout** – creates large and complex layouts with a flat view hierarchy (no nested view groups).
  - **Linear Layout** – organizes its child view elements into a single horizontal or vertical row.
  - **Relative Layout** – is used to specify the location of child objects relative to each other (child A to the left of child B) or to the parent (aligned to the top of the parent).
  - **Web View** – is used for displaying web pages.
  - **Frame Layout** – is designed to block out an area on the screen to display a single item.
  - **Table Layout** - arranges its child objects into rows and columns.
  - **Grid Layout** - arranges its child objects in a rectangular grid that can be scrolled.
- To create a layout in XML:

```
<LinearLayout
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
     <Button
       ... />
     <TextView
       ... />
     <Button
       ... />
</LinearLayout>
```

- To create a layout in Java code:

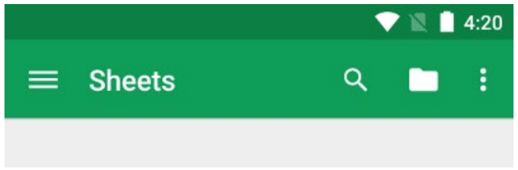
```
LinearLayout linearL = new LinearLayout(this);
linearL.setOrientation(LinearLayout.VERTICAL);
TextView myText = new TextView(this);
myText.setText("Display this text!");
linearL.addView(myText);
setContentView(linearL);
```

User Interface Components

- A **notification** is a message that Android displays outside the app's UI to provide the user with reminders, communication from other people, or other timely information from the app.

```
NotificationCompat.Builder builder = new
NotificationCompat.Builder(this, CHANNEL_ID)
        .setSmallIcon(R.drawable.notification_icon)
        .setContentTitle(textTitle)
        .setContentText(textContent)
        .setPriority(NotificationCompat.PRIORITY_DEFAULT);
```

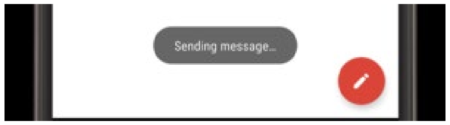
- The **notification drawer** allows to view more details and take actions with the notification.
- The **app bar/action bar** provides a visual structure and interactive elements that are familiar to users.



Its key features are:
  - A dedicated space for giving the app an identity and indicating the user's location in the app
  - Access to important actions in a predictable way, such as search
  - Support for navigation and view switching (with tabs or drop-down lists)

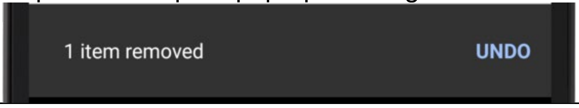
```
Toolbar myToolbar = (Toolbar) findViewById(R.id.my_toolbar);
  setSupportActionBar(myToolbar);
```

- A **toast** provides simple feedback about an operation in a small popup.



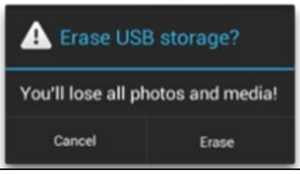
```
Context context = getApplicationContext();
CharSequence text = "Hello toast!";
int duration = Toast.LENGTH_SHORT;
Toast toast = Toast.makeText(context, text, duration);
toast.show();
//or Toast.makeText(context, text, duration).show();
```

- A ***Snackbar*** provides a quick pop-up message to the user.



```
Snackbar mySnackbar = Snackbar.make(view, stringId, duration);
mySnackbar.show();
//or Snackbar.make(view, stringId, duration).show();
```

- A **dialog** is a small window that prompts the user to make a decision or enter additional information. It is not designed to fill the screen.



```
AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
builder.setMessage(R.string.dialog_message)
     .setTitle(R.string.dialog_title);
AlertDialog dialog = builder.create();
```

- A **menu** is used to present user actions and other options in the app's activities.

```
<menu
xmlns:android="http://schemas.android.com/apk/res/android">
  <item android:id="@+id/new_game"
      android:icon="@drawable/ic_new_game"
      android:title="@string/new_game"
      android:showAsAction="ifRoom"/>
  <item android:id="@+id/help"
      android:icon="@drawable/ic_help"
      android:title="@string/help" />

</menu>
```

References:
DiMarzio, J. (2017). *Beginning Android programming with Android Studio*. Indiana: John Wiley & Sons, Inc.
Google Developers Training Team. (2018). *Android developer fundamentals (version 2)*. Retrieved from https://google-developer-training.github.io