LAB_1

Teacher: Veikko Tapaninen

Student: Oussama Lghachi

Group: DIN18SP

- ***1 - Study following object oriented concepts:***

➢ *Description of: (Object, Class, Instantiation of object)*

- A Class is like a template that we use to define an object, in it we specify all the variable types and name that an object can have. As well as procedures and methods for operating on those objects.
- The object is a variable of that class. A program can have multiple objects or types of classes in the same time.
- Instantiation of object or in other words creating a new object is done by the **new** operator that instantiates a class by allocating memory for a new object of that type. The new operator creates the object, and then the constructor initializes it

Example from lab1 application:

MainActivity.java

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // Object person_1 is created with name, age and gender using DeliveryMan
Class
        DeliveryMan person_1 = new DeliveryMan("Bob", 25, "male");
        // Getter is used to get the object Name and Age then display it on the
toast
        Toast.makeText(getApplicationContext(), "You chose
"+person_1.getmName()+"with age    "+person_1.getmAge(),
                Toast.LENGTH_SHORT).show();
        // Setter is used to set object person_1 name and age  then display new
information
```

```
        person_1.setmName("Jack");
        person_1.setmAge(49);
        Toast.makeText(getApplicationContext(), "You chose
"+person_1.getmName()+"with age   "+person_1.getmAge(),
                Toast.LENGTH_SHORT).show();
    }
```

DeliveryMan.java

```
public class DeliveryMan {
    private String mName;
    private int mAge;
    private String gender;
    public DeliveryMan(String mName, int mAge, String gender) {
        this.mName = mName;
        this.mAge = mAge;
        this.gender = gender;
    }
                // Getters used to get the variable name,age and gender
    public String getmName() {
        return mName;
    }
                // Setters used to set the variable name,age and gender
    public void setmName(String mName) {
        this.mName = mName;
    }
    public int getmAge() {
        return mAge;
    }

    public void setmAge(int mAge) {
        this.mAge = mAge;
    }
    public String getGender() {
        return gender;
    }
    public void setGender(String gender) {
        this.gender = gender;
    }
}
```

In the line:

```
DeliveryMan person_1 = new DeliveryMan("Bob", 25, "male");
```

- It creates a new DeliveryMan object (DeliveryMan is a class in the java.util package). This single statement actually performs three actions: declaration, instantiation, and initialization.” DeliveryMan person_1” is a variable declaration which simply declares to the compiler that the name person_1 will be used to refer to an object whose type is DeliveryMan, the new operator instantiates the

DeliveryMan class (thereby creating a new DeliveryMan object), and DeliveryMan initializes the object.

- *Description of: (Visibility (public / private / protected))*

  - *Definition and basic knowledge:* Visibility or rather “access modifiers” are what basically allows us to access a class or method in the package depending on the keywork used “modifier used” in the project.
    There are three kinds of access modifiers (public, protected, private and no modifier). The table below shows the differences between each modifier, Y is Yes and N is NO.

Modifier	Class	Package	Subclass	World
Public	Y	Y	Y	Y
Protected	Y	Y	Y	N
No modifier	Y	Y	N	N
Private	Y	N	N	N

Giving the wrong modifier can give other users access to our class or variable when they are not supposed to. It is always good practice to use private when declaring variables in classes, and public when declaring classes so we can use them in same package.

- *Example from Lab1 application:*

1 - No modifier: ***Sports.java***

```
package com.example.lab1_examples;
public class Sports {
        // THIS CLASS IS IN THE SAME PACKAGE WITH OUR ACTIVITY
        // NO_MODIFIER EXAMPLE BEFORE THE VARIABLES
        String mType;
        int mYear;
}
```

MainActivity.java

```
// NO_MODIFIER EXAMPLE PART
Sports sport = new Sports();
sport.mType = "BODYBUILDING";
```

In this case we are able to access the object and make the type of the sport into bodybuilding. However, if we move the Sports.java from the package android studio cannot run the application and will give in an error message that it is not visible. The variable becomes red

```
// NO_MODIFIER EXAMPLE PART
Sports sport = new Sports();
sport.mType = "BODYBUILDING";
```

With Error message: “mType is not public in Sports; cannot be accessed from outside package”

2 - Public: Sports.java

One way to solve this problem is by making the variable Public like in the example:

```
public class Sports {
/*
        // THIS CLASS SHOULD BE IN THE SAME PACKAGE WITH OUR ACTIVITY
        // NO_MODIFIER EXAMPLE BEFORE THE VARIABLES
        String mType;
        int mYear;
 */
        // THIS CLASS DOESNT HAVE TO BE IN THE SAME PACKAGE WITH OUR ACTIVITY
        // USING PUBLIC BEFORE THE VARIABLES MAKES THEM ACCESSIBLE FROM EVERYWHERE
        public String mType;
        public int mYear;
```

This solved the problem and the emulator runs perfectly now like before:

MainActivity.java

```
// NO_MODIFIER EXAMPLE PART
Sports sport = new Sports();
sport.mType = "BODYBUILDING";
```

However, it is not good practice to make variable public because that would give other users access to them as well and would indirectly infect them.

3 – Private & protected: Sports.java

The good practice would be to declare the variable private or protected as shown in the example so it is only accessible through the class itself.

