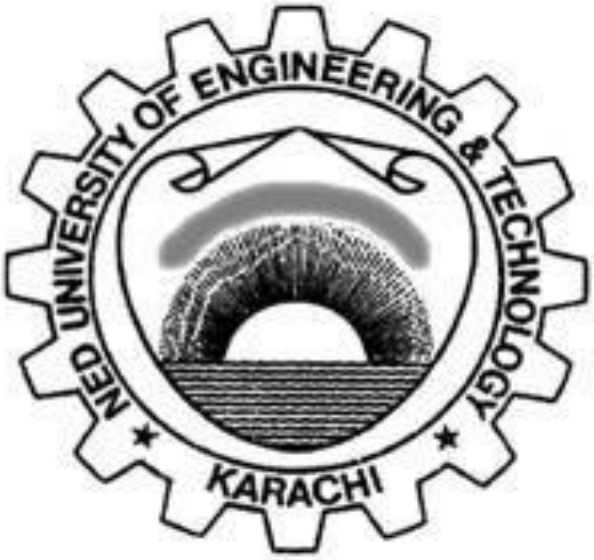
Practical Workbook
CS-219
Computer Engineering Workshop



Name : MUHAMMAD FASIHULLAH

Year : 2021

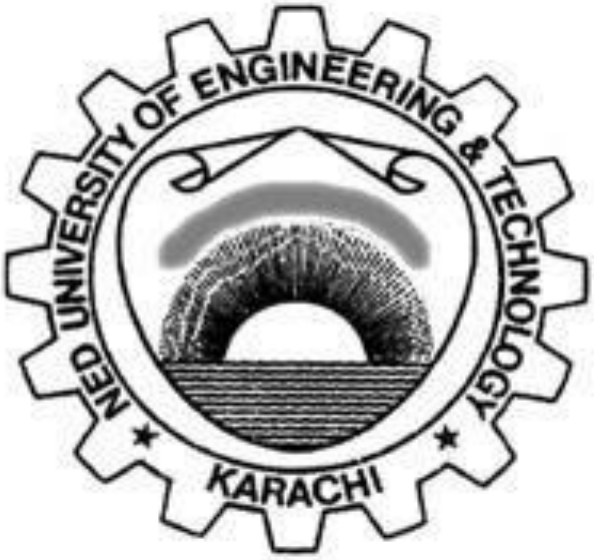
Batch : 2019

Roll No : CS-19042

Department: COMPUTER & INFO. SYS ENG

Department of Computer & Information Systems Engineering
NED University of Engineering & Technology

Practical Workbook
CS-219
Computer Engineering Workshop



Prepared by:

Ms. Mehwish Raza

Department of Computer & Information Systems Engineering
NED University of Engineering & Technology

INTRODUCTION

This workbook has been compiled to assist the conduct of practical classes for CS-319 Computer Engineering Workshop. Labs are designed with an aim to provide students an opportunity to learn basic Android programming concepts using Java programming language.

Android is an operating system and programming platform developed by Google for smartphones and other mobile devices (such as tablets). It can run on many different devices from many different manufacturers. Apps are developed for a variety of reasons: addressing business requirements, building new services, creating new businesses, and providing games and other types of content for users. Developers choose to develop for Android in order to reach the majority of mobile device users. As the world's most popular mobile platform, Android powers hundreds of millions of mobile devices in more than 190countries around the world. It has the largest installed base of any mobile platform and is still growing fast.

The Lab manual starts with understanding development process for building simple Android Application, exploring the project structure, manifest file and gradle build system. The next lab session covers the basic of Java programming language, since Java is the base language for Android development. In the next lab sessions students will learn to build a complete delightful interactive android application, which is followed by testing, debugging and adding support libraries in the project as per requirement. The next lab sessions covers different advanced views, menu styles and content picker to improve the user experience. Lab 8 help students to learn localization, resizing and repositioning of content displayed on screen.

CONTENT

Lab Session 01

Understanding development process for building Android Application

Android is a mobile operating system developed by Google. It is based on a modified version of the Linux kernel and other open source software, and is designed primarily for touchscreen mobile devices such as smartphones and tablets.

