Layout

This lesson shows how to work with Android layout.

WE'LL COVER THE FOLLOWING

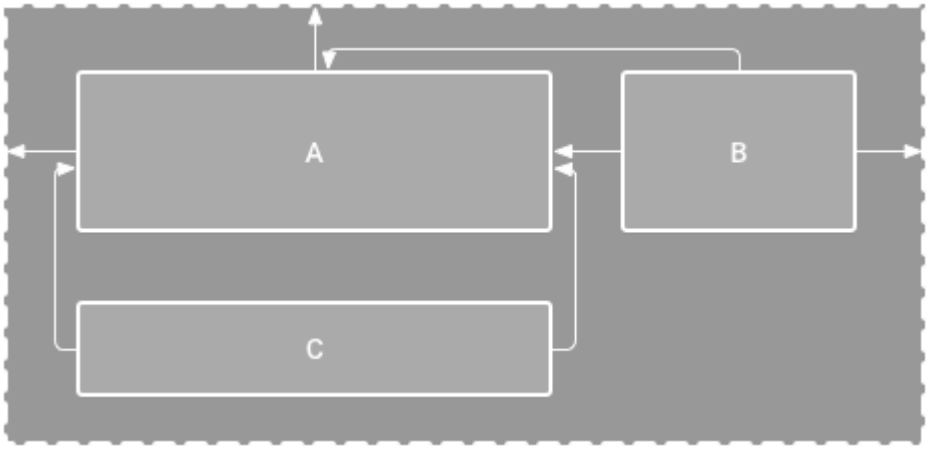
Layout concept

A layout defines the structure of the user interface. Layouts are built via views and view groups.

Views, sometimes also called “widgets,” represent interactable components such as:

- `TextView` - component to render text
- `EditText` - input field component where user can type text
- `Button` - clickable text component with background

ViewGroups, sometimes also called “layouts,” represent invisible containers which define the position of its children on the screen. While Android SDK contains a number of view groups, which still can be used, Google not so long ago released a new view group called `ConstraintLayout`.



Visual representation of constrains

This layout comes as a separate library and uses constraints to position views on the screen. The `ConstraintLayout` is more complicated than Android SDK view groups, but it has a rich visual editor to help build the user interface and in most cases has better performance.

While we can create a layout in Java code, it's easier to build a layout in the XML file and then inflate *(bind)* this layout to a specific activity.

Building layout

Let's try to create a layout with the "Hello World" text in the middle of the screen.

First, create a new *activity_main.xml* layout file inside *app/src/main/res/layout* folder. As a root layout we are going to use `ConstraintLayout` along with some XML attributes:

- `layout_width="match_parent"` attribute defines the width of the layout, in our case we want to take all the width of the screen
- `android:layout_height="match_parent"` attribute defines the height of the layout, in our case we want to take all the height of the screen
- `xmlns:android` and `xmlns:app` attributes defines XML namespace, *android* namespace for attributes from Android SDK and *app* namespace for attributes from libraries

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
```

```
        xmlns:app="http://schemas.android.com/apk/res-auto"
        android:layout_width="match_parent"
        android:layout_height="match_parent">


</androidx.constraintlayout.widget.ConstraintLayout>
```

activity_main.xml

Right now, our layout is empty, let's define a child view with "Hello World" text. To display some static text, a `TextView` can be used along with the `text` attribute. Instead of using `match_parent` for `layout_width` and `layout_height`, we are going to use `wrap_content` in order for the view to take as much space as required.

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:app="http://schemas.android.com/apk/res-auto"
        android:layout_width="match_parent"
        android:layout_height="match_parent">

    <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Hello World!" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

activity_main.xml

Note: For views that are available in Android SDK, we don't need to specify the full package name *(e.g. TextView)*, while for views which are available in libraries, we do need to specify the full package name *(e.g. androidx.constraintlayout.widget.ConstraintLayout)*.

Preview of the layout.

Alignment

By default, our `TextView` is going to be positioned in the top left corner of the screen. To move it into the center, we need to add the following constraints:

- `layout_constraintTop_toTopOf` attribute declares a constraint to align the top of the view to the top of the `ConstraintLayout`
- `layout_constraintBottom_toBottomOf` attribute declares a constraint to align the bottom of the view to the bottom of the ConstraintLayout
- `layout_constraintLeft_toLeftOf` attribute declares a constraint to align the left of the view to the left of the ConstraintLayout
- `layout_constraintRight_toRightOf` attribute declares a constraint to align the right of the view to the right of the ConstraintLayout

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:app="http://schemas.android.com/apk/res-auto"
        android:layout_width="match_parent"
        android:layout_height="match_parent">

    <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
```