```
public class Sports {
/*
        // THIS CLASS SHOULD BE IN THE SAME PACKAGE WITH OUR ACTIVITY
        // NO_MODIFIER EXAMPLE BEFORE THE VARIABLES
        String mType;
        int mYear;
 */
/*
        // THIS CLASS DOESNT HAVE TO BE IN THE SAME PACKAGE WITH OUR ACTIVITY
        // USING PUBLIC BEFORE THE VARIABLES MAKES THEM ACCESSIBLE FROM EVERYWHERE
```

```
        public String mType;
        public int mYear;
 */
        // THIS CLASS HAVE TO BE IN THE SAME PACKAGE WITH OUR ACTIVITY
        // USING PRIVATE MAKES SURE THAT OUR VARIABLES ARE NOT AFFECTED INDIRECTLY
BY OTHER USERS
        private String mType;
        private int mYear;
}
```

This solved the problem and the emulator runs perfectly now like before:

MainActivity.java

```
// NO_MODIFIER EXAMPLE PART
Sports sport = new Sports();
sport.mType = "BODYBUILDING";
```

However, it is not good practice to make variable public because that would give other users access to them as well and affect them indirectly.

Protected can be used when we focus more on inheritance and other similar subjects where protected would be more useful than private.

- *Description of: (Member datas / methods)*

  - *Definition and basic knowledge:* Data member are variables that are declared inside a class that we can simply call global variables or fields. If the variable is declared static then it can be called without an object in the class, but if it is declared non static then it should be called using the class object.

  - *Example from Lab1 application:*

DataMembers_example.java

```
package com.example.lab1_examples;
public class DataMembers_example {
    int Money;
    String FirstNAme;
    public static void main(String[] args) {
        System.out.println(Money);
    }
}
```

The error message that is shown is “Non-static field”Money” cannot be referenced from a static context”. This is because the variable were declared non static.

After changing the Money variable to static the error disappeared and application returned 100.

```
 package com.example.lab1_examples;
public class DataMembers_example {
/*
```

```
    int Money;
    String FirstNAme;
    public static void main(String[] args) {
        System.out.println(Money);

 */
    static int Money = 100;
     String FirstNAme;
    public static void main(String[] args) {
        System.out.println(Money);
    }
}
```

There is another way of doing this which is using the class object like shown in the new example below:

```
package com.example.lab1_examples;
public class DataMembers_example {
/*
         // EXAMPLE 1 USING NON STATIC
    int Money;
    String FirstNAme;
    public static void main(String[] args) {
        System.out.println(Money);
 */
        // EXAMPLE 2 USING STATIC
  /*  static int Money = 100;
     String FirstNAme;
        public static void main(String[] args) {
        System.out.println(Money);
   */
        // EXAMPLE 3 USING CLASS OBJECT
    int Money = 100;
    String FirstNAme;
    public static void main(String[] args) {
        DataMembers_example m = new DataMembers_example();
        System.out.println(m.Money);
    }
}
```

- *Definition and basic knowledge:* A method simply put is a chunk of code that we run with one line. It used to avoid repetitive coding and simplifying the program.

- *Example from Lab1 application:*

Let's say we wanted to multiply two variables together multiple times, without methods we will have to write code like this and its very annoying and energy wasting

Methods_Example.java

```
package com.example.lab1_examples;
public class Methods_Example {
    public static void main(String[] args) {
        int a = 5;
        int b = 8;
```

```
        System.out.println(a*b);
        int c = 5;
        int d = 8;
        System.out.println(c*d);

        int e = 5;
        int f = 8;
        System.out.println(e*f);
    }
}
```

A method is created by using public static void similar like the main method

```
package com.example.lab1_examples;
public class Methods_Example {
    public static void main(String[] args) {
        int a = 5;
        int b = 8;
        System.out.println(a*b);
        int c = 5;
        int d = 8;
        System.out.println(c*d);
        int e = 5;
        int f = 8;
        System.out.println(e*f);
    }
    public static void Method(int a, int b) {
       System.out.println(a*b);
    }
}
```

Using the Method all we have to do is call it and put in the variable we want to use and its much better this way

```
public class Methods_Example {
    public static void main(String[] args) {
/*
        int a = 5;
        int b = 8;
        System.out.println(a*b);
        int c = 6;
        int d = 9;
        System.out.println(c*d);
        int e = 3;
        int f = 8;
        System.out.println(e*f);
*/
          Method(5,8);
          Method(6,9);
          Method(3,8);
    }
    public static void Method(int a, int b) {
       System.out.println(a*b);
    }
}
```

➢ *Description of: (Inheritance):*

- *Definition and basic knowledge:* Inheritance is basically inheriting stuff from another class, it useful to instead of repeating similar methods in every kind of class we have we can easily make all of them inherit of one main class. the classes that inherit stuff from the main class are called Subclasses and the class they inherit from is called the superclass

- *Example from Lab1 application:*

Let's say we have types of food classes (chicken / Fish) and then we have the main class which is FOOD.

Each of these classes have the same method eat () in them.

Without the inheritance method are code will have repetitive codes as shown below:

Chicken.java

```
package com.example.lab1_examples.Inheritance_example;
public class Chicken {
    public void  eat() {
        System.out.println("I am the eat Method");
    }
}
```

Fish.java

```
package com.example.lab1_examples.Inheritance_example;
public class Fish {
        public void  eat() {
            System.out.println("I am the eat Method");
        }
    }
```

Note: that the Super class need to have public method to be inherited instead of private or it will show an error saying that is not visible (this was explained before in the access modifiers part)

If we now use the Inheritance method code will be:

Chicken.java

```
package com.example.lab1_examples.Inheritance_example;
public class Chicken extends FOOD {
}
```

Fish.java

```java
package com.example.lab1_examples.Inheritance_example;
public class Fish extends FOOD {
}
```

FOOD.java