Android Studio is the official integrated development environment (IDE) for Google's Android operating system, built on JetBrain IntelliJ IDEA software and designed specifically for Android Development. It is available for download on Windows, macOS and Linux based operating systems. It is a replacement for the Eclipse Android Development Tools (ADT) as the primary IDE for native Android application development.

Versions Android 1.0 and 1.1 were not released under specific code names, although Android 1.1 was unofficially known as Petit Four.

Android code names are confectionery-themed and have been in alphabetical order since 2009's Android 1.5 Cupcake. The most recent version of Android is Android 9 Pie, which was released in August 2018.

On August 22, 2019, Google announced the end of the confectionery theming scheme with the upcoming release of Android 10.

Code name	Version number	API level
No Codename	1.0	1
Petit Four (internal)	1.1	2
Cupcake	1.5	3
Donut	1.6	4
Eclair	2.0 – 2.1	5 – 7
Froyo	2.2 – 2.2.3	8
Gingerbread	2.3 – 2.3.7	9 – 10
Honeycomb	3.0 – 3.2.6	11 – 13
Ice Cream Sandwich	4.0 – 4.0.4	14 – 15
Jelly Bean	4.1 – 4.3.1	16 – 18
KitKat	4.4 – 4.4.4	19 – 20
Lollipop	5.0 – 5.1.1	21 – 22
Marshmallow	6.0 – 6.0.1	23
Nougat	7.0 – 7.1.2	24 – 25
Oreo	8.0 – 8.1	26 – 27
Pie	**9.0**	28
Queen Cake (internal)	10.0	

Table 1.1

Legends: Older Versions Older versions but supported Latest version Latest preview version

In this lab, you will implement the "Hello World" app to verify that Android studio is correctly installed and learn the basics of developing with Android Studio.

1. Launch Android Studio if it is not already opened.
2. In the main **Welcome to Android Studio** window, click "Start a new Android Studio project".
3. In the **New Project** window, give your application an **Application Name**, such as "Hello World".
4. Verify the Project location, or choose a different directory for storing your project.
5. Choose a unique **Company Domain**.

Apps published to the Google Play Store must have a unique package name. Since domains are unique, Prepending your app's name with your or your company's domain name is going to result in a unique package name. If you are not planning to publish your app, you can accept the default example domain. Be aware that changing the package name of your app later is extra work.

6. Verify that the default **Project location** is where you want to store your Hello World app and other Android Studio projects, or change it to your preferred directory. Click Next.
7. On the **Target Android Devices** screen, "Phone and Tablet" should be selected. And you should ensure that API 15: Android 4.0.3 IceCreamSandwich is set as the **Minimum SDK**. (Fix this if necessary.) Choosing this API level makes your "Hello World" app compatible with 97% of Android devices active on the Google Play Store.
8. Click **Next**.
9. If your project requires additional components for your chosen target SDK, Android Studio will install them automatically. Click **Next**.
10. **Customize the Activity** window. Every app needs at least one activity. An activity represents a single screen with a user interface and Android Studio provides templates to help you get started. For the Hello World project, choose the simplest template (as of this writing, the "Empty Activity" project template is the simplest template) available.
11. It is a common practice to call your main activity MainActivity. This is not a requirement.
12. Make sure the **Generate Layout file** box is checked (if visible).
13. Make sure the **Backwards Compatibility (App Compat)** box is checked.
14. Leave the **Layout Name** as activity_main. It is customary to name layouts after the activity they belong to. Accept the defaults and click **Finish**.

Project Structure and layout

In the Project > Android view of your previous task, there are three top-level folders below your **app** folder: **manifests**, **java**, and **res**.

1. Expand the **manifests** folder.

This folder contains **AndroidManifest.xml.** This file describes all of the components of your Android app and is read by the Android run-time system when your program is executed.

