Android tutorial: Coffee ordering app

For who?

Students who take the *Software development* course at KU Leuven Groep T Campus

For what?

To get started with Android app development

To get to know data storage solutions provided by our campus

To be used in conjunction with explanation given in the lab sessions

What is included?

IDE set up guide

Introduction to Android studio, Android app development concepts and Event-driven paradigm

Step-by-step guide to create a coffee ordering app – Java bean and communicate with a web service

Small exercises to brush up your coding skill

What are the prerequisites?

Java programming language

Object oriented concept

Relational database concepts

Android Studio version?

Originally created for Android Studio version 3.5.2; updated for version 4.1.2 (March, 2021)

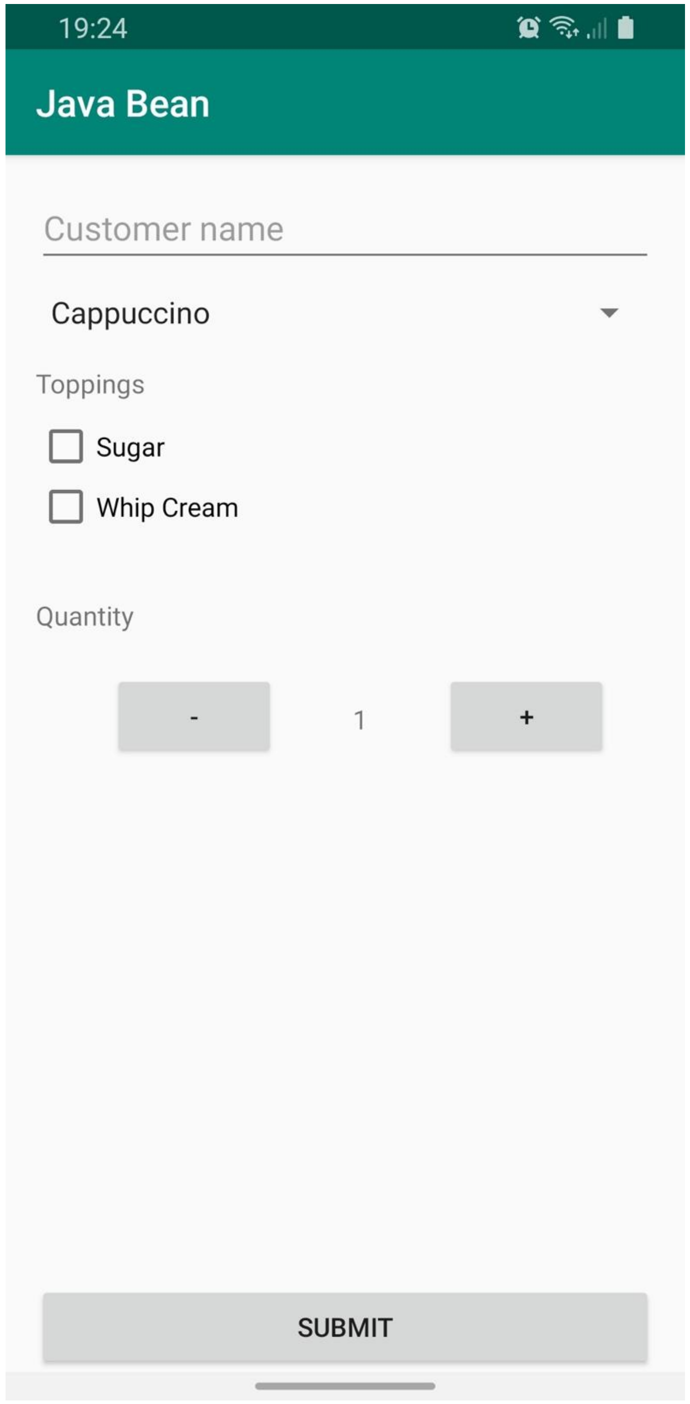


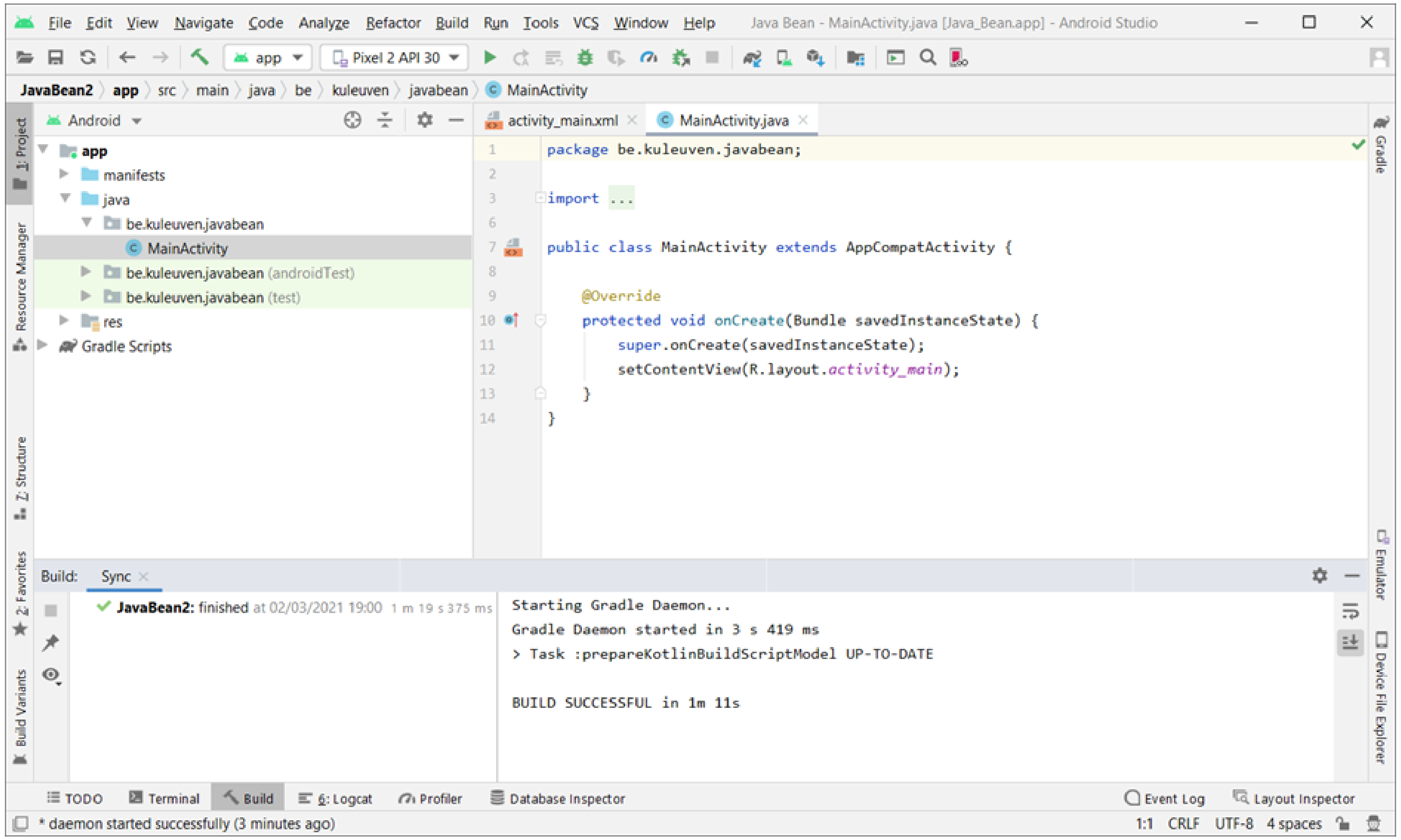
Table of Contents

Setting Up a Project

Android app development process can be done entirely on Android studio. Before a project can be set up, you need to download and install Android studio from https://developer.android.com/studio. Once installed, follow the steps below to set up your first project:

1. Open Android studio, and you will be welcomed with a wizard screen. Choose **Create New Project**
2. Select a Project Template and choose how the first screen of your app looks like. In this case, we will choose **Phone and Tablet** and **Empty Activity** as an example in this tutorial. Then click **Next**
3. On the **Configure Your Project** screen, you can set the name of your app and other information. In this tutorial, we will leave the settings with their default value but we set the following information:
   - Name: Java Bean
   - Package name: be.kuleuven.javabean
   - Language: Java
   - Minimum API level: API 23: Android 6.0 (Marshmallow) - to ensure that our app is compatible with 84.9% of all android devices. For more information about API level distribution click **help me choose**.

   Click **Finish**


File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
Java Bean - MainActivity.java [Java_Bean.app] - Android Studio
app
Pixel 2 API 30
JavaBean2 app src main java be kuleuven javabean MainActivity
Android
app
manifests
java
be.kuleuven.javabean
MainActivity
be.kuleuven.javabean (androidTest)
be.kuleuven.javabean (test)
res
Gradle Scripts
activity_main.xml
MainActivity.java
1: Project
Resource Manager
Z: Structure
2: Favorites
Build Variants
Gradle
Emulator
Device File Explorer
Build:
Sync
JavaBean2: finished at 02/03/2021 19:00 1 m 19 s 375 ms
TODO
Terminal
Build
6: Logcat
Profiler
Database Inspector
Event Log
Layout Inspector
daemon started successfully (3 minutes ago)
1:1 CRLF UTF-8 4 spaces

```java
package be.kuleuven.javabean;

import ...

public class MainActivity extends AppCompatActivity {

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

```
Starting Gradle Daemon...
Gradle Daemon started in 3 s 419 ms
> Task :prepareKotlinBuildScriptModel UP-TO-DATE