```java
package com.example.lab1_examples.Inheritance_example;
public class FOOD {
        public void  eat() {
            System.out.println("I am the eat Method");
        }
    }
```

Note: if Chicken inherits from Fish and Fish inherits from FOOD, Then Chicken will inherit from FOOD as well which is called hierarchy.

➢ *Description of: (Interface):*

- *Definition and basic knowledge:* An interface is simply put as an outline for a class, it can be implemented when we can't to create for example a crate but we didn't know how. That is where interface comes in because we can use one that was already made and implement it in out class.
- *Example from Lab1 application:*

Let's say we wanted to create a crate but we didn't know how? the interface above our class Crate can be used to create one for us:

Inheritance_Example.java

```java
package com.example.lab1_examples;
    interface Crate {
    String Color = "Brown";
    void openBox() ;
    }
    public class Interface_Example implements Crate {
        public static void main(String[] args) {
            System.out.println(Color);
            Interface_Example box_1 = new Interface_Example();
            box_1.openBox();
        }
        @Override
        public void openBox() {
            System.out.println("The box Has been opened!");
        }
    }
```

Note: Everything in the interface has to be mentioned in the class!

➢ *Description of: (Overriding /Polymorphism):*

- *Definition and basic knowledge:* For Polyphormism we will use the previous example for Inheritance to understand the concept. Because we need to have a class that inherits from another. This way of working saves us the trouble of having to always create new object with its methods repetitively. This pretty much loop through each of our object and calls the eat method for each one of them. With we can assign different objects to variable as long as the reference variable is of a superclass type.

Fish.java and Chicken.java have eat method that Overridee the Food eat method to their own also mentioned in the example.

Example from Lab1 application:

FOOD.java

```
package com.example.lab1_examples.Inheritance_example;
public class FOOD {
    public void  eat() {
        System.out.println("I am the eat Method");
    }
}
```

Fish.java

```
package com.example.lab1_examples.Inheritance_example;
public class Fish extends FOOD {
    public void  eat() {
        System.out.println("This Fish is awesome");
    }
    }
```

Chicken.java

```
package com.example.lab1_examples.Inheritance_example;
public class Chicken extends FOOD {
    public void  eat() {
        System.out.println("I love Chicken");
    }
}
```

Apples.java