2. Expand the **java** folder. All your Java language files are organized in this folder. The **java** folder contains three subfolders:

- ✓ **com.example.hello.helloworld (or the domain name you have specified):** All the files for a package are in a folder named after the package. For your Hello World application, there is one package and it only contains MainActivity.java (the file extension may be omitted in the Project view).
- ✓ **com.example.hello.helloworld(androidTest):** This folder is for your instrumented tests, and starts out with a skeleton test file.
- ✓ **com.example.hello.helloworld(test):** This folder is for your unit tests and starts out with an automatically created skeleton unit test file.

3. Expand the **res** folder. This folder contains all the resources for your app, including images, layout files, strings, icons, and styling. It includes these subfolders:

- ✓ **drawable**. Store all your app's images in this folder.
- ✓ **layout**. Every activity has at least one layout file that describes the UI in XML. For Hello World, this folder contains activity_main.xml.
- ✓ **mipmap**. Store your launcher icons in this folder. There is a sub-folder for each supported screen density. Android uses the screen density, that is, the number of pixels per inch to determine the required image resolution. Android groups all actual screen densities into generalized densities, such as medium (mdpi), high (hdpi), or extra-extraextra-high (xxxhdpi). The ic_launcher.png folder contains the default launcher icons for all the densities supported by your app.
- ✓ **values**. Instead of hardcoding values like strings, dimensions, and colors in your XML and Java files, it is best practice to define them in their respective values file. This makes it easier to change and be consistent across your app.

4. Expand the **values** subfolder within the res folder. It includes these subfolders:

- ✓ **colors.xml**. Shows the default colors for your chosen theme, and you can add your own colors or change them based on your app's requirements.
- ✓ **dimens.xml**. Store the sizes of views and objects for different resolutions.
- ✓ **strings.xml**. Create resources for all your strings. This makes it easy to translate them to other languages.
- ✓ **styles.xml**. All the styles for your app and theme go here. Styles help give your app a consistent look for all UI elements.

The Gradle build system

Android Studio uses Gradle as its build system. As you progress through the next lab sessions you will learn more about gradle and what you need to build and run your apps.
1. Expand the **Gradle Scripts** folder. This folder contains all the files needed by the build system.
2. Look for the **build.gradle(Module:app)** file. When you are adding app-specific dependencies, such as using additional libraries, they go into this file.

Create a virtual device (emulator)

Android Virtual Device (AVD) manager is used to create a virtual device or emulator that simulates the configuration for a particular type of Android device.
Using the AVD Manager, you define the hardware characteristics of a device and its API level, and save it as a virtual device configuration.
When you start the Android emulator, it reads a specified configuration and creates an emulated device that behaves exactly like a physical version of that device, but it resides on your computer.
Why: With virtual devices, you can test your apps on different devices (tablets, phones) with different API levels to make sure it looks good and works for most users. You do not need to depend on having a physical device available for app development.

In order to run an emulator on your computer, you have to create a configuration that describes the virtual device.

In Android Studio, select Tools > Android > AVD Manager, or click the AVD Manager icon in the toolbar.
2. Click the **+Create Virtual Device**…. (If you have created a virtual device before, the window shows all of your existing devices and the button is at the bottom.)

- The Select Hardware screen appears showing a list of preconfigured hardware devices. For each device, the table shows its diagonal display size (Size), screen resolution in pixels (Resolution), and pixel density (Density).

- For the Nexus 5 device, the pixel density is xxhdpi, which means your app uses the launcher icons in the xxhdpi folder of the mipmap folder. Likewise, your app will use layouts and drawables from folders defined for that density as well.

3. Choose the Nexus 5 hardware device and click **Next**.
4. On the **System Image** screen, from the **Recommended** tab, choose which version of the Android system to run on the virtual device. You can select the latest system image. There are many more versions available than shown in the **Recommended** tab. Look at the **x86 Images** and **Other Images** tabs to see them.
5. If a **Download** link is visible next to a system image version, it is not installed yet, and you need to download it. If necessary, click the link to start the download, and click **Finish** when it's done.
6. On **System Image** screen, choose a system image and click **Next**.
7. Verify your configuration, and click **Finish**.

