

CST2335
GRAPHICAL
INTERFACE
PROGRAMMING

Week 2

Using Widgets in Android

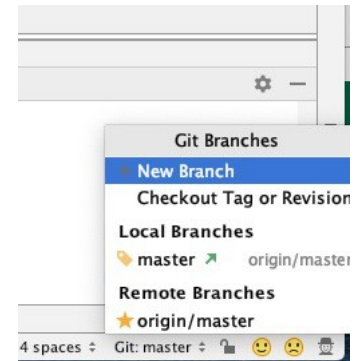
Agenda

- Making Branches in Git
- Using Text Widgets
- Lambda Functions
- ViewModel class
- Using Compound Widgets
- Image Widgets



Git branch

- A branch in Git is like an optional pathway for building your code.
- Click on the “Git: main” button at the bottom right of AndroidStudio.
- Click on New Branch and call it “NewOptions”.
- Go to the activity_main.xml layout and drag a button onto the screen. The filename turns blue because of uncommitted changes
- Video example: <http://bit.ly/CST2335W2S6>



Git commands

- Git commit -am “message”
- Git log - -oneline
- Git branch
- Git checkout

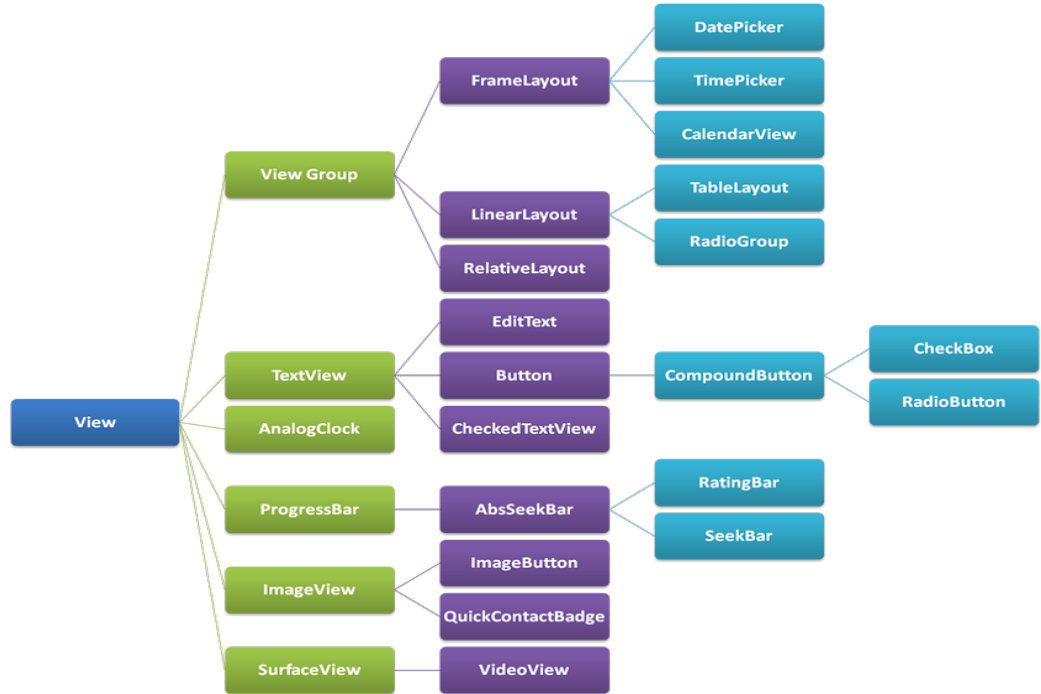


Views and View Groups

Every item in a user interface is a subclass of the Android View class (`android.view.View`)

Android SDK provides a set of pre-built views that can be used to construct a user interface such as the Button, CheckBox, ProgressBar and TextView classes