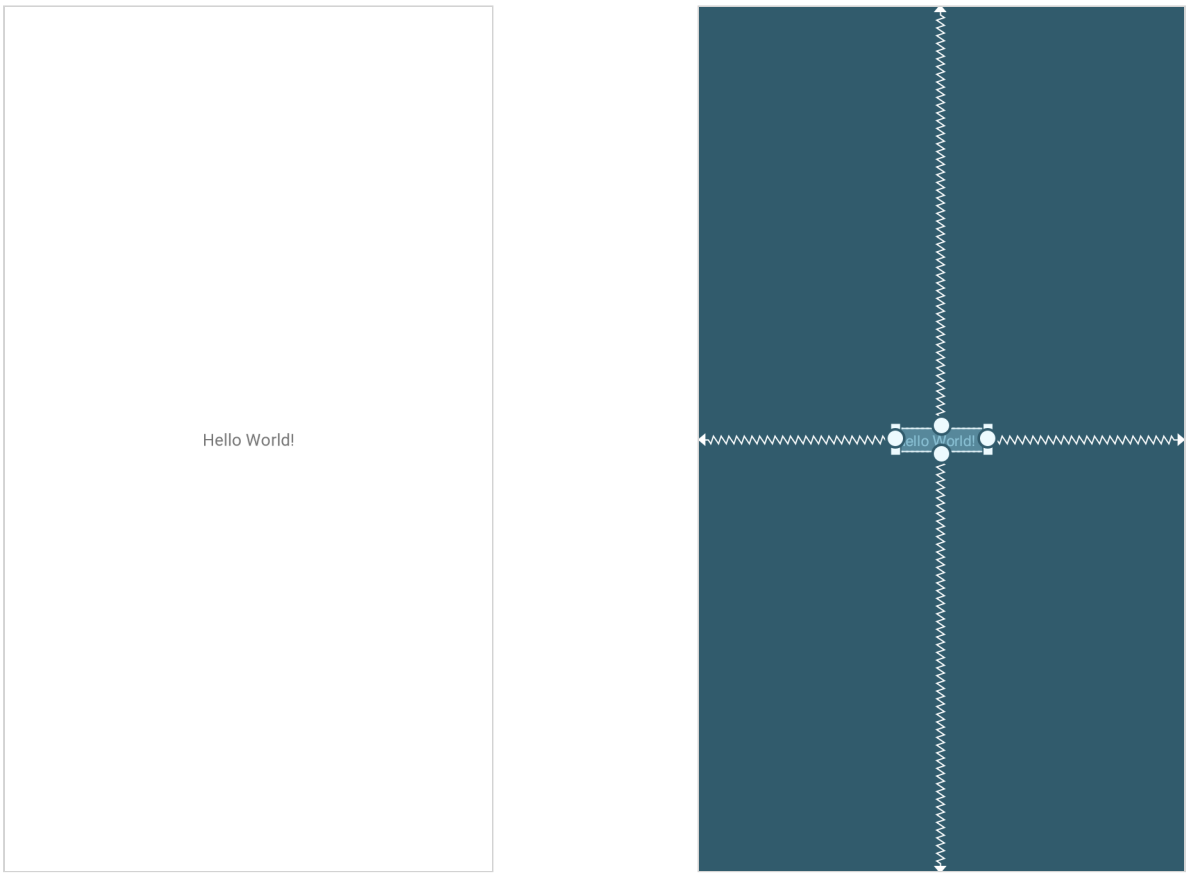
```
            android:text="Hello World!"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintLeft_toLeftOf="parent"

            app:layout_constraintRight_toRightOf="parent"
            app:layout_constraintTop_toTopOf="parent" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

activity_main.xml

Preview of the layout.



Layout binding

To associate *activity_main.xml* layout with *MainActivity*, we need to bind them via `setContentView` method. We can do that when activity is created inside the `onCreate` method.

A `setContentView` accepts layout resource ID, which can be referenced by special Android `R` class, which is auto-generated by the Android build system and contains resource IDs for all the resources of *app/src/main/res* directory.

In our case, to get the ID of *activity_main.xml* we can use `R.layout.activity_main`.

```
public class MainActivity extends AppCompatActivity {
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

MainActivity.java

Now, when *MainActivity* is launched, it's going to render layout from *activity_main.xml* file. Hit the *run* button to try it yourself.

```
package com.travelblog;

import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

View binding

In order to interact with views on runtime, we need to bind the view from XML to Java object, but before doing the binding, we must define a unique ID for the `TextView`.

To specify the ID for the `TextView`, we can use `id` attribute with `@+id/mainTextView` value, where `@+id/` indicates ID creation and `mainTextView` is the ID itself.

```
<TextView
        android:id="@+id/mainTextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello World!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
```

activity_main.xml

It's time to bind the `TextView` from XML file to Java object via `findViewById`

method and use `setText` method to change the text to 'Hello educative.io'.

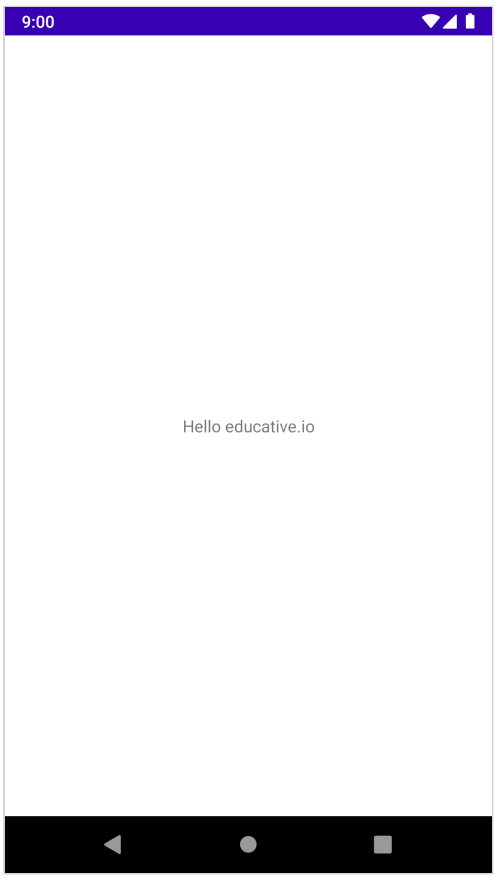
```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        TextView mainTextView = findViewById(R.id.mainTextView);
        mainTextView.setText("Hello educative.io");
    }
}
```

MainActivity.java

As you can see in the preview below, now when we launch the application `TextView` shows 'Hello educative.io' text.



Hit the *run* button to try it yourself.

```
package com.travelblog;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        TextView mainTextView = findViewById(R.id.mainTextView);
        mainTextView.setText("Hello educative.io");
    }
}
```

In the next lesson, we will give an overview of the application we will build throughout this course.