Android Manifest.xml

Every app includes an Android Manifest file (AndroidManifest.xml).The manifest file contains essential information about your app and presents this information to the Android runtime system. Android must have this information before it can run any of your app's code. As your apps add more functionality and the user experience becomes more engaging and interactive, the AndroidManifest.xml file contains more and more information. In later lab sessions we will modify this file to add features and feature permissions.

Explore the build.gradle file

1. In your project hierarchy, find **Gradle Scripts** and expand it. There several build.gradle files. One with directives for your whole project, and one for each app module. The module for your app is called "app". In the Project view, it is represented by the **app** folder at the top-level of the Project view.

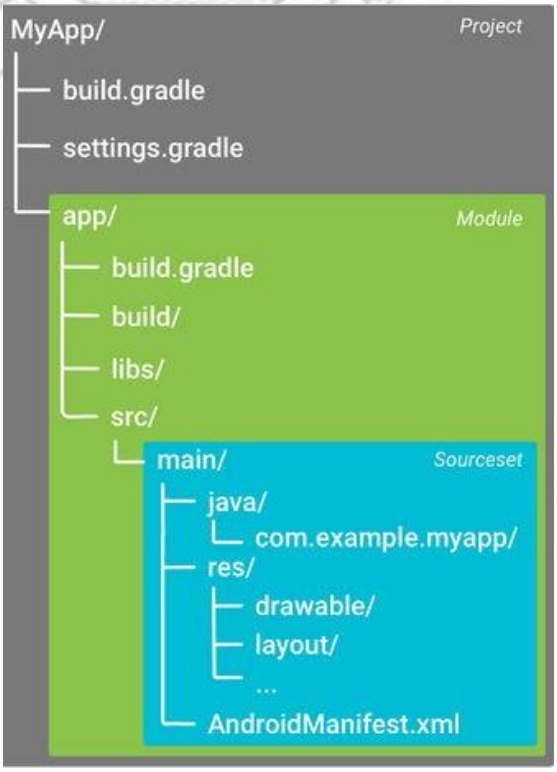


Figure 1.1

- The top-level build.gradle file, located in the root project directory, defines build configurations that apply to all modules in your project. By default, the top-level build file uses the buildscript block to define the Gradle *repositories* and *dependencies* that are common to all modules in the project.
- The module-level build.gradle file, located in each *project*/*module*/ directory, allows you to configure build settings for the specific module it is located in. Configuring these build settings allows you to provide custom packaging options, such as additional build types and product flavors, and override settings in the main/ app manifest or top-level build.gradle file.

Exercise

Create a new project in android studio. Display your name and roll no as label on the first activity. Take a screenshot of your finished app and attach the print out.