We refer to them as ***widgets*** or ***components***



Layouts

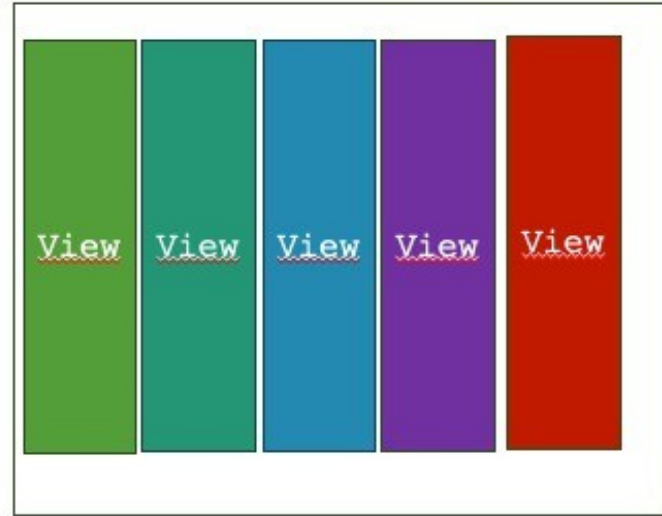
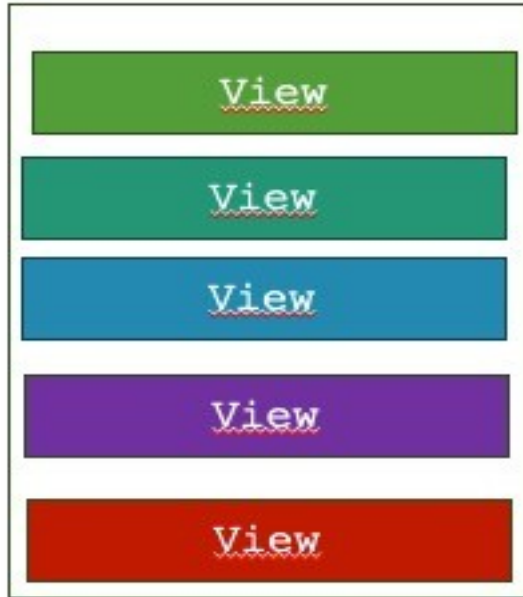
- Layouts are container views, designed to control how child views are positioned on the screen
- Each view in a user interface represents a rectangular area of the display.
- A view is responsible for what is drawn in that rectangle and for responding to events that occur within that part of the screen (such as a touch event).



Linear Layout

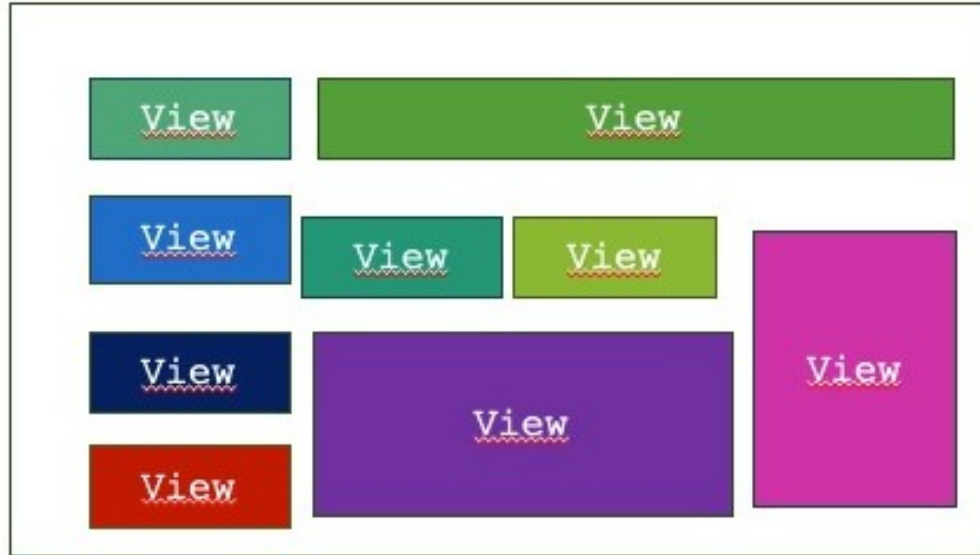
Positions child views in a single row or column depending on the orientation selected.

This is used for both horizontal and vertical directions and use the **android:orientation** parameter to tell the difference. It can be either “horizontal” or “vertical”. If the orientation is not specified, it is horizontal by default.