BUILD SUCCESSFUL in 1m 11s
```

4. Once you clicked **Finish,** the project setup wizard is completed. You will see a window as shown on the right.

Setting Up a Test Environment

The app that we are developing is not meant to be run on a computer. Hence, you may need an Android device to test/run your app. If you own an android device, you may skip the following section and go to Configuring an Android phone as a development device.

Setting up an Android emulator

To test/run Android apps on your computer without the need for a real Android device, you can set up an Android Virtual Device (AVD). However, the standard Android Studio AVD is far from optimal and can be very taxing on your computer. In this tutorial, we will use Genymotion which is a faster/more efficient AVD.

1. To install Genymotion, go to https://www.genymotion.com/fun-zone/, create an account, download and install '*Genymotion for Personal Edition*'. When the installer prompts you to select a folder to install, note down the installation path as we will need it in later stage.
2. Once installed, run Genymotion. When it prompts you to activate a license, log in with the account you've just created earlier. Create/install a virtual Android device by choosing the Google Pixel 2 template and choose Android 8.0 as the phone's operating system version.
3. To integrate Genymotion into Android studio, switch back to the Android studio project that we created earlier and follow the steps bellows:
   a) **For Windows/Linux:** Go to File/Settings; **For Mac:** go to Android Studio/Preferences.
   b) Select **Plugins** from the left navigation menu.
   c) Search for Genymotion from the list of plugins and install it.
   d) Restart your Android Studio.
   e) On Android studio, go to **View-Appearance** menu and enable **Toolbar**. You should now see the GenyMotion Logo in the toolbar.
   f) Now go to File/Settings/Other Settings/Genymotion and add the installation path of Genymotion you noted earlier.
   g) Try opening the Pixel 2 you made by pressing the Genymotion icon, selecting your device, and pressing start.

Configuring an Android phone as a development device

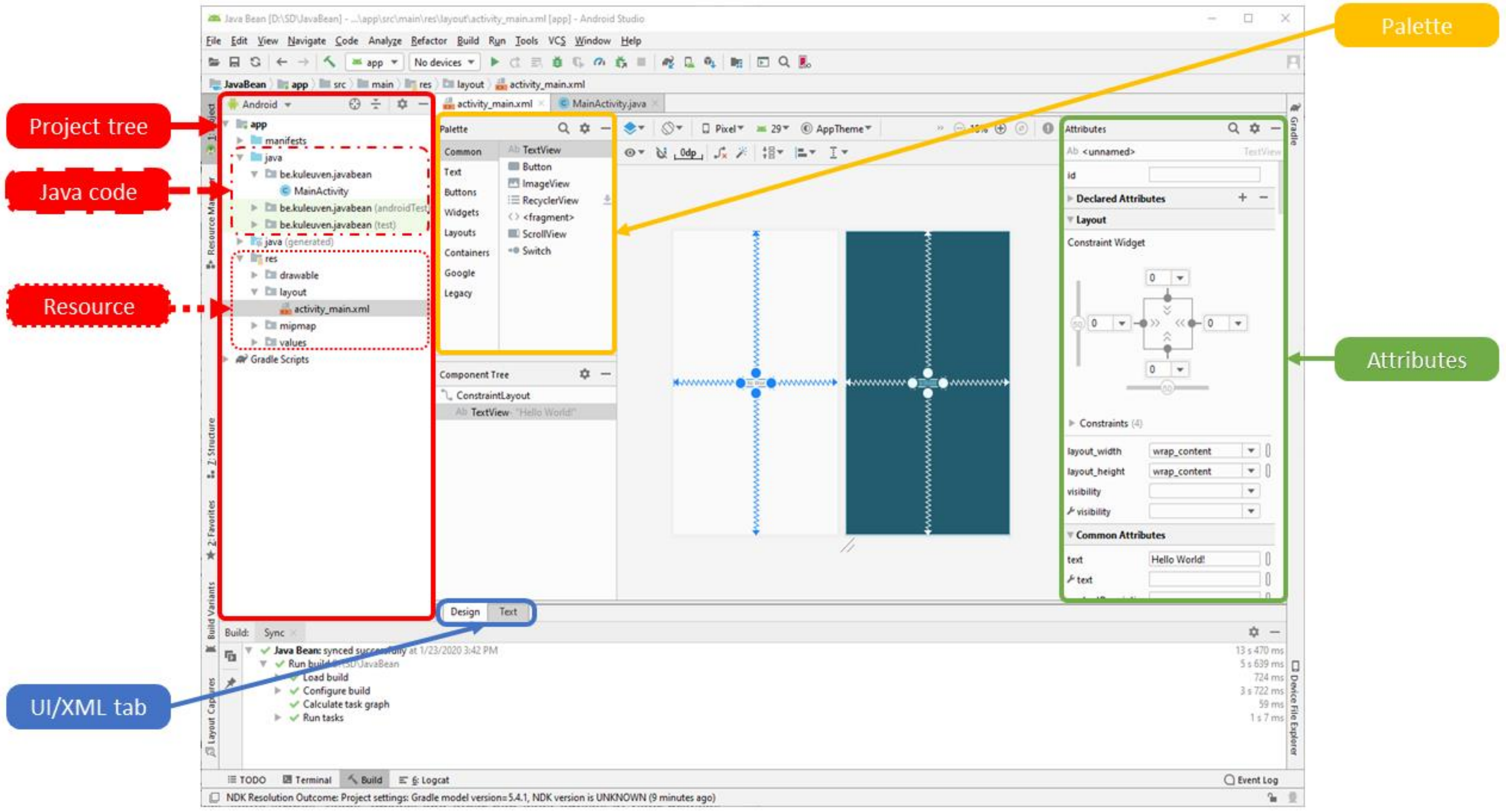
If you have a physical Android device, you may also connect it to your computer and use this as test environment. This will always be faster and smoother than any emulator. More info on how to set up a physical device can be found here: https://developer.android.com/studio/run/device.html.

The most important step is to enable the **Developer Options** on the **Settings** of your device by tapping 7 times (no, it is no joke!) the **build number** (**Settings** -> **About Phone**). Finally, go to **Developer Options** on the **Settings** of your phone and turn on **USB Debugging**.

Getting Started

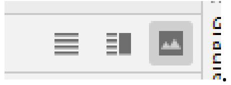
Navigating Android Studio

The window of Android studio contains several important areas as shown below:



```
Build: Sync
✓ Java Bean: synced successfully at 1/23/2020 3:42 PM      13 s 470 ms
  ✓ Run build D:\SD\JavaBean                                 5 s 639 ms
      ✓ Load build                                             724 ms
      ✓ Configure build                                      3 s 722 ms
      ✓ Calculate task graph                                    59 ms
      ✓ Run tasks                                             1 s 7 ms