EXERCISE:

```
package com.example.lab1;

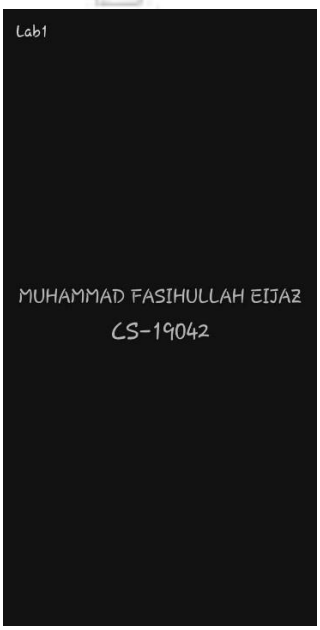
import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

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

OUTPUT:



Lab Session 02

Java Basics

There are a number of ways to create apps for Android devices, but the recommended method for most developers is to write native apps using Java and the Android SDK.

Java is an Object-Oriented Language. As a language that has the Object-Oriented feature, Java supports the following fundamental concepts −

- Polymorphism
- Inheritance
- Encapsulation
- Abstraction
- Classes
- Objects
- Instance
- Method
- Message Passing

When we consider a Java program, it can be defined as a collection of objects that communicate via invoking each other's methods. Let us now briefly look into what do class, object, methods, and instance variables mean.

- **Object** − Objects have states and behaviors. Example: A dog has states - color, name, breed as well as behavior such as wagging their tail, barking, eating. An object is an instance of a class.
- **Class** − A class can be defined as a template/blueprint that describes the behavior/state that the object of its type supports.
- **Methods** − A method is basically a behavior. A class can contain many methods. It is in methods where the logics are written, data is manipulated and all the actions are executed.
- **Instance Variables** − Each object has its unique set of instance variables. An object's state is created by the values assigned to these instance variables.

Basic Syntax

About Java programs, it is very important to keep in mind the following points.

- **Case Sensitivity** − Java is case sensitive, which means identifier **Hello** and **hello** would have different meaning in Java.
- **Class Names** − For all class names the first letter should be in Upper Case. If several words are used to form a name of the class, each inner word's first letter should be in Upper Case. Example: *class MyFirstJavaClass*
- **Method Names** − All method names should start with a Lower Case letter. If several words are used to form the name of the method, then each inner word's first letter should be in Upper Case. Example: *public void myMethodName()*
- **public static void main(String args[])** − Java program processing starts from the main() method which is a mandatory part of every Java program.

Data Types in Java
Following are the explicit data types used in Java

- Boolean
- Byte
- Char
- Short
- Int
- Long
- Float
- double

Variables in Java
There are three types of variables in Java:

- Local Variables
- Instance Variables
- Static Variables

Let us now learn about each one of these variables in detail.

1. **Local Variables**: A variable defined within a block or method or constructor is called local variable.
    - Variables defined inside methods, constructors or blocks are called local variables.
    - These variables are created when the block in the function is called and destroyed after exiting from the block or when the call returns from the function.
    - The scope of these variables exists only within the block in which the variable is declared. i.e. we can access these variable only within that block.
    - Initialization of Local Variable is Mandatory.
2. **Instance Variables**: Instance variables are non-static variables and are declared in a class outside any method, constructor or block.
    - As instance variables are declared in a class, these variables are created when an object of the class is created and destroyed when the object is destroyed.
    - Unlike local variables, we may use access specifiers for instance variables. If we do not specify any access specifier then the default access specifier will be used.
    - Initilisation of Instance Variable is not Mandatory. Its default value is 0
    - Instance Variable can be accessed only by creating objects.
3. **Static Variables**: Static variables are also known as Class variables.

    - These variables are declared similarly as instance variables; the difference is that static variables are declared using the static keyword within a class outside any method constructor or block.
    - Unlike instance variables, we can only have one copy of a static variable per class irrespective of how many objects we create.
    - Static variables are created at the start of program execution and destroyed automatically when execution ends.
    - Initilisation of Static Variable is not mandatory. Its default value is 0
    - To access static variables, we need not create an object of that class, we can simply access the variable as class_name.variable_name;

Collections in Java

- The **Collection in Java** is a framework that provides architecture to store and manipulate the group of objects.
- Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion.
- Java Collection means a single unit of objects.
- Java Collection framework provides many interfaces (Set, List, Queue, Deque) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet).

Strings in Java

Strings are defined as an array of characters. The difference between a character array and a string is the string is terminated with a special character '\0'. Below is the basic syntax for declaring a string in **Java programming** language.

Syntax:

<String_Type> <string_variable> = "<sequence_of_string>";

There are two ways to create string in Java:

- ***String literal***
  String s = "abc";
- **Using *new* keyword**
  String s = new String ("abc");

Arrays in Java

An array is a group of like-typed variables that are referred to by a common name. Array can contains primitives data types as well as objects of a class depending on the definition of array.

Example :

- `int intArray[]; or  int[] intArray; //declaring array`

- ```
  int intArray[];
  intArray = new int[20];  // allocating memory to array OR
  int[] intArray = new int[20]; // combining both statements in one
  ```

Java Package

In simple words, it is a way of categorizing the classes and interfaces. When developing applications in Java, hundreds of classes and interfaces will be written, therefore categorizing these classes is a must as well as makes life much easier.

Import Statements

In Java if a fully qualified name, which includes the package and the class name is given, then the compiler can easily locate the source code or classes. Import statement is a way of giving the proper location for the compiler to find that particular class.

For example, the following line would ask the compiler to load all the classes available in directory java_installation/java/io – `import java.io.*;`
  ```