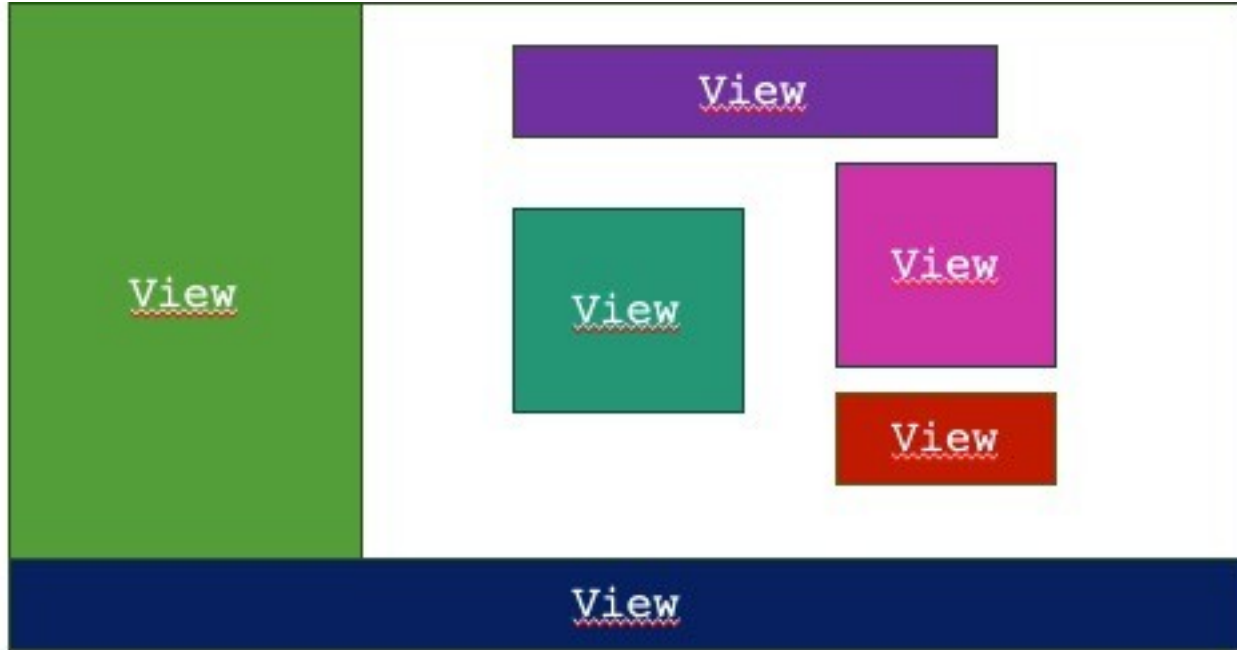
Grid Layout

View is divided by invisible lines that form a grid containing rows and columns of cells



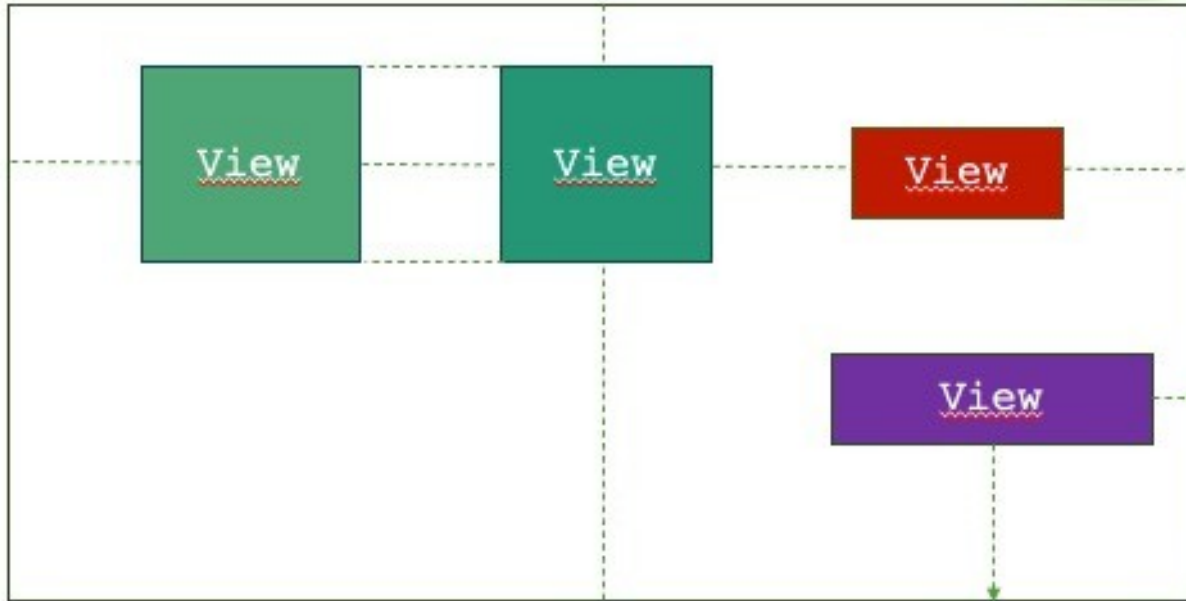
RelativeLayout

Allows child views to be positioned relative both to each other and the containing layout view through the specification of alignments and margins on child views.