```
package com.example.lab1_examples.Inheritance_example;
public class Apples {
    public static void main(String[] args) {
        FOOD Jacky [] = new FOOD[1];
        Jacky[0] = new Fish();
        Jacky[1] = new Chicken();
        for (int x=0 ; x < 2 ; ++x) {
                Jacky[x].eat();
        }
    }
}
```

This pretty much loop through each of our object and calls the eat method for each one of them. With we can assign different objects to variable as long as the reference variable is of a superclass type.

- *Description of: (Abstract Classes):*

  - *Definition and basic knowledge:* An abstract class is basically a bunch of variables and methods that can be used to create other classes. However, the abstract class cannot be used like a normal class instead it is used only used to help build other classes.

  - Example from Lab1 application:

In the Example the Phone Abstract class is used to help build a new class called Iphone, the Iphone has everything that the Phone class has and is used in the code below:

Abstract_Example.java

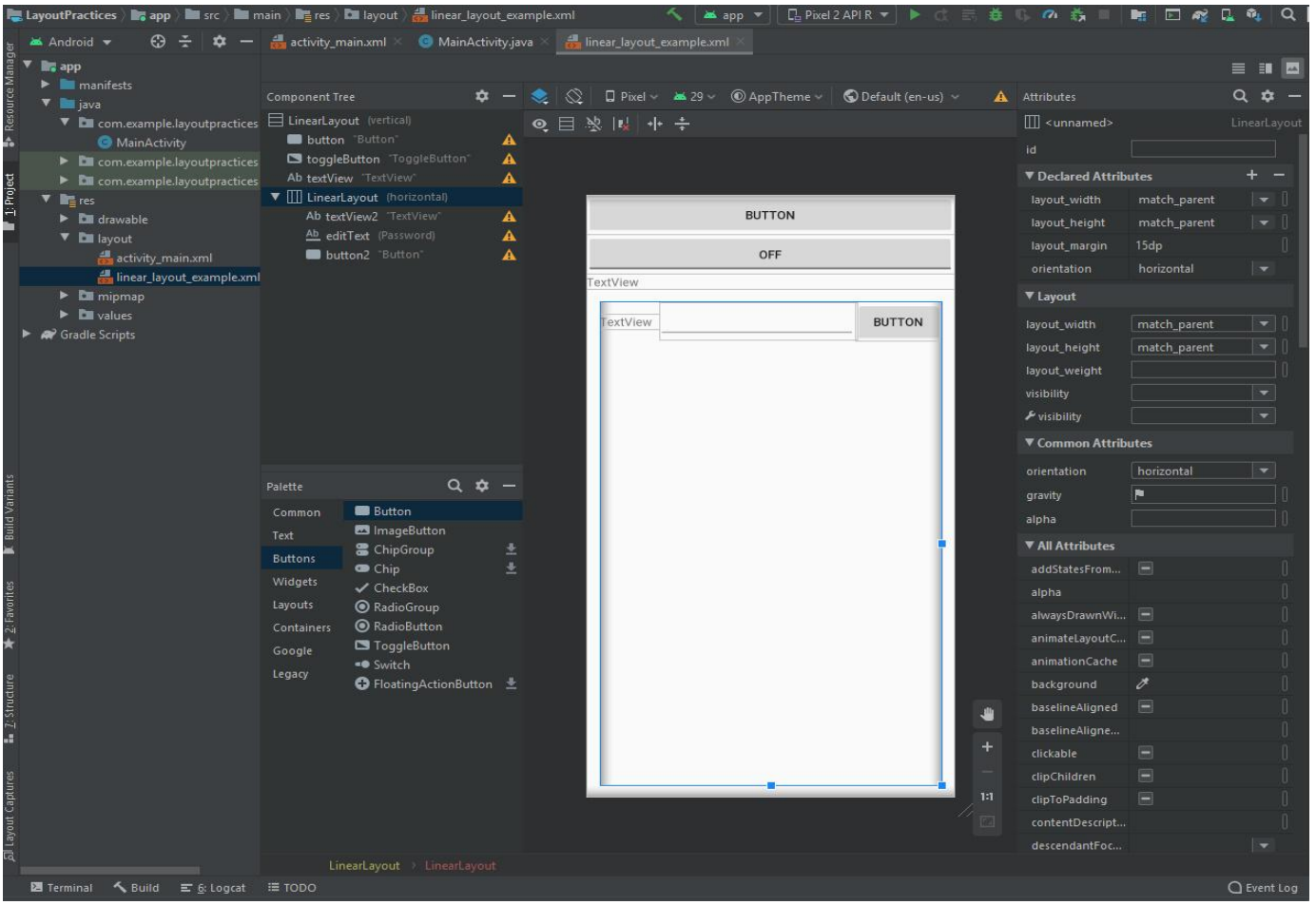
```
package com.example.lab1_examples;
abstract class Phone {
        String Owner;
        public void Ring() {
            System.out.println("The phone is ringing");
        }
}
class Iphone extends Phone {
}
public class Abstract_Example {
    public static void main(String[] args) {
        Iphone telephone = new Iphone();
        telephone.Ring();
        System.out.println(telephone.Owner);
    }
}
```

- ***2 – More UI Design: (These are included in Definition Examples/ and EASY-FIT folders)***

➢ ***Linear Layout:***

Below I have been practicing the LinearLayout to see how easy it can be to be used, and also to see what would be the biggest advantage the disadvantage for future comparisons with ConstraintLayout and CoordinatorLayout.

- *Experiment Actions:* At the start, I played a bit with the orientation which was Vertical. Then I implemented few stuffs into the application like a Button,ToggleButton and a TextView…etc. This part of the practice showed all these components being lined up vertically in order which led me to change the orientation to horizontal for the next part of the experiment and include it as a child of the previous LinearLayout.
  I have also tried the margins padding and many other features to see how they all work and tested the application in the emulator.
- Example from Lab1 application:


LayoutPractices