NDK Resolution Outcome: Project settings: Gradle model version=5.4.1, NDK version is UNKNOWN (9 minutes ago)
```

The area in red box is the navigation tree of all project files. In Android app development, user interface (UI) and codes are separated into different files (java for code, and xml or other files for UI/resources). The highlighted **java** folder contains all java code of your project whereas the **res** (so-called resource folder) contains UI of your activities, static strings, icons, images and other non-code objects of your projects. There, you can find a **layout** folder, which contains all your activities user interface in xml format.

Areas in orange, green, and blue provide necessary tools for UI design, and are only visible once you have opened an xml file of your activity user interface. To create a user interface you can use the design view; it has a rendered view (left) and a blueprint view (right) with only the outlines. You can choose to work with one of the two or with both ("Select Design Surface"). New since **Android Studio version 3.6** the "Design/Text" tabs are replaced by these "Split/Design" buttons (in the top-right corner of the tab): .

Now, if you press the ▶ button when having your phone plugged in or having an emulator, the device will launch your newly made app with "hello world" text in the middle.

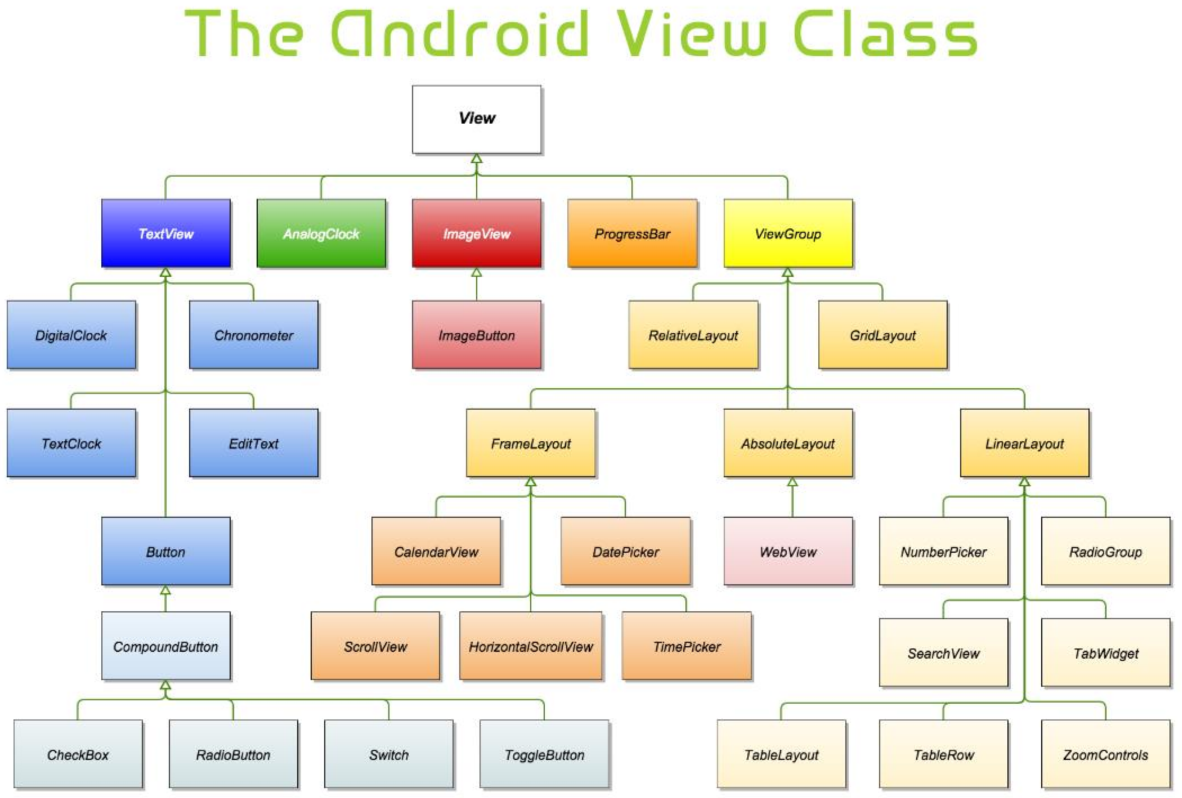
Understanding Android app development concept

Project structure

An Android project comprises of many folders and files, all of which can be overwhelming at the first glance. To make sense of this complex project structure, we first need to grasp the philosophy behind this organizational structure; an Android project is structured according the Model-View-Controller architectural pattern, which separates user interfaces from the application logic (the code that produce dynamic behavior of the application at runtime) and data models. This is the reason behind the separation of the **java** folder and the **res** folder in the project tree. The idea is not only to ensure maintainability of your project, but also to allow user interface designers working independently from programmers. For example, you can give a makeover to your app without changing your java code but just styling the user interface via the drag-drop design tab or the xml text tab (area in blue).

View Hierarchy

In an Android app, everything that you can see on the screen (i.e. all user interface components) is View. View is not only limited to the user interface controls that you can see and interact with (e.g. button or text) but also the components that group and organize other Views. A good metaphor is to think of the way you organize physical objects in everyday life. You may have individual objects laying around; you can also organize them by putting those individual objects into boxes and you can also organize those boxes by putting them into larger boxes. Both individual objects and boxes are physical objects. Similar to View, there are individual Views (e.g., Buttons, Progress bar) but there is also a ViewGroup which organize other Views just like those boxes. Figure[1] on the right shows an overview of the View hierarchy in an Android app.



In this tutorial, we will stick with the terminology View for individual user interface components and ViewGroup for components that organize other views.

Each screen of the app is a so-called Activity; recall that you have chosen **Empty Activity** as the first screen of your app when setting up the project.

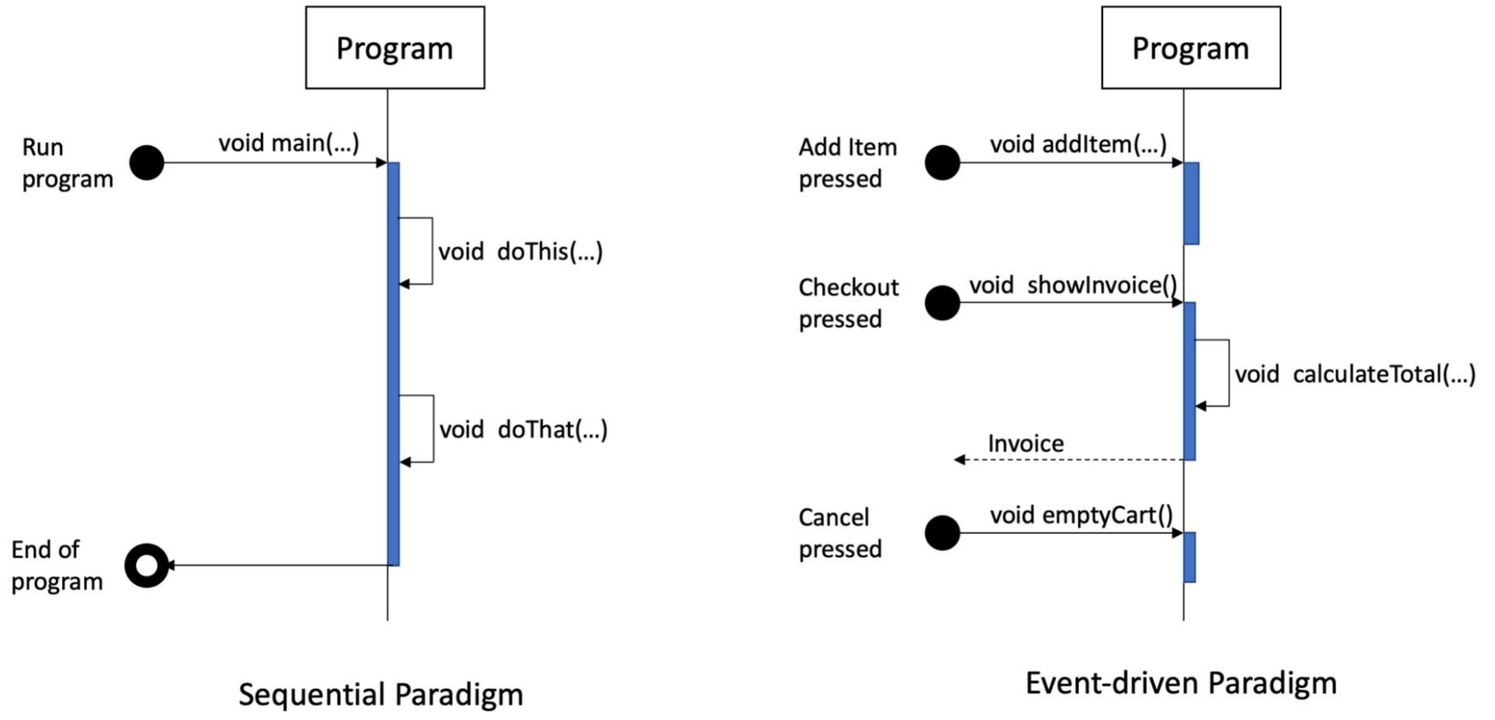
Event-driven system

In an Android app, your code will be executed in an event-driven manner. In other words, your code will only be executed when there is a trigger. For example, if you press a button, a dialog box shows up. The dialog box would not show up until the button is pressed. This means the button pressing action is a trigger, a so called event. The event invokes a piece of code (e.g. a method) that does some computations and show up the dialog box. This piece of code or method is called event handler and it is used as a callback method.

This event-driven paradigm is conceptually different from, but also similar with the paradigm you are familiar with: the sequential programming paradigm, in which each statement executes sequentially one after another from the start to the end of the program (e.g., from the first line of code to the last line in the static void main(…) function). Here, it is important to not be confused between a statement and a line of code. A statement is not necessary referring to a line of code, think of a compound statement that is comprised of many lines of code like *if (condition) {…} and else {…}* which is considered as 1 statement. In the event-driven paradigm, you can consider each event triggers a small program (or more accurately a method) to execute. For instance, if you press the Buy Now button, a piece of code or program or function that calculates the total price and charges the user's credit card will be executed. Afterall, a method or function behaves like a small program for a specific job.

Fact: An event handler can be assigned to an event statically on the XML layout (as shown in the Hands-on section of this tutorial) or dynamically in your java code (see Appendix A: Coupling Layout to Code)

The figure below illustrates the similarities and differences between the sequential and the event-driven paradigm through an example on two UML sequence diagrams.



Hands-on

In this Java Bean tutorial, we will develop a simple app that allows users to order coffees. We will first start with designing a menu and proceed further to accessing a web-service to show users when their coffee will be ready.

User interface design

We are now starting the layout/UI design process of the main activity. In the project tree (the box in red), look for the *activity_main.xml* file. This should be in the *app/res/layout* folder. If you open this file the design view will show up. At the top-right corner you can switch between text, split and design view (new since version 3.6). For this first activity you can choose between working with design or text view. You can switch between these two views by clicking on the buttons. If you choose to use the design view to develop your layouts, frequently check the Text tab to see how it changes. Learn more about XML at Appendix D: XML**.**

Palette window

In the Palette window (area in orange) you can see all kinds of user interface (UI) components or "Views" (subclasses of android.view.View). Some of the common UI components/Views are

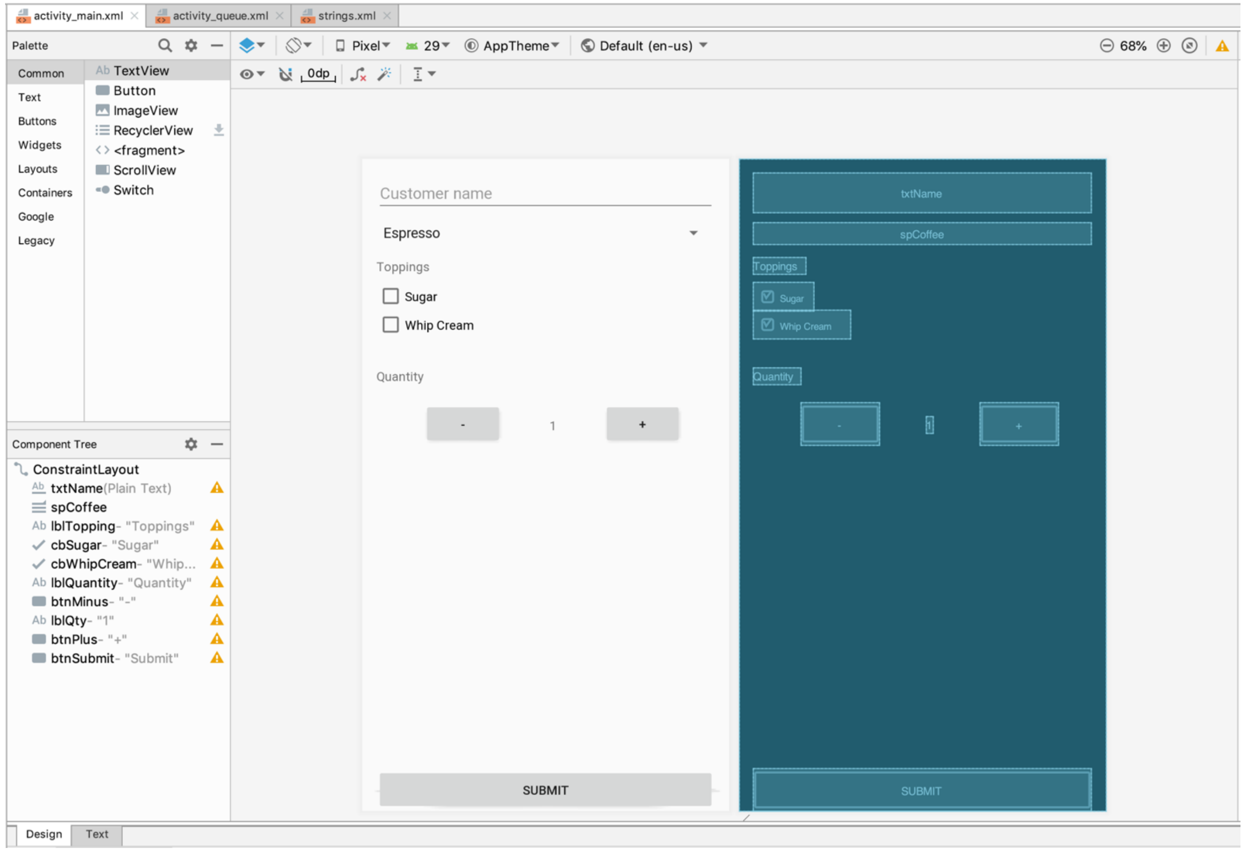
- Widgets and TextFields: visible elements in the layout (buttons, labels, all kinds of textfields, …)
- Containers: elements that can contain widgets and other containers
- Layouts: elements to lay out the different widgets in a container

Attributes

When adding them to the design at least two properties are required for all views: android:layout_width and android:layout_height. (look in the properties window on the right – area in Green). These two properties determine the dimension of each view. For example, some of the possible values are:

- "match_parent": fill the entire parent element
- "wrap_content": as big as necessary to show the content

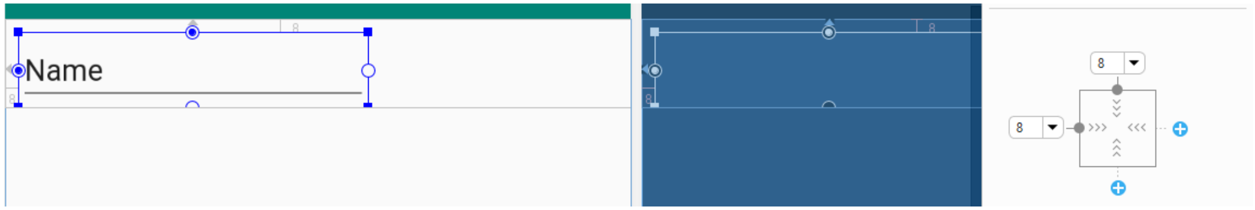
Every widget (so-called view) needs a unique identifier or an "id", to address this widget from within the Java code. Provide for every widget you add a meaningful id, in the format "@+id/<your id>". Step-by-step, try to create the design as shown in the picture on the right - keep an eye on "Component tree" when setting the id of each view.



Working with Views in design mode

a) <u>ConstraintLayout:</u>
A constraint layout structures child elements in a way that all views are laid out according to relationships between sibling views and the parent layout (ScrollView). You need to constraint every view at least one time horizontally and vertically.

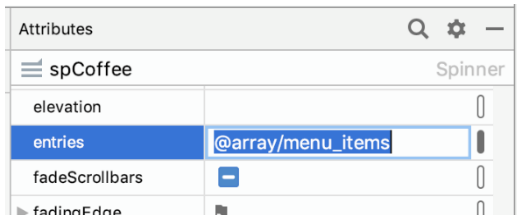
b) Add an EditText field (plain text) to the layout to allow the users to type their name.
This field will have an id to allow us to get a reference in the code. Whenever we first declare an id in XML mode, we need to write “@+id/nameOfTheView” (note the + sign); in Design mode you do not have to type the “@+id/” sequence, because Android Studio will add it. Then we are going to constraint the view. Drag an arrow from the upper circle to the green border of the toolbar to constraint it vertically. Finally, drag the left circle to the edge of the screen to constraint it horizontally. Now the view is constrained in both axes and will stay in the top left corner. Instead of setting the text field to ‘Name’ we will set the hint to ‘Name’. This way our users don’t need to delete the text ‘Name’ to fill in their own name. Your screen should look like this (Tip: you can change the constraint type by pressing on the >>>). Give it the id `txtName`.



c) Spinner (from: Containers)
Add a spinner (with id `spCoffee`) to choose between Espresso, Cappuccino and Latte. Adding menu items (string array) to res/values/strings.xml, so your string.xml looks like the following:

Choose @array/menu_items as entries of the spinner by changing the entries attribute in activity_main.xml (see screenshot on the right). Add constraints to the left of the the edge of the screen and to the bottom of the Name field.