ConstraintLayout

Introduced in Android 7, use of this layout manager is recommended for most layout requirements. It allows the positioning and behavior of the views in a layout to be defined by simple constraint settings assigned to each child view.



Layouts and Widgets

- All layouts and widgets must have a **height** and **width**. This information is defined in the **xmlns:android** parameter.
- layout_width & layout_height – this layout can either take the full size of the parent: “match_parent”, or just big enough to fit the object: “wrap_content”
- Layouts and Widgets should have ids: android:id="@+id/myId". IDs in android are used in the same way they are used in HTML: id="myId"



Widgets

- We will use Button, ImageButton, TextView, EditText, Checkbox, Switch, RadioButton, ListView and Spinner.
- All widgets must have `layout_width` and `layout_height` parameters set. You should also set the *id* parameter.
- We define the widgets in XML, and then to import them into Java, use the **findViewById()** method. It will return the Java object that has a matching ID to what is in the brackets.



TextView and EditText

- TextView is a widget that puts text on the screen.
- EditText is a widget that puts text on the screen, but the user can edit the text.
- **Step 1 XML:** Add <TextView> or <EditText> to your XML layout. Set the layout_width and layout_height parameters. Give your tag an id parameter:
android:id="@+id/myText"
- **Step 2 JAVA:** right after the setContentView() function, import the TextView from XML into Java:
TextView myTextView = findViewById(R.id.myText);
- This function will search all objects on the screen and return the one with id=myText. If there are no objects found, then it returns null.
- Video example: <http://bit.ly/CST2335W2S17>



Text and EditText

- In XML, use **android:text="@string/something"** to specify what words will show.
- EditText also has a hint, which is text that shows when the field is empty, but disappears when the user starts typing:
`android:hint="@string/hint_text"`
- In Java, use *setText()* to set the string that is shown, or use *getText()* to get what is in the EditText
- In Java, use *getResources().getString(R.string.hint_text)* to get the right version of a string for one of the languages you support.
- You can change the size of the text using the **android:textSize** parameter.



Text and EditText

- The `setText()` function also has a version that takes an **int** instead of a String. This version of `setText` takes the id of the string and will automatically translate the string to the right language:

setText(R.string.hello_world);

- For the `EditText`, you can also specify in XML which keyboard to show by using the **inputType=" .."** parameter. You can have a number-only keypad, an Integer only keypad without a decimal key, an email keyboard that has the "@" symbol, etc. Look at the various options offered by AndroidStudio.

Video Example: <http://bit.ly/CST2335W2S19>



Button

- In XML, buttons also have a text parameter. It specifies the text that will show on the button:

android:text="I'm a button"

android:id="@+id/button1"

- Step 1 XML:** To add a button, add a <Button> tag in your XML layout, and set the layout_width and layout_height parameters.
- Step 2 JAVA:** In Java, in the onCreate function, after setContentView():
Button btn = findViewById(R.id.button1)
- In general, the XML tag matches the Java class that you should import

Same!



Button

- Normally you want to run code when a button is clicked, so you need to listen for click events from the button:

```
btn.setOnClickListener( OnClickListener obj )
```

- If you look at the Android documentation, you will find that OnClickListener is an interface that has one function: <https://developer.android.com/reference/android/view/View.OnClickListener>
onClick(View v)

Video example: <http://bit.ly/CST2335W2S21>



View.OnClickListener

- You can write an anonymous class that implements the onClick function:

```
setOnClickListener( new View.OnClickListener()
{
    void onClick(View v){
        // This code gets run when the button is clicked
    }
})
```

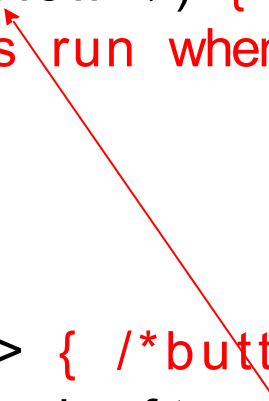
- When an interface has only 1 function, instead of writing an anonymous class, you can just write the one function without specifying the function name. The name is inferred from the interface.

Video Example: <http://bit.ly/CST2335W2S22>



Lambda functions

```
setOnClickListener( new View.OnClickListener()  
    { void onClick(View v) {  
      /* This code gets run when the button is clicked */  
    } })
```



Becomes →

```
setOnClickListener( v -> { /*button is clicked*/ } )
```

The compiler knows that **v** is of type **View** from the declaration of the `setOnClickListener(OnClickListener obj)` function.