app
src
main
res
layout
linear_layout_example.xml
Pixel 2 API R
activity_main.xml
MainActivity.java
linear_layout_example.xml
Android
app
manifests
java
com.example.layoutpractices
MainActivity
com.example.layoutpractices
com.example.layoutpractices
res
drawable
layout
activity_main.xml
linear_layout_example.xml
mipmap
values
Gradle Scripts
Component Tree
LinearLayout (vertical)
button "Button"
toggleButton "ToggleButton"
textView "TextView"
LinearLayout (horizontal)
textView2 "TextView"
editText (Password)
button2 "Button"
Palette
Common
Text
Buttons
Widgets
Layouts
Containers
Google
Legacy
Button
ImageButton
ChipGroup
Chip
CheckBox
RadioGroup
RadioButton
ToggleButton
Switch
FloatingActionButton
Pixel
29
AppTheme
Default (en-us)
BUTTON
OFF
TextView
TextView
BUTTON
1:1
Attributes
<unnamed>
LinearLayout
id
Declared Attributes
layout_width
match_parent
layout_height
match_parent
layout_margin
15dp
orientation
horizontal
Layout
layout_width
match_parent
layout_height
match_parent
layout_weight
visibility
visibility
Common Attributes
orientation
horizontal
gravity
alpha
All Attributes
addStatesFrom...
alpha
alwaysDrawnWi...
animateLayoutC...
animationCache
background
baselineAligned
baselineAligne...
clickable
clipChildren
clipToPadding
contentDescript...
descendantFoc...
LinearLayout
LinearLayout
Terminal
Build
6: Logcat
TODO
Event Log
Resource Manager
1: Project
Build Variants
2: Favorites
7: Structure
Layout Captures

Linear_layout_example.xml

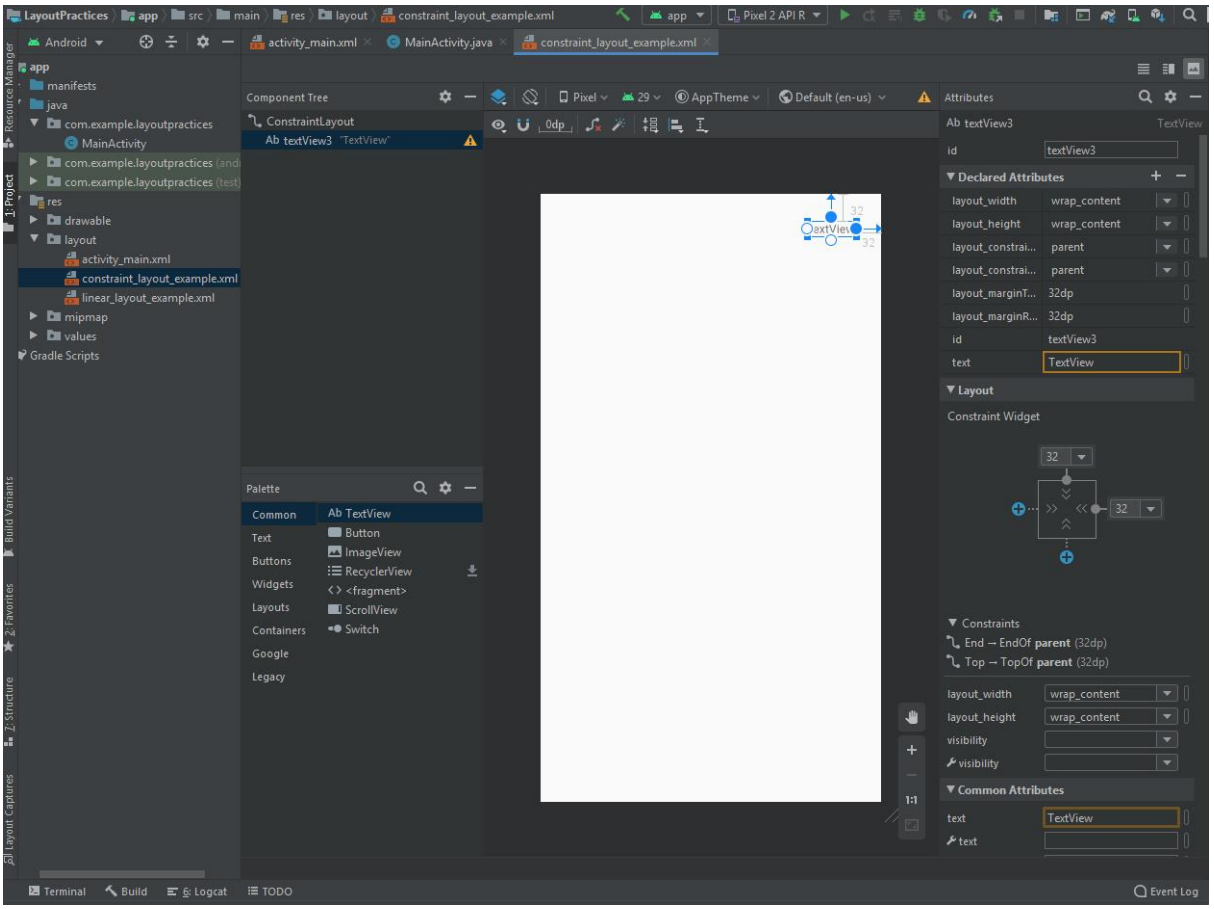
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">
    <Button
        android:id="@+id/button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Button" />
    <ToggleButton
        android:id="@+id/toggleButton"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="ToggleButton" />
    <TextView
        android:id="@+id/textView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="TextView" />
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="horizontal"
        android:layout_margin="15dp">
        <TextView
            android:id="@+id/textView2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="TextView" />
        <EditText
            android:id="@+id/editText"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:ems="10"
            android:inputType="textPassword" />
        <Button
            android:id="@+id/button2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Button" />
    </LinearLayout>
</LinearLayout>
```