```
<resources>
    <string name="app_name">Java
Bean</string>
    <string-array name="menu_items">
        <item>Espresso</item>
        <item>Cappuccino</item>
        <item>Latte</item>
    </string-array>
</resources>
```



Attributes
spCoffee Spinner
elevation
entries @array/menu_items
fadeScrollbars
fadingEdge

d) TextView
Add a TextView (“Toppings”) to the layout. This is a field that is not changeable by the user, however, this field can be changed by the app. Constraint it horizontally to the left of the edge of the screen and vertically to the bottom of the Spinner. Give it the id `lblTopping`. You can add a reference to a value defined in strings.xml instead of the hard-coded name in the activity_main.xml (the IDE gives a warning and a “hint” to solve it).

e) Checkbox (two times)
Add checkboxes for cream and sugar and put them below the toppings TextView. Do not forget to constraint them. Give them the id `cbSugar` and `cbWhipCream`.

f) TextView (“Quantity”), with id `lblQuantity` (and do not forget the constraints).

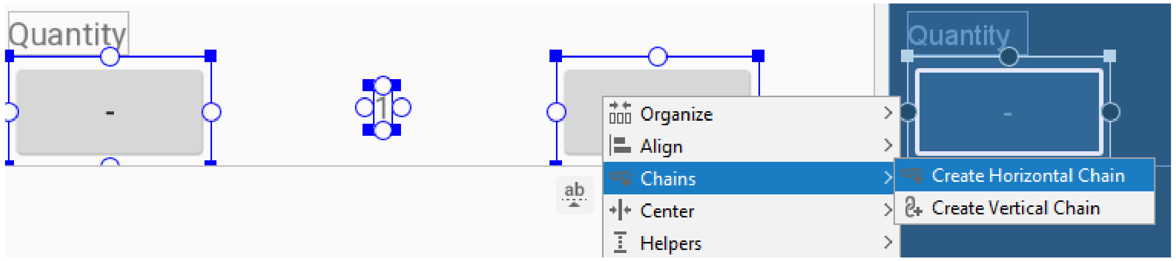
g) Add the following components and drag them in an orderly fashion

i) Button
   Add a button to hold a minus sign and give it the id `btnMinus`.

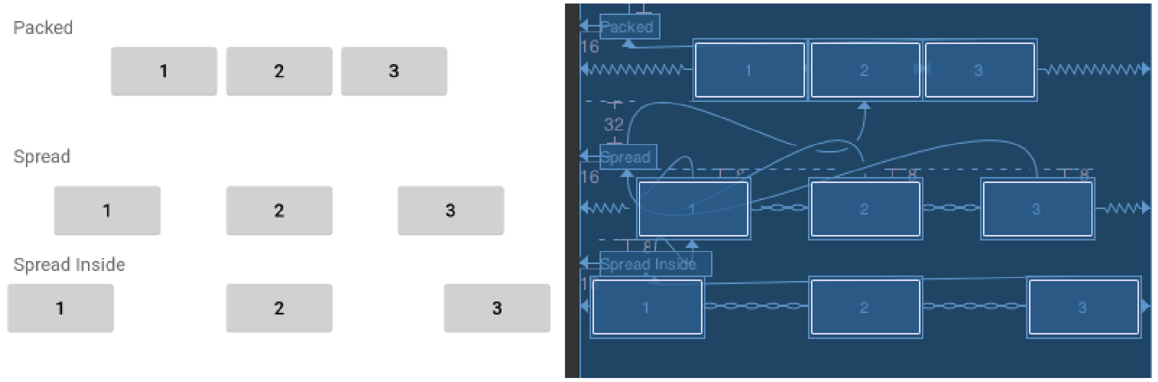
ii) TextView
   Add a TextView to hold the current amount of coffees ordered and give it the id `lblQty`.

iii) Button
   Add another button to hold the plus sign. Define vertical constraints and give it the id `btnPlus`.

Then select all three components while holding ctrl, right click, and choose Horizontal Chain.



Chains allow you to control the space between elements and how the elements use the space. It is possible to cycle through the different chain modes. There are four different modes: Packed, Spread, Spread_inside and Weighted. You can cycle by pressing the chain logo while hovering over a view object in the chain. For now, we will keep it on Spread.



h) Button
   At the bottom, add a button that will be used to submit the order and give it the id `submit`.

Take your time to play around with the settings of the layout editor (take a look at the advanced properties and follow the following link to the Android Developer documentation for more information about the UI properties: https://developer.android.com/guide/topics/ui/index.html).

At this moment, we designed the user interface of the app, but the app has no logic in it. It is now the excellent time to run your JavaBean app. Click the green 'Run App' arrow ▶ if you want to launch your app on your phone. If you want to run it on the emulator, click the Genymotion icon and start your device of choice, then click the green arrow.

As you can see, everything seems to work on the surface. We can type our name in the field, check the boxes, and push the buttons. But nothing happens when buttons are pushed. Now we will add functionality to these visual components.

Event handling & View manipulation

Recall Project structure and

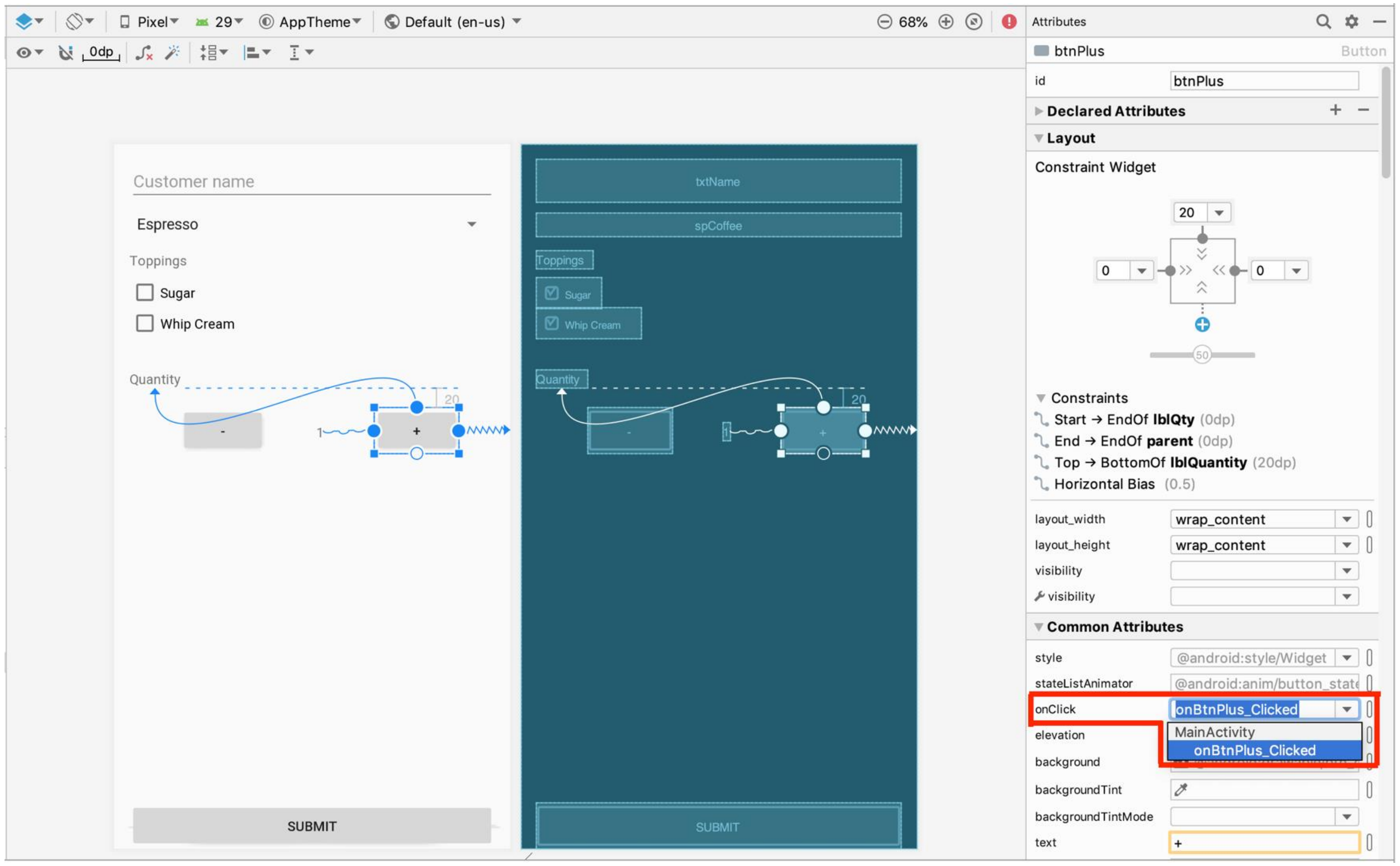
Event-driven system, every UI xml file has a corresponding Java file where all logic resides. When we compile our app, Android Studio will automatically search for corresponding Java files to implement said logic. So, to program our activity_main.xml, we work with the MainActivity.java file in the Java folder.

The MainActivity class comes with the onCreate(Bundle savedInstanceState) method, which overrides a method from a superclass. This is a lifecycle callback method that is called by the Android system when an activity is first created; here it sets up the correct view. See Appendix B for the other lifecycle callback methods. We will now implement + (btnPlus) and – (btnMinus) button handlers which update the quantity (lblQty) in TextView in the middle once any of the buttons are clicked.

To handle the click event of + (btnPlus), add the following code to your MainActivity.java file.

```java
package be.kuleuven.javabean;
import androidx.appcompat.app.AppCompatActivity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.CheckBox;
import android.widget.EditText;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {
    private Button btnPlus;
    private TextView lblQty;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        btnPlus = (Button) findViewById(R.id.btnPlus);
        lblQty = (TextView) findViewById(R.id.lblQty);
    }

    public void onBtnPlus_Clicked(View caller) {
        int quantity = Integer.parseInt(lblQty.getText().toString()) + 1;
        lblQty.setText(Integer.toString(quantity));
    }