Lambda functions requires Java 8. Ensure the settings are correct in Project Structure → Modules → Properties(tab) → Source Compatibility = \$VERSION_1_8 and Target Compatibility = \$VERSION_1_8



Exercise

Write a click handler that changes the text of the button to 'You clicked me!'

```
btn.setOnClickListener( v -> { v.setText("You clicked me");} );
```

If the Lambda function has only 1 line, then you don't need the { } braces around the function. You can just write this:

```
btn.setOnClickListener( v -> v.setText("You clicked me") );
```



ImageButton

- In XML, ImageButtons are a special kind of Button that have a jpeg or png on them instead of text.
- Use ***android:src="@drawable/picture_file"*** to specify what picture to show on the button.
- Add <ImageButton> to the layout and specify the src of what picture to show.
- In Java, find the ImageButton and add a click listener:
ImageButton iButton = findViewById(R.id.buttonId);
iButton.setOnClickListener(btn -> System.out.println("Click!"));

Video Example: <http://bit.ly/CST2335W2S27>



CheckBox, Switch and RadioButton

- The pattern is always the same for adding objects.

- In XML:

- `<CheckBox id="...">`

- `<SwitchCompat id="...">`

- `<RadioButton id=" ... ">`

In Java:

`CheckBox cb = findViewById(...);`

`SwitchCompat s = findViewById(...);`

`RadioButton r = findViewById(...);`

Same

```
graph LR; Same((Same)) --> XML1["<CheckBox id='...'>"]; Same --> XML2["<SwitchCompat id='...'>"]; Same --> XML3["<RadioButton id=' ... '>"]; Same --> Java1["CheckBox cb = findViewById( ... );"]; Same --> Java2["SwitchCompat s = findViewById( ... );"]; Same --> Java3["RadioButton r = findViewById( ... );"];
```



CheckBox, Switch and RadioButton

- These are all buttons that can be either on, or off. The only difference is how they look. In XML, use **android:checked="true" or "false"**
- In Java, use isChecked() or setChecked(Boolean) to set the value:
CheckBox cb = (CheckBox)findViewById(R.id.checkbox)
cb.setChecked(true);
*cb.setOnCheckedChangeListener(**OnCheckedChangeListener**)*
- OnCheckedChangeListener is an interface that has one function:
onCheckChanged(CompoundButton cb, boolean isChecked) { }
- Since there is only one function, you can use lambda functions:
- *setOnCheckedChangeListener((cb, **isChecked**) -> { });*



Adding Widgets

- The pattern is the same, no matter what widget you want to add. First, you add `<MyGuiObject id="anId">` in the XML.

- In Java, you then have to find that view:

```
MyGuiObject obj = findViewById(R.id.anId);
```

The Java class always matches the XML tag name (MyGuiObject in this case)

- Lastly, add some click listener depending on the object:
 - `setOnClickListener()` <- for buttons
 - `setOnCheckedChangeListener()` <- for checked buttons
 - `setOnKeyListener()` <- for text input



User messages

- We will learn how to show pop-up messages to the user.
- Normally you would do this when the user clicks on something.
- There are two main pop-up messages in Android:
 - Toast – a window that shows for a brief time in the bottom part of the screen.
 - Snackbar – a window that shows up for a brief time from the bottom of the screen

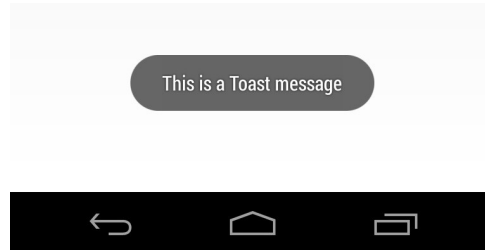


Toast

- A Toast is one line of code:

```
Toast.makeText(Context, "This is a Toast message", int duration).show();
```

- Context is the AppCompatActivity that is currently active (use **this** pointer). Message String is what shows up on the window. Duration is either Toast.LENGTH_SHORT or Toast.LENGTH_LONG. Try it out!



Summary

- Use XML to define your layout:
 - `<Text>` and `<EditText>` use ***android:text=""***, and ***android:hint=""***
 - Button uses ***android:text=""***, ImageButton uses ***android:src=""***
 - CheckBox, RadioButton and Switch are all two-state buttons. They use ***android:checked*** to specify if they are on or off..
- In Java:
 - After `setContentView()` , use the pattern:
Widget anObject = findViewById(R.id.objectId) ;
 - Text and EditText use `setText()` and `getText()`
 - Button and ImageButton use: `setOnClickListener(e -> { });`
 - CheckBox, RadioButton and Switch use:
`setOnCheckedChangeListener((btn, checked) -> { });`