Decision Making in Java

Following are the Java's selection statements:

- if
- if-else
- nested-if
- if-else-if
- switch-case
- jump – break, continue, return

1. **if**:
```
if(condition)

{
   // Statements to execute if
   // condition is true}
```
2. **if-else**:
```
if (condition)

{
    // Executes this block if
    // condition is true
}
else
{
    // Executes this block if
    // condition is false
}
```

3. **if-else-if ladder:**
```
 if (condition)
     statement;
 else if (condition)
     statement;
 .
 .
 else    statement;
```
4. **switch-case** The switch statement is a multiway branch statement. It provides an easy way to dispatch execution to different parts of code based on the value of the expression.

   Syntax:
```
switch (expression)
{
  case value1:
    statement1;
    break;
  case value2:
    statement2;
    break;
  .
  .
  case valueN:
    statementN;
    break;
  default:    statementDefault;}
```

5. **jump:** Java supports three jump statement: **break, continue** and **return**. These three statements transfer control to other part of the program.

Loops in Java

Java provides three ways for executing the loops. While all the ways provide similar basic functionality, they differ in their syntax and condition checking time.

1. **while loop:**

```
while (boolean condition)
{
   loop statements...
}
```

2. **for loop:**

```
for (initialization condition; testing condition;increment/decrement)
{
    statement(s)
}
```

3. **Enhanced For loop**

   Java also includes another version of for loop introduced in Java 5. Enhanced for loop provides a simpler way to iterate through the elements of a collection or array. Also note that the object/variable is immutable when enhanced for loop is used i.e it ensures that the values in the array can not be modified, so it can be said as read only loop where you can't update the values as opposite to other loops where values can be modified.

```
for (T element:Collection obj/array)
{
    statement(s)
}
```

4. **do while:**

```
do
{
  statements..
}   while (condition);
```

Access Modifiers in Java

As the name suggests access modifiers in Java helps to restrict the scope of a class, constructor , variable , method or data member. There are four types of access modifiers available in java:

1. Default – No keyword required
2. Private
3. Protected
4. Public

	default	private	protected	public
Same Class	Yes	Yes	Yes	Yes
Same package subclass	Yes	No	Yes	Yes
Same package non-subclass	Yes	No	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes

Table 2.1

Inheritance

We use **extends** keyword in java to implement inheritance. Below is a simple example of inheritance in java.

```
class Subclass-name extends Superclass-name
{
   //methods and fields
}
```

A very important fact to remember is that Java does not support multiple inheritance. This means that a class cannot extend more than one class. Therefore following is illegal −

Example

```
public class extends Animal, Mammal{}
```

Super Keyword: The **super** keyword in Java is a reference variable which is used to refer immediate parent class object. Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

Usage of Java super Keyword

1. super can be used to refer immediate parent class instance variable.
2. super can be used to invoke immediate parent class method.
3. super() can be used to invoke immediate parent class constructor.

Final keyword: The **final** keyword in java is used to restrict the user. The java final keyword can be used in many contexts. Final can be variable, method or class.

- **Final Variable:** Final keyword is used to make a variable as a constant. This is similar to constant in other language.
- **Final Method:** It makes a method final, meaning that sub classes cannot override this method. The compiler checks and gives an error if you try to override the method. When we want to restrict overriding, and then make a method as a final.
- **Final Class:** It makes a class final, meaning that the class cannot be inheriting by other classes. When we want to restrict inheritance then make class as a final.

A final variable that has no value, it is called blank final variable or uninitialized final variable. It can be initialized in the constructor only. The blank final variable can be static also which will be initialized in the static block only.

Polymorphism

There are two types of polymorphism in Java: compile-time polymorphism and runtime polymorphism. We can perform polymorphism in java by method overloading (compile-time polymorphism) and method overriding (runtime polymorphism).
If you overload a static method in Java, it is the example of compile time polymorphism.