}
```

Then on your activity_main.xml file, click on the plus button and change the onClick property of the button to “onBtnPlus_Click” as show in the figure below (recall static event handler assignment):



btnPlus Button

id btnPlus

Constraints
Start → EndOf lblQty (0dp)
End → EndOf parent (0dp)
Top → BottomOf lblQuantity (20dp)
Horizontal Bias (0.5)

layout_width wrap_content
layout_height wrap_content
visibility
visibility

style @android:style/Widget
stateListAnimator @android:anim/button_state
onClick onBtnPlus_Clicked
MainActivity
onBtnPlus_Clicked
elevation
background
backgroundTint
backgroundTintMode
text +

Run your code and try press the + button and see the effect. Recall

Event-driven system; what happened is when the + button is clicked (= the event), the public void onBtnPlus_Clicked(View caller) callback method is executed as a handler. Error messages appear under the "Run"-tab (ALT-4).

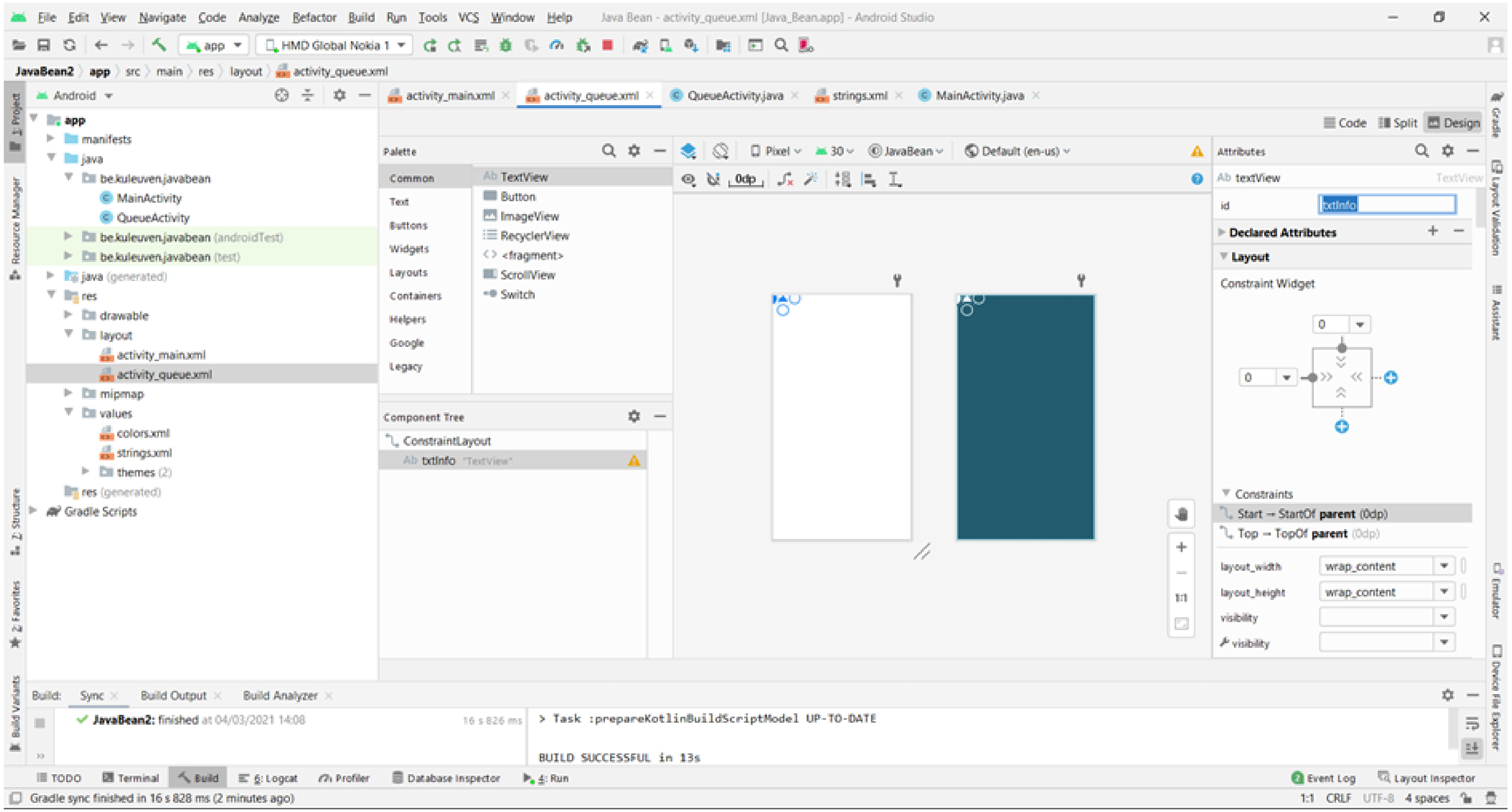
Challenge: Implement the onClick event of the – button. Keep in mind that the quantity should not go below 0 and if the quantity is 0, the user shall not be able to press the submit button (setEnabled method). Do not forget to couple the listener/callback method to the button. Check if the submit button is enabled/disabled as it should; maybe you will have to add some code to the btnPlus listener too.

Intent

You can think of an intent as an "intent to do something". It is a type of message that allows you to bind separate objects (such as activities) together at runtime. If one activity wants to start a second activity, it does it by sending an intent to Android. Android will start the second activity and pass it the intent. (from: Head First Android Development, ISBN-13: 978-1449362188, O'Reilly).

In this tutorial, we intend (no pun intended) to open a new activity and pass data (customer name, coffee, toppings, and quantity) to the new activity that we intended to open.

1. Create a new empty activity called "QueueActivity" and place a TextView with id `txtInfo` as a child. See screenshot below:



```
JavaBean2: finished at 04/03/2021 14:08        16 s 826 ms
```

```
> Task :prepareKotlinBuildScriptModel UP-TO-DATE

BUILD SUCCESSFUL in 13s
```

```
Gradle sync finished in 16 s 828 ms (2 minutes ago)
```

2. Add the following method to your MainActivity class, and don't forget to assign this method as the handler for the onClick event of the Submit button (btnSubmit) in main_activity.xml file.

```java
public void onBtnSubmit_Clicked(View caller) {
    EditText txtName = (EditText) findViewById(R.id.txtName);
    Spinner spCoffee = (Spinner) findViewById(R.id.spCoffee);
    CheckBox cbSugar = (CheckBox) findViewById(R.id.cbSugar);
    CheckBox cbWhipCream = (CheckBox) findViewById(R.id.cbWhipCream);

    Intent intent = new Intent(this, QueueActivity.class);
    intent.putExtra("Name", txtName.getText());
    intent.putExtra("Coffee", spCoffee.getSelectedItem().toString());
    intent.putExtra("Sugar", cbSugar.isChecked());
    intent.putExtra("WhipCream", cbWhipCream.isChecked());
    intent.putExtra("Quantity", Integer.parseInt(lblQty.getText().toString()));

    startActivity(intent);
}
```

If you read the code on the left closely, you may notice that this method does the following:

a. Get user inputs from name EditText, coffee spinner, toppings' checkboxes
b. Create a new intent to open the newly created activity (QueueActivity)
c. Pass user inputs to the intent, which will in turn, pass those data to the activity we want to start
d. Start the new activity

(in another tutorial we will refactor this code an introduce a model)

You may now try running your app and click the submit button. The newly created activity will be started once the button is pressed.

3. You will notice the emptiness of the newly opened QueueActivity i.e. with only the word "info" or "TextView" written on it. We will show the data passed from the MainActivity by adding the following code to your QueueActivity.java file:

```java
package be.kuleuven.javabean;

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

public class QueueActivity extends AppCompatActivity {
    private TextView txtInfo;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_queue);

        txtInfo = (TextView) findViewById(R.id.txtInfo);
        Bundle extras = getIntent().getExtras();
        String coffeeData = extras.get("Name") + ": " +
                extras.get("Quantity") + "x " +
                extras.get("Coffee") +
                (extras.getBoolean("Sugar")? " + sugar" : "") +
                (extras.getBoolean("WhipCream")? " + whip cream" : "") + "\n";
        txtInfo.setText(coffeeData);
    }
}
```

The code on the left is rather straight forward. The code does the following:

a. When the activity is created (See Appendix B for the other lifecycle callback methods), the method tries to get the instance of TextView "txtInfo" from the QueueActivity
b. Get data passed from the MainActivity (extras) from the intent -to open this activity-
c. Getting individual piece of coffee information and concatenate them in a string "coffeeData"
d. Show the concatenated string on TextView txtInfo.

Challenges:

- What does '`(extras.getBoolean("Sugar")? " + sugar" : "")`' mean?
- Format the information as follows:

Show the name of the customer and the ordered coffee *n* times, where *n* is the quantity.

E.g., if the "John" orders 2 cappuccinos with sugar, show:

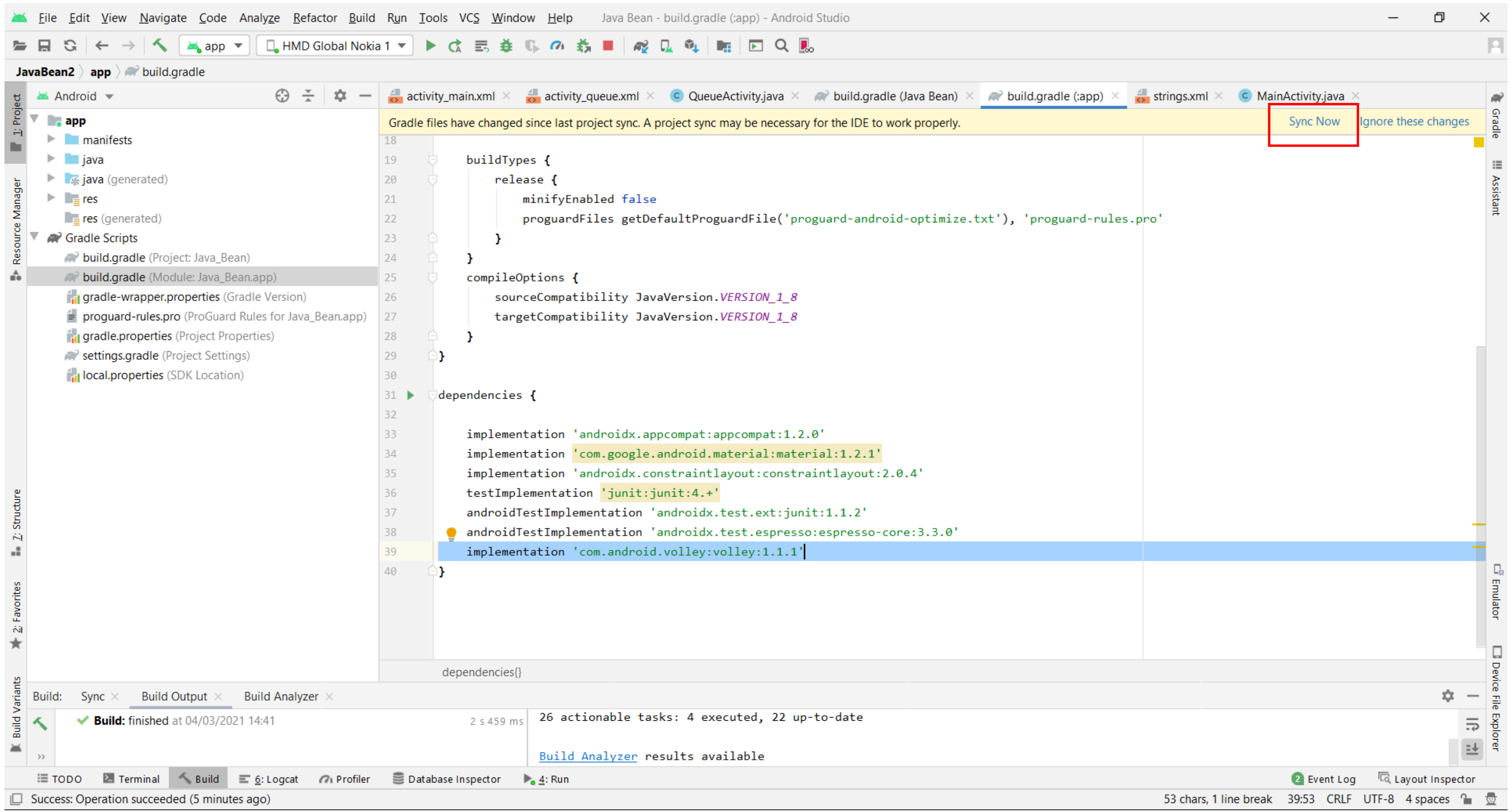
John:
cappuccino + sugar
cappuccino + sugar

Working with external data sources

Our coffee ordering app is somehow boring and unrealistic. This is because the app does not remember anything, and the customers do not know when their coffee will be ready. In this tutorial, we will make this app communicate with a server that takes coffee orders and allows the app to inquire a coffee order list from a waiting queue. The communication with the server is done through a RESTful web service. Explanation about web service architecture will be given in the lab session or, due to the corona issues, in a short video on Toledo.

Adding Volley to the project

We will use an external library “Volley” to facilitate the app-server communication through http-calls; to do so, add the `implementation 'com.android.volley:volley:1.1.1'` line to the dependency of build.gradle (Module:Java_Bean.app). As a result, your build.gradle should look like the following:


File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
Java Bean - build.gradle (:app) - Android Studio
app
HMD Global Nokia 1
JavaBean2 › app › build.gradle
Android
app
manifests
java
java (generated)
res
res (generated)
Gradle Scripts
build.gradle (Project: Java_Bean)
build.gradle (Module: Java_Bean.app)
gradle-wrapper.properties (Gradle Version)
proguard-rules.pro (ProGuard Rules for Java_Bean.app)
gradle.properties (Project Properties)
settings.gradle (Project Settings)
local.properties (SDK Location)
activity_main.xml
activity_queue.xml
QueueActivity.java
build.gradle (Java Bean)
build.gradle (:app)
strings.xml
MainActivity.java
Sync Now
Ignore these changes
TODO
Terminal
Build
6: Logcat
Profiler
Database Inspector
4: Run
Event Log
Layout Inspector

Gradle files have changed since last project sync. A project sync may be necessary for the IDE to work properly.