➢ ***Constraint Layout:***

- *General idea about constraint layout and experiment example:* The constraint layout is an evolution of the Relative layout, it should be implemented in all newer android studio version, the main idea is that when using relative layout you will face problems that will force you to implement linear layout as a sub layout to make the application look as you wish and solve your problem. This is what the constraint layout role plays. In the example, I have been experimenting with it to see how it works and how I could use it for my future project.

For example, when putting a simple TextView in the constraint layout the application shows me on the sides the margins option but also the horizontal and vertical constraint. Without these two constraints defined it will not run, these constraints define where in my screen the TextView is placed exactly.

Example from Lab1 application:



constraint_layout_example.xml

```
<?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"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:id="@+id/textView3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="32dp"
        android:layout_marginEnd="32dp"
        android:layout_marginRight="32dp"
        android:text="TextView"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Note: The components can also be linked to each other when building a layout which can be very helpful and make sure that one component doesn't take another's position by mistake.

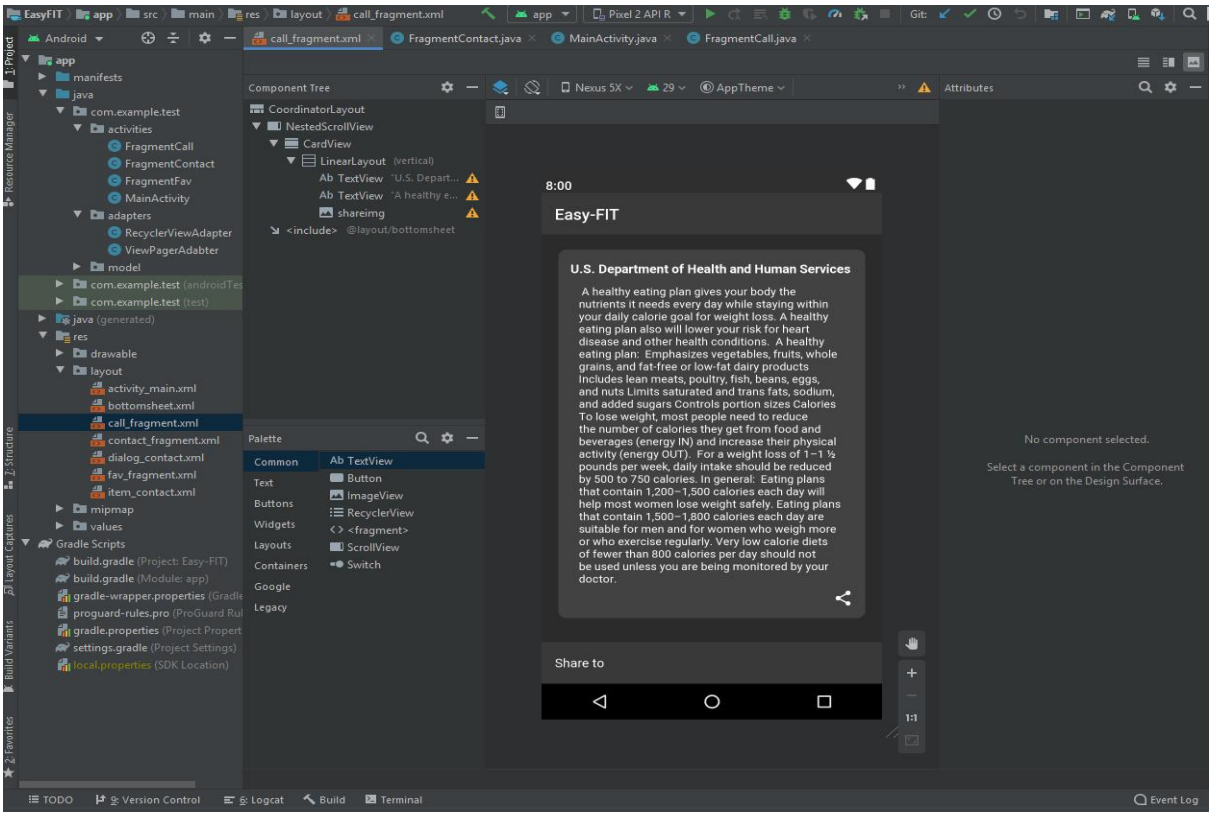
- ***Coordinator Layout: (EASY-FIT is the example used here)***

  - *General idea about Coordinator layout and experiment example:* The coordinator layout is the most powerful layout designer in android studio. It is a superpowered frame layout that facilitate how views interact with each other in the application.
    This is done by assigning specific behavior to each of those view, these behaviors are what these kinds of design unique.

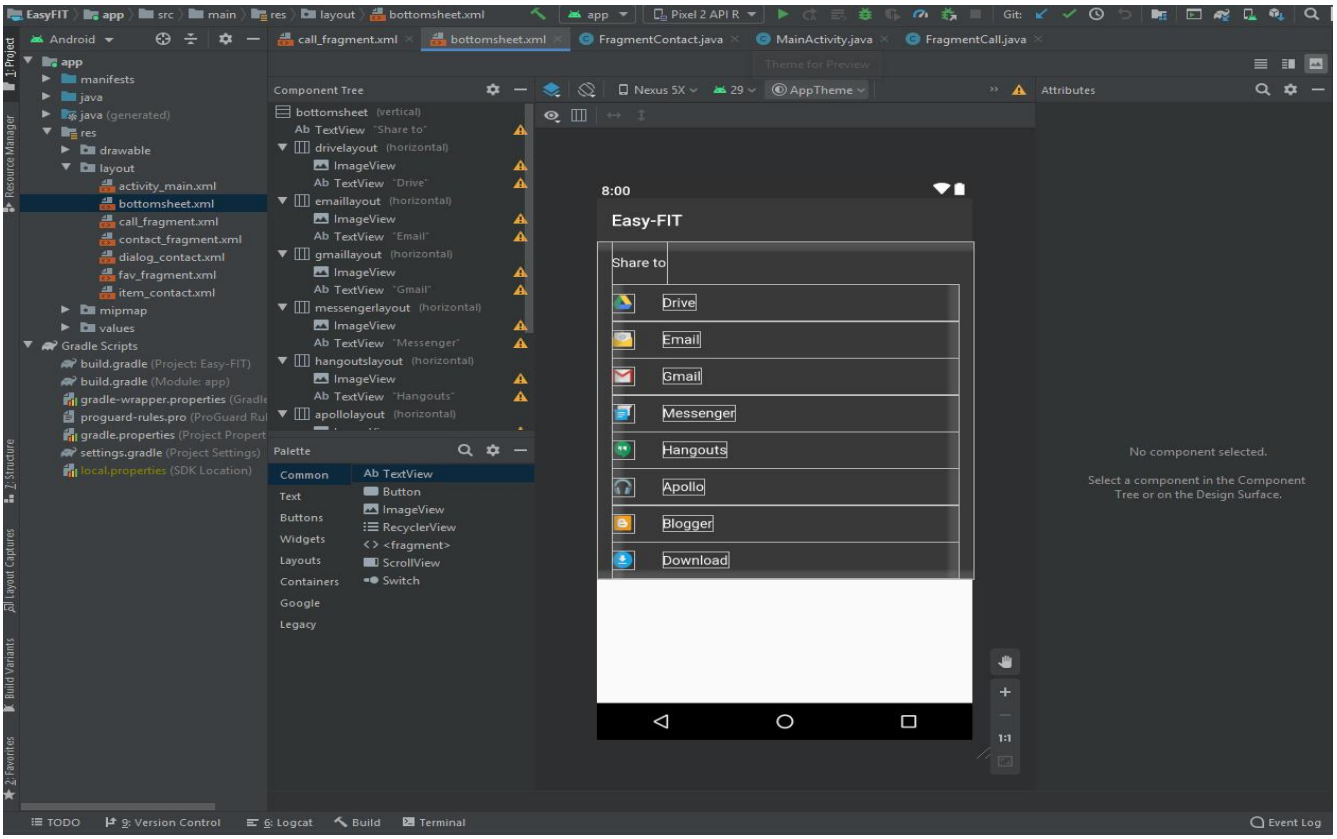
Example from Lab1 application:

As a good example, the coordinator and scroll view is what I used for my project in the news feed.
When you click on the Share Icon the Bottomsheet appears giving you the choice to choose which option you would like to use:

Call_fragment.xml



bottomsheet.xml



Call_fragment.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.coordinatorlayout.widget.CoordinatorLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:background="@color/backgroundfragment">

    <androidx.core.widget.NestedScrollView
        android:padding="20dp"
        android:layout_width="match_parent"
        android:layout_height="match_parent">
        <androidx.cardview.widget.CardView
            app:cardCornerRadius="10dp"
            android:layout_width="match_parent"
            android:layout_height="160dp"
            app:cardBackgroundColor="@color/cardbackground"
            app:cardElevation="10dp">
            <LinearLayout
                android:layout_margin="15dp"
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:orientation="vertical">
                <TextView
                    android:layout_width="wrap_content"
```

```
                    android:layout_height="wrap_content"
                    android:textStyle="bold"
                    android:textSize="16sp"
                    android:textColor="@android:color/white"
                    android:text="U.S. Department of Health and Human Services" />
                <TextView
                    android:layout_marginTop="10dp"
                    android:layout_marginHorizontal="10dp"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:textColor="@android:color/white"
                    android:text="
A healthy eating plan gives your body the nutrients it needs every day while
staying within your daily calorie goal for weight loss. A healthy eating plan also
will lower your risk for heart disease and other health conditions.
A healthy eating plan:
Emphasizes vegetables, fruits, whole grains, and fat-free or low-fat dairy
products
Includes lean meats, poultry, fish, beans, eggs, and nuts
Limits saturated and trans fats, sodium, and added sugars
Controls portion sizes
Calories
To lose weight, most people need to reduce the number of calories they get from
food and beverages (energy IN) and increase their physical activity (energy OUT).
For a weight loss of 1–1 ½ pounds per week, daily intake should be reduced by 500
to 750 calories. In general:
Eating plans that contain 1,200–1,500 calories each day will help most women lose
weight safely.
Eating plans that contain 1,500–1,800 calories each day are suitable for men and
for women who weigh more or who exercise regularly.
Very low calorie diets of fewer than 800 calories per day should not be used
unless you are being monitored by your doctor."/>
                <ImageView
                    android:id="@+id/shareimg"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:src="@drawable/ic_share_black_24dp"
                    android:layout_marginTop="5dp"
                    android:layout_gravity="end"/>
            </LinearLayout>
        </androidx.cardview.widget.CardView>

    </androidx.core.widget.NestedScrollView>
    <include layout="@layout/bottomsheet"/>
</androidx.coordinatorlayout.widget.CoordinatorLayout>
```

bottomsheet.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/bottomsheet"
    android:background="@color/colorPrimary"
```

```
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:paddingRight="16dp"
    android:paddingLeft="16dp"

app:layout_behavior="com.google.android.material.bottomsheet.BottomSheetBehavior"
    app:behavior_hideable="true"
    app:behavior_peekHeight="?android:attr/actionBarSize">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="56dp"
        android:textSize="16sp"
        android:textColor="@android:color/white"
        android:text="Share to"
        android:gravity="center_vertical"/>
    <LinearLayout
        android:background="?android:attr/selectableItemBackground"
        android:focusable="true"
        android:clickable="true"
        android:id="@+id/drivelayout"
        android:gravity="center_vertical"
        android:layout_width="match_parent"
        android:layout_height="48dp"
        android:orientation="horizontal">
        <ImageView
            android:src="@drawable/drive"
            android:layout_width="24dp"
            android:layout_height="24dp"/>
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textSize="16sp"
            android:textColor="@android:color/white"
            android:text="Drive"
            android:layout_marginLeft="32dp"/>
    </LinearLayout>
    <LinearLayout
        android:background="?android:attr/selectableItemBackground"
        android:focusable="true"
        android:clickable="true"
        android:id="@+id/emaillayout"
        android:gravity="center_vertical"
        android:layout_width="match_parent"
        android:layout_height="48dp"
        android:orientation="horizontal">
        <ImageView
            android:src="@drawable/email"
            android:layout_width="24dp"
            android:layout_height="24dp"/>
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textSize="16sp"
            android:textColor="@android:color/white"
            android:text="Email"
            android:layout_marginLeft="32dp"/>
    </LinearLayout>
    <LinearLayout
```

```
        android:background="?android:attr/selectableItemBackground"
        android:focusable="true"
        android:clickable="true"
        android:id="@+id/gmaillayout"
        android:gravity="center_vertical"
        android:layout_width="match_parent"
        android:layout_height="48dp"
        android:orientation="horizontal">
        <ImageView
            android:src="@drawable/gmail"
            android:layout_width="24dp"
            android:layout_height="24dp"/>
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textSize="16sp"
            android:textColor="@android:color/white"
            android:text="Gmail"
            android:layout_marginLeft="32dp"/>
    </LinearLayout>
    <LinearLayout
        android:background="?android:attr/selectableItemBackground"
        android:focusable="true"
        android:clickable="true"
        android:id="@+id/messengerlayout"
        android:gravity="center_vertical"
        android:layout_width="match_parent"
        android:layout_height="48dp"
        android:orientation="horizontal">
        <ImageView
            android:src="@drawable/messenger"
            android:layout_width="24dp"
            android:layout_height="24dp"/>
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textSize="16sp"
            android:textColor="@android:color/white"
            android:text="Messenger"
            android:layout_marginLeft="32dp"/>
    </LinearLayout>
    <LinearLayout
        android:background="?android:attr/selectableItemBackground"
        android:focusable="true"
        android:clickable="true"
        android:id="@+id/hangoutslayout"
        android:gravity="center_vertical"
        android:layout_width="match_parent"
        android:layout_height="48dp"
        android:orientation="horizontal">
        <ImageView
            android:src="@drawable/hangouts"
            android:layout_width="24dp"
            android:layout_height="24dp"/>
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textSize="16sp"
            android:textColor="@android:color/white"
```