```
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
        }
    }
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
}

dependencies {

    implementation 'androidx.appcompat:appcompat:1.2.0'
    implementation 'com.google.android.material:material:1.2.1'
    implementation 'androidx.constraintlayout:constraintlayout:2.0.4'
    testImplementation 'junit:junit:4.+'
    androidTestImplementation 'androidx.test.ext:junit:1.1.2'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.3.0'
    implementation 'com.android.volley:volley:1.1.1'
}
```

```
Build: Sync  Build Output  Build Analyzer
✓ Build: finished at 04/03/2021 14:41     2 s 459 ms
26 actionable tasks: 4 executed, 22 up-to-date

Build Analyzer results available
```

```
Success: Operation succeeded (5 minutes ago)     53 chars, 1 line break  39:53  CRLF  UTF-8  4 spaces
```

Don’t forget to click **Sync Now** to let Android Studio download volley and add it to your project.

Sending and Querying data

Modify the QueueActivity.java to the following code. It executes 2 requests:

1. request to submit an order (after submitting an order, a “date_due” will be calculated by adding a random nr of minutes between 4 and 9 to date of order)
2. request to get information about the waiting queue for that order (this request is sent on receiving the response of the first request and shows all “open” orders)

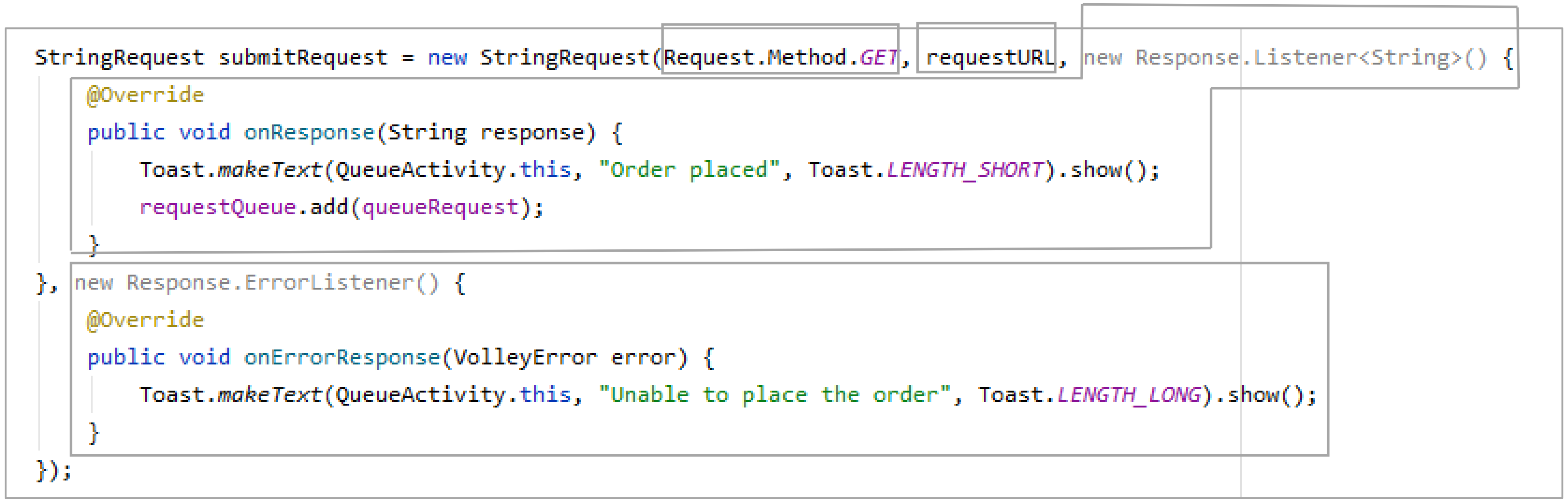
```java
public class QueueActivity extends AppCompatActivity {
    private TextView txtInfo;
    private RequestQueue requestQueue;
    private static final String SUBMIT_URL = "https://studev.groept.be/api/ptdemo/order/";
    private static final String QUEUE_URL = "https://studev.groept.be/api/ptdemo/queue";
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_queue);

        txtInfo = (TextView) findViewById(R.id.txtInfo);
        requestQueue = Volley.newRequestQueue(this);

        Bundle extras = getIntent().getExtras();
        String toppings = (extras.getBoolean("Sugar")? "+sugar" : "") + (extras.getBoolean("WhipCream")? "+cream" : "");
        String requestURL = SUBMIT_URL + extras.get("Name") + "/" +
                extras.get("Coffee") + "/" +
                ((toppings.length() == 0)? "-" : toppings) + "/" +
                extras.get("Quantity");


        JsonArrayRequest queueRequest = new JsonArrayRequest(Request.Method.GET,QUEUE_URL,null,new Response.Listener<JSONArray>() {
            @Override
            public void onResponse(JSONArray response) {
                String info = "";
                for (int i=0; i<response.length(); ++i) {
                    JSONObject o = null;
                    try {
                        o = response.getJSONObject(i);
                        info += o.get("customer") + ": " + o.get("coffee") + " x " + o.get("quantity") + " " +
                                o.get("toppings") + " will be ready at " + o.get("date_due") + "\n";
                    } catch (JSONException e) {
                        e.printStackTrace();
                    }
                }
                txtInfo.setText(info);
            }
        }, new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error) {
                Toast.makeText(QueueActivity.this, "Unable to communicate with the server", Toast.LENGTH_LONG).show();
            }
        });

        StringRequest submitRequest = new StringRequest(Request.Method.GET, requestURL, new Response.Listener<String>() {
            @Override
            public void onResponse(String response) {
                Toast.makeText(QueueActivity.this, "Order placed", Toast.LENGTH_SHORT).show();
                requestQueue.add(queueRequest);
            }
        }, new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error) {
                Toast.makeText(QueueActivity.this, "Unable to place the order", Toast.LENGTH_LONG).show();
            }
        });
        requestQueue.add(submitRequest);
    }
}
```

A full explanation of the structure of a volley request is given in a couple of short videos on Toledo, but here you can find a quick explanation of the most important parts of the code:

Defining the submitRequest

These lines of code initialize the submitRequest variable by calling its constructor with 4 parameter (to be clear: this is the initialization of the "submitRequest" object of type StringRequest; no code is executed):

1. the http-request method to be used (GET or POST)
2. the request URL to be used
3. the callback method to be called in case of a normal response, defined by an anonymous inner class that implements the `Response.Listener` interface; this method will also add the queueRequest to the requestQueue
4. the callback method to be called in case of an error, defined by an anonymous inner class that implements the `Response.ErrorListener` interface

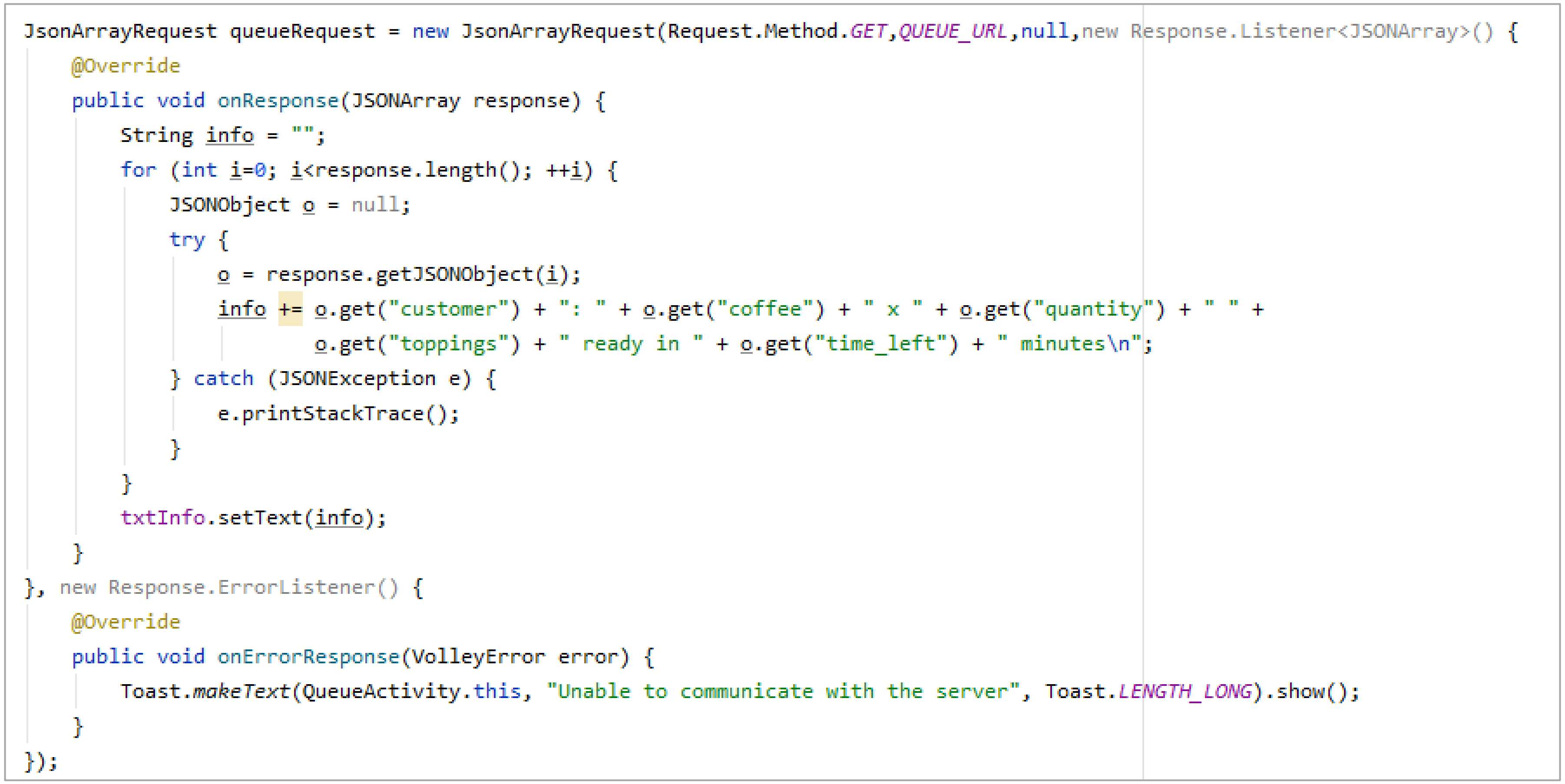


```
StringRequest submitRequest = new StringRequest(Request.Method.GET, requestURL, new Response.Listener<String>() {
    @Override
    public void onResponse(String response) {
        Toast.makeText(QueueActivity.this, "Order placed", Toast.LENGTH_SHORT).show();
        requestQueue.add(queueRequest);
    }
}, new Response.ErrorListener() {
    @Override
    public void onErrorResponse(VolleyError error) {
        Toast.makeText(QueueActivity.this, "Unable to place the order", Toast.LENGTH_LONG).show();
    }
});
```

Defining the queueRequest

These lines of code initialize the queueRequest (you can compare it with the submitRequest initialization):

1. the http-request method to be used (GET or POST)
2. the request URL to be used
3. the callback method to be called in case of a normal response
4. the callback method to be called in case of an error



```java
JsonArrayRequest queueRequest = new JsonArrayRequest(Request.Method.GET,QUEUE_URL,null,new Response.Listener<JSONArray>() {
    @Override
    public void onResponse(JSONArray response) {
        String info = "";
        for (int i=0; i<response.length(); ++i) {
            JSONObject o = null;
            try {
                o = response.getJSONObject(i);
                info += o.get("customer") + ": " + o.get("coffee") + " x " + o.get("quantity") + " " +
                        o.get("toppings") + " ready in " + o.get("time_left") + " minutes\n";
            } catch (JSONException e) {
                e.printStackTrace();
            }
        }
        txtInfo.setText(info);
    }
}, new Response.ErrorListener() {
    @Override
    public void onErrorResponse(VolleyError error) {
        Toast.makeText(QueueActivity.this, "Unable to communicate with the server", Toast.LENGTH_LONG).show();
    }
});
```

Send the submitRequest

The method