```
            android:text="Hangouts"
            android:layout_marginLeft="32dp"/>
    </LinearLayout>
    <LinearLayout
        android:background="?android:attr/selectableItemBackground"
        android:focusable="true"
        android:clickable="true"
        android:id="@+id/apollolayout"
        android:gravity="center_vertical"
        android:layout_width="match_parent"
        android:layout_height="48dp"
        android:orientation="horizontal">
        <ImageView
            android:src="@drawable/apollo"
            android:layout_width="24dp"
            android:layout_height="24dp"/>
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textSize="16sp"
            android:textColor="@android:color/white"
            android:text="Apollo"
            android:layout_marginLeft="32dp"/>
    </LinearLayout>
    <LinearLayout
        android:background="?android:attr/selectableItemBackground"
        android:focusable="true"
        android:clickable="true"
        android:id="@+id/bloggerlayout"
        android:gravity="center_vertical"
        android:layout_width="match_parent"
        android:layout_height="48dp"
        android:orientation="horizontal">
        <ImageView
            android:src="@drawable/blogger"
            android:layout_width="24dp"
            android:layout_height="24dp"/>
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textSize="16sp"
            android:textColor="@android:color/white"
            android:text="Blogger"
            android:layout_marginLeft="32dp"/>
    </LinearLayout>
    <LinearLayout
        android:background="?android:attr/selectableItemBackground"
        android:focusable="true"
        android:clickable="true"
        android:id="@+id/downloadslayout"
        android:gravity="center_vertical"
        android:layout_width="match_parent"
        android:layout_height="48dp"
        android:orientation="horizontal">
        <ImageView
            android:src="@drawable/downloads"
            android:layout_width="24dp"
            android:layout_height="24dp"/>
        <TextView
```

```
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textSize="16sp"
            android:textColor="@android:color/white"
            android:text="Download"
            android:layout_marginLeft="32dp"/>
    </LinearLayout>
</LinearLayout>
```

- ***3 - Study Android fundamental concepts:***

  - ***What programming languages you can use for Android app development?***

Programming in Android application gives us many options to choose from, including Java,Kotlin,C++ and even HTML/CSS/Javascrip,Python,C# to some extent.

Most of the languages mentioned above have very obvious and big limitations and problems when it comes to developing a project. That is why the only best 2 options are java and kotlin.

JAVA

the main programing language for android development is JAVA, and it is the most recommended to be used because it has a lot of support and will be very helpful when it comes to going against errors (Which usually are a lot) and is also the most supported language by Google. On the negative side JAVA can be very complex for beginners sometimes especially with concepts like constructors, null pointer exceptions, concurrency, checked exceptions, etc. Also, not to forget the Android Software Development Kit (SDK) increases the complexity to a new level.

Kotlin

Kotlin is also a good option for android development because it is a cross-platform programming language that can run on the Java Virtual Machine and was even introduced as “official” Java language in 2017”. Kotlin is much simpler for beginners to try as compared to Java, since it removed the necessity of ending every line with a semicolon in addition to a few more stuff, which makes it an “entry point” for Android App Development.

  - ***What programming languages you can use for Android app development?***

APK file is the package file format that the operating system of android use for distribution and installation of mobile apps, mobile games and middleware.

❖ *How Android system runs apps?*

Android runs its applications in a virtual machine, each application has its own and only runs when the component of an application needs to be run and stops it as soon as its not needed or the system runs out of memory space.

❖ *Name four types of Android components. Describe each.*

These are four Android app components:

Services

A service is mainly a background process that can be something that take long time to execute or is ongoing.

Content Providers

A content provider is a component for managing a data set. This data set can be private to your application or can be shared with other applications able to modify or query the data.

Broadcast Receivers

A broadcast receiver is a component that enables the system to deliver events to the app outside of a regular user flow, allowing the app to respond to system-wide broadcast announcements.

Fragments

With fragments, we can divide the parts of our user interface into sections and reuse those sections in more than one screen of our app. This saves us from the task of implementing the same visual/interactive elements more than once.

❖ *What is manifest file and what is its purpose?*

A manifest file is an XML document that describes the manifest, or package contents of our application. Manifest file is mainly known for contains the name of the class that holds the main function, among various classes in the package

❖ *What are resources? Why they are needed?*

There are many more items which you use to build a good Android application. Apart from coding for the application, you take care of various other resources like static content that your code uses, such as bitmaps, colors, layout definitions, user interface strings, animation instructions, and more. These resources are always maintained separately in various sub-directories under res/ directory of the project.

- ***4 - UI Hierarchies (Layouts): Check Attached files for the answers!***

  - ***Analyze the code you wrote. What it does and why?***

In the code we implemented three button for adding,editing and removing. These buttons are included in a single linear layout horizontally to match the requirements of the exercise.

```
<LinearLayout
    android:id="@+id/LinearLayout1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:orientation="horizontal">
    <Button
        android:id="@+id/add_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Add" />
    <Button
        android:id="@+id/edit_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Edit" />
    <Button
        android:id="@+id/remove_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="fill_vertical"
        android:text="Remove" />
</LinearLayout>
```

In addition, there are Edittext and Listview items. They are both aligned vertically following the parent Linear layout attribute highlighted below:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    tools:context=".MainActivity"
    android:orientation="vertical">
```

The list view is used to list all countries vertically according to the layout attributes and are added in the Mainactivity.java

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

This part of the code means that what ever is included in between will be executed when the application is lunched.

```
final String[] COUNTRIES = new String[]  {
    "Afghanistan","Albania","Algeria","American Samoa", "Andorra",
        "Angola","Anguilla","Antarctica","Antigua and barbuda","Argentina",
        "Armenia","Aruba","Australia","Austria","Azerbaijan",
};
```

This part here is a simple declaration of a string array that will be used to fill our ListView with country name later.

```
ListView myListView = (ListView) findViewById(R.id.country_list_view);
final ArrayAdapter<String> aa;
aa = new ArrayAdapter<String>(this,android.R.layout.simple_list_item_1,COUNTRIES);
myListView.setAdapter(aa);
```

Finally, this block of code takes our ListView by ID, then set up the type of the listview and affects it with our previous StringArray of countries using the adapter aa.