```
requestQueue.add(submitRequest);
```

will send the submitRequest and starts the execution of this Activity.

Before running your app, you will have to give it the permission to do an INTERNET request. Add this line in the manifest file, before the closing </manifest> tag:

```
<uses-permission android:name="android.permission.INTERNET"/>
```

When running your app, you will be able to place coffee orders, know when your orders will be ready and see a list of orders made by you and your classmates.

Facts: The web service is nothing more than a web application running on a web server and a database server (MySQL server). You can add custom services at https://studev.groept.be/api/[YOUR_STUDEV_ACCOUNT], and log in with your studev account – to be provided in the class. Instructions & explanation to be provided in the lab session ☺ - another reason to stay in the lab session. Read Appendix C about JSON first.

To design your database, download the MySQL workbench, add a database connection to the studev mySQL server using the following information:

- Host : mysql.studev.groept.be
- Port : 3306
- Login : [YOUR_STUDEV_ACCOUNT]
- Password : [YOUR_STUDEV_PASSWORD]

Note: the database server is located at our -Groep T- campus and can only be accessed within the campus. To get your MySQL workbench to work with the server, your development computer must be connected to any of the following Wifi: Campusroam, Campusroam 2.4, eduroam, eduroam 2.4 in the campus. From outside the campus you can install a SSL VPN Pulse client to create a VPN (Virtual Private Network) and connect to the B-zone.

To setup your database and how to use the web service that queries the database and returns JSON data will be explained in two short videos on Toledo (or in class).

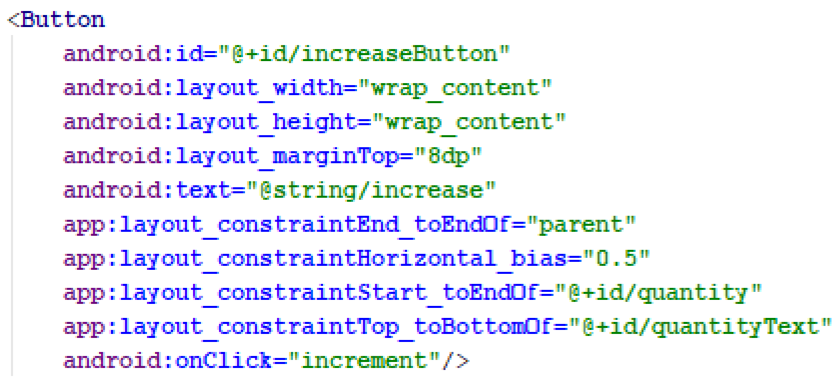
Appendix A: Coupling Layout to Code

There are two ways to couple actions (= Java code) with the interface. This way you can couple a function to a button click. The two ways are:

- in the XML layout file
- using listeners in Java code

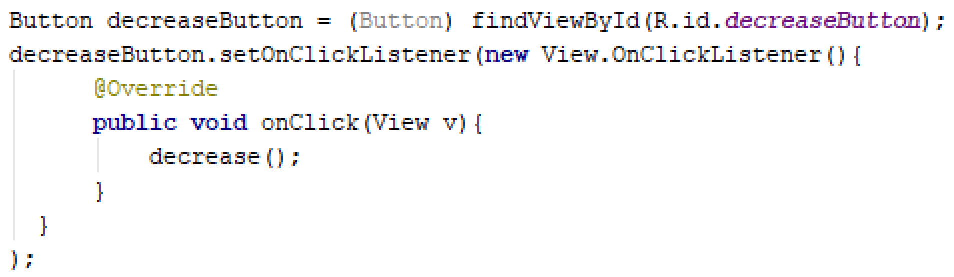
a) Link button using XML

The method should have this signature: `public void <method name>(View view)`



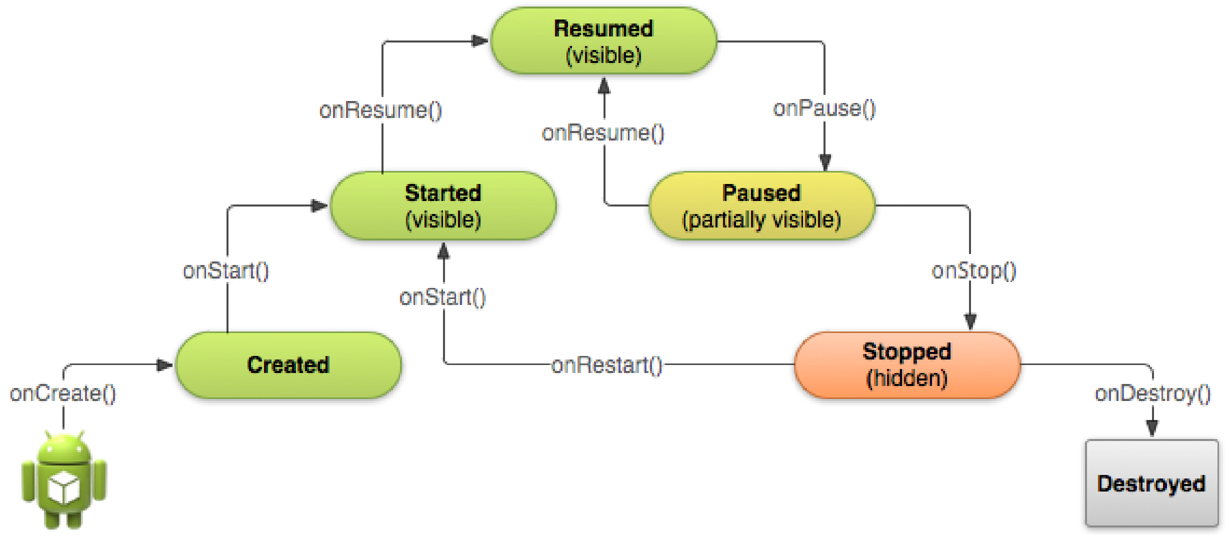
```xml
<Button
    android:id="@+id/increaseButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="8dp"
    android:text="@string/increase"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.5"
    app:layout_constraintStart_toEndOf="@+id/quantity"
    app:layout_constraintTop_toBottomOf="@+id/quantityText"
    android:onClick="increment"/>
```

b) Link button in code (within the onCreate method) using onClickListener



```java
Button decreaseButton = (Button) findViewById(R.id.decreaseButton);
decreaseButton.setOnClickListener(new View.OnClickListener(){
      @Override
      public void onClick(View v){
          decrease();
      }
  }
);
```

Appendix B: Lifecycle callback methods



Source: https://developer.android.com/training/basics/activity-lifecycle/starting.html

Here are the Android activity lifecycle callbacks methods (information from: Head First Android Development, ISBN-13: 978-1449362188, O'Reilly) and when they are called:

- `onCreate()`: When the activity is first created. Use it for normal static setup, such as creating views. It also gives you a Bundle giving the previous saved state of the activity.
- `onRestart()`: When your activity has been stopped just before it gets started again.
- `onStart()`: When your activity is becoming visible. It is followed by `onResume()` if the activity comes into the foreground or `onStop()` if the activity is made invisible.
- `onResume()`: When your activity is in the foreground.
- `onPause()`: When your activity is no longer in the foreground because another activity is resuming. The next activity is not resumed until this method finishes, so any code in this method needs to be quick. It is followed by `onResume()` if the activity returns to the foreground or `onStop()` if it becomes invisible.
- `onStop()`: When the activity is no longer visible. This can be because another activity is covering it or because the activity is being destroyed. It is followed by `onRestart()` if the activity becomes visible again or `onDestroy()` if the activity is going to be destroyed.
- `onDestroy()`: When your activity is about to be destroyed or because the activity is finishing.

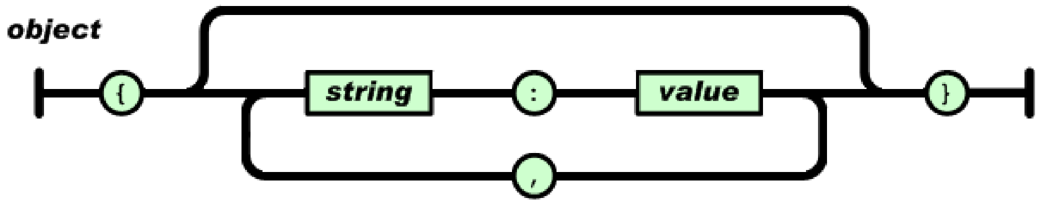
Appendix C: JSON

JSON (JavaScript Object Notation) is a lightweight data-interchange format. Its main advantage is that it is language independent. In other words, it can be easily used to send data from a java application to a C++ application or from a database to an android app.

There are two main structures in JSON:

JSON Objects

Collections of key/values pairs, typically representing objects. An object starts with a {, followed by one or more String:Value pairs separated by a , and is closed with a }.



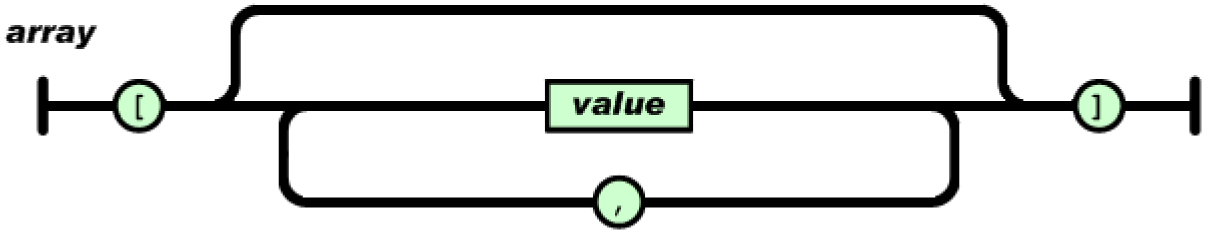
JSON Object Example

This example represents a Person object with a String name, an integer age and a String city.

```
{
    "name":"John",
    "age":31,
    "city":"New York"
}
```

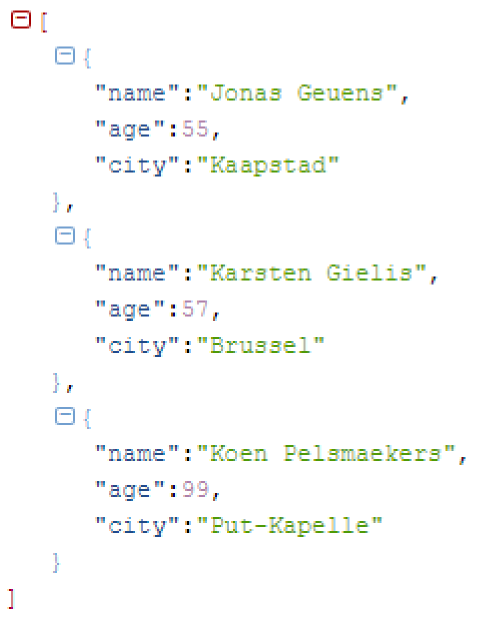
JSON Array

An ordered list of values or JSON Objects, typically representing collections such as Lists, Arrays, Vectors, … . An array starts with a [and ends with a]. Objects in an array are separated with a ,.



JSON Array Example

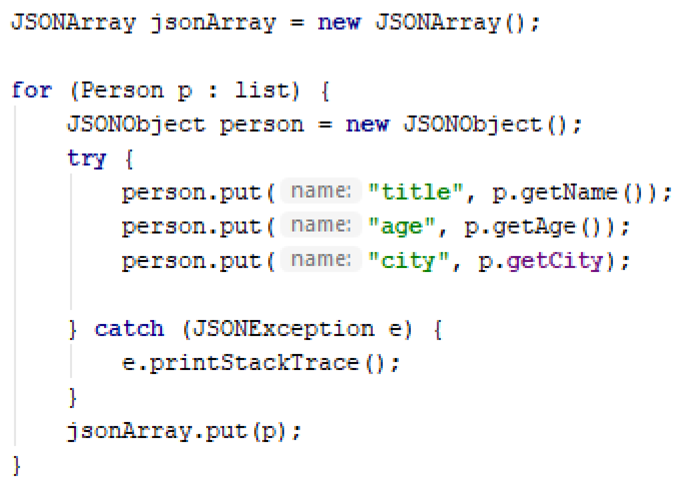
The JSON Array starts with a **[** , contains 3 person objects separated with a **,** and ends with a **]**.



```
[
    {
        "name":"Jonas Geuens",
        "age":55,
        "city":"Kaapstad"
    },
    {
        "name":"Karsten Gielis",
        "age":57,
        "city":"Brussel"
    },
    {
        "name":"Koen Pelsmaekers",
        "age":99,
        "city":"Put-Kapelle"
    }
]
```

Creating JSON Data in Java

This code recreates the JSONArray as found above. Please note that the first parameter of the put function represents the key and the second parameter the value with which it is associated. These key values are essential to getting your data out of JSON.

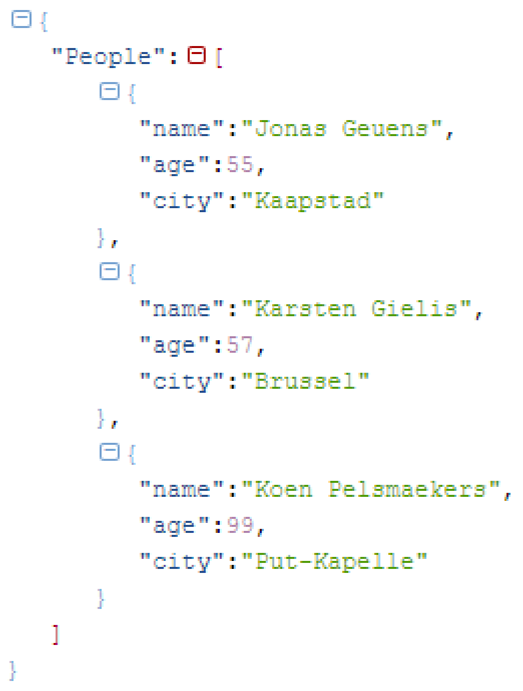


```
JSONArray jsonArray = new JSONArray();

for (Person p : list) {
    JSONObject person = new JSONObject();
    try {
        person.put( name: "title", p.getName());
        person.put( name: "age", p.getAge());
        person.put( name: "city", p.getCity);

    } catch (JSONException e) {
        e.printStackTrace();
    }
    jsonArray.put(p);
}
```

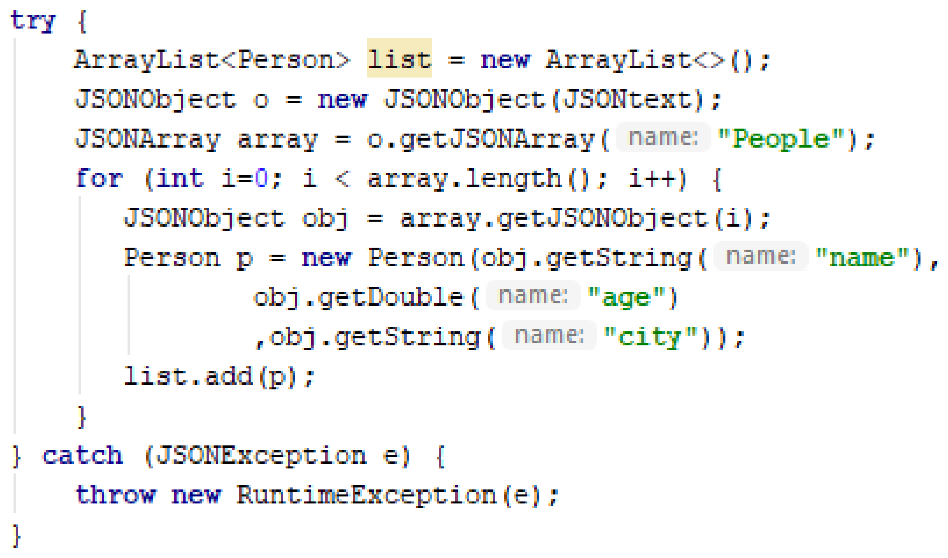
Retrieving JSON Data in Java

Please note that in this example the JSONArray is wrapped in a JSONObject (indicated with the {}) with as key People. So to retrieve our objects, we first need to get the JSONArray from the JSONObject. Then we need to iterate over all JSONObjects (Jonas, Karsten and Koen) and instantiate them as Plain Old Java Objects (POJOs).

We assume that the JSON data is given to us as a very long String: {"People":[{ "name":"Jonas Geuens", "age":55, "city":"Kaapstad" }, { "name":"Karsten Gielis", "age":57, "city":"Brussel" }, { "name":"Koen Pelsmaekers", "age":99, "city":"Put-Kapelle" }]}. JSON parsers such as http://json.parser.online.fr/ can help us to visualize the data in a more comprehensible way.



```
{
   "People":[
      {
         "name":"Jonas Geuens",
         "age":55,
         "city":"Kaapstad"
      },
      {
         "name":"Karsten Gielis",
         "age":57,
         "city":"Brussel"
      },
      {
         "name":"Koen Pelsmaekers",
         "age":99,
         "city":"Put-Kapelle"
      }
   ]
}
```



```
try {
    ArrayList<Person> list = new ArrayList<>();
    JSONObject o = new JSONObject(JSONtext);
    JSONArray array = o.getJSONArray( name: "People");
    for (int i=0; i < array.length(); i++) {
        JSONObject obj = array.getJSONObject(i);
        Person p = new Person(obj.getString( name: "name"),
                obj.getDouble( name: "age")
                ,obj.getString( name: "city"));
        list.add(p);
    }
} catch (JSONException e) {
    throw new RuntimeException(e);
}
```

More on JSON

https://www.json.org/

https://www.w3schools.com/js/js_json_intro.asp

http://www.vogella.com/tutorials/AndroidJSON/article.html

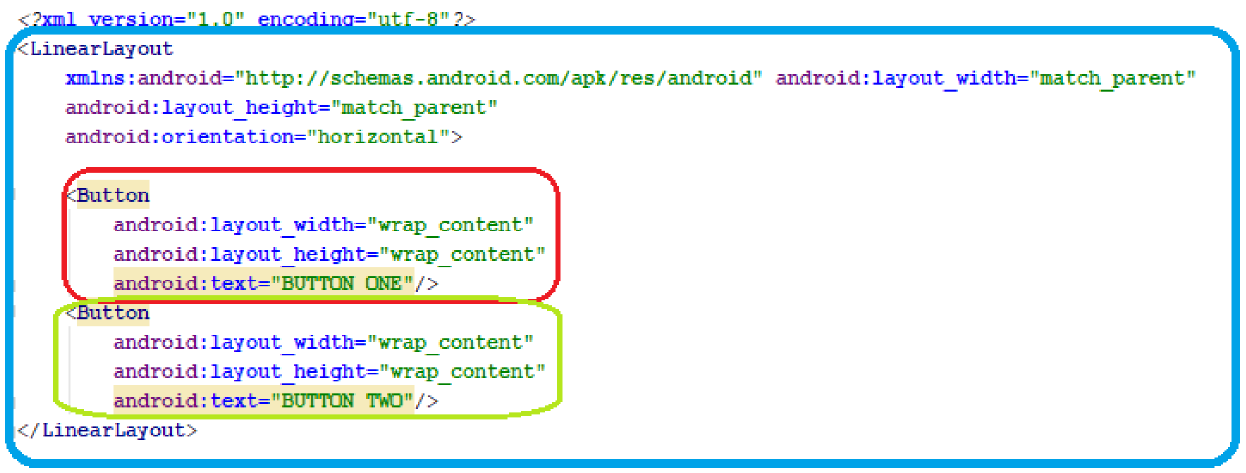
Appendix D: XML

XML stands for eXtensible Markup Language. It was originally designed as a comprehensible way to store and transport data. In the Android environment it is used to describe resources, for example the user interface, fixed text, animations and even icons used by your application. It decouples the design (or view) of your app from the logic behind it. This is beneficial because you can modify the user interface of your app without changing the source code. You can create XML layouts for different devices, screens and languages without duplicating logic.

Each Android XML layout typically starts with a ViewGroup. This element defines the way that child elements behave.

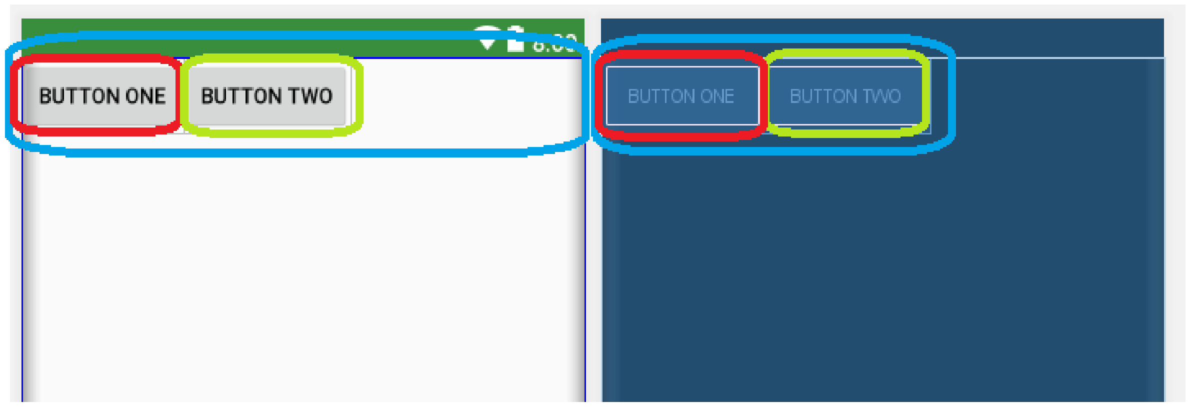
Example: Two buttons (child elements) inside a linear layout (viewgroup) with horizontal orientation will be placed horizontally beside each other.

XML



```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android" android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal">

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="BUTTON ONE"/>
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="BUTTON TWO"/>
</LinearLayout>
```

Rendered layout



Each individual layout element can have multiple properties. The android:text property, for example, defines the text of the button. Layout_width and layout_height define the width and height of the button. In this case this means that the button will not be bigger than needed to wrap the content.

The "android:" is used to target the namespace. What this means is that when the application is compiled, the compiler knows that it has to look in the android namespace to find, for example, the "text" field.

Please do note that the manner of defining a XML element without child nodes differs from one that has child nodes.

Type 1: Childless XML element

The childless XML element starts with a single <, followed by the type of element, the properties, and ends with a />.

```xml
<Button android:id="@+id/cancel"
    android:layout_height="wrap_content"
    android:layout_width="wrap_content"
    android:layout_weight="1"
    android:text="@string/cancel" />
```

Type 2: Child nodes

The XML element with children starts with a single <, followed by the type of element, the properties and a closing tag >. Then the child elements are defined. When all children are defined, the parent is closed with a </type> tag.

```xml
<LinearLayout android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <Button
        android:id="@+id/ok"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:text="Ok"
        android:layout_weight="1"
        />

</LinearLayout>
```

For each element there are multiple properties with multiple options. To style each element to your design, check the android documentation exhaustively at https://developer.android.com/index.html

More on XML:

https://www.w3schools.com/xml/

https://developer.android.com/guide/topics/ui/declaring-layout.html https://developer.android.com/guide/topics/ui/declaring-layout.html#CommonLayouts