Abstraction in Java

Abstraction is a process of hiding the implementation details and showing only functionality to the user. It shows only essential things to the user and hides the internal .You don't know the internal processing about the message delivery.
Abstraction lets you focus on what the object does instead of how it does it.
There are two ways to achieve abstraction in java

1. Abstract class (0 to 100%) –using abstract keyword prior to class or method
2. Interface (100%)- using interface keyword prior to class

Interfaces in Java

An **interface in java** is a blueprint of a class. It has static constants and abstract methods. The interface in Java is *a mechanism to achieve abstraction*. An interface is different from a class in several ways, including

- You cannot instantiate an interface.
- An interface does not contain any constructors.
- All of the methods in an interface are abstract.
- An interface cannot contain instance fields. The only fields that can appear in an interface must be declared both static and final.
- An interface is not extended by a class; it is implemented by a class.
- An interface can extend multiple interfaces.

An interface is declared by using the interface keyword. It provides total abstraction; means all the methods in an interface are declared with the empty body, and all the fields are public, static and final by default.
A class uses the **implements** keyword to implement an interface. The implements keyword appears in the class declaration following the extends portion of the declaration. A class that implements an interface must implement all the methods declared in the interface.

Exercises

1) Write a java program for abstract class to find areas of different shapes.
2) Write a Java program to create a new array list, add some colors (string) and print out the collection.
3) Write a Java program to remove the third element from the array list.
4) Write a Java program to test if a given string contains the specified sequence of char values.

EXERCISE 1:

```
import java.util.Scanner;
import java.lang.Math;
abstract class shape{
    public abstract double area();
}
class circle extends shape{
    public double area(){
        double r;
        Scanner sc= new Scanner(System.in);
        System.out.println("Enter the radius of circle in cm:");
        r=sc.nextDouble();
        sc.close();
        double areaoftheshape;
        areaoftheshape=Math.PI*r*r;
        return areaoftheshape;
    }
}
```

```
class square extends shape{
    public double area(){
        double r;
        Scanner sc= new Scanner(System.in);
        System.out.println("Enter any side of the square in cm:");
        r=sc.nextDouble();
        sc.close();
        double areaoftheshape;
        areaoftheshape=r*r;
        return areaoftheshape;
}
}
class rectangle extends shape{
    public double area(){
        double l, b;
        Scanner sc= new Scanner(System.in);
        System.out.println("Enter the length of rectangle in cm:");
        l=sc.nextDouble();
        System.out.println("Enter the breadth of rectangle in cm:");
        b=sc.nextDouble();
        sc.close();
        double areaoftheshape;
        areaoftheshape=l*b;
        return areaoftheshape;
}
}
public class lab02task1 {
    public static void main(String[] args){
        int choice;
        Scanner sd= new Scanner(System.in);
        System.out.println("Type sequence number to figure out the area of the following
shape:\n1.Cirlce\n2.Rectangle\n3.Square");
        choice=sd.nextInt();
        switch (choice){
            case 1:
                circle circle1= new circle();
                System.out.println("The area of cirlce is:"+circle1.area()+" cm");
                break;
            case 2:
                rectangle rectangle1= new rectangle();
                System.out.println("The area of rectangle is:"+rectangle1.area()+" cm");
                break;
            case 3:
                square square1= new square();
                System.out.println("The area of square is:"+square1.area()+" cm");
                break;
            default:
                System.out.println("You have entered the wrong option or the shape don't exist.");
        }



    }

}
```

EXERCISE 2:

```
import java.util.Scanner;
import java.util.ArrayList;
public class lab02task2 {
    public static void main(String[] args){
        ArrayList<String> colors = new ArrayList<String>();
        Scanner sc= new Scanner(System.in);
        String i;
        int choice;
        do{
            System.out.println("Enter the color name:");
            i=sc.next();
            colors.add(i);
            System.out.println("If you are done with adding colors type 1 or else type anyother
number.");
            choice=sc.nextInt();
            if(choice == 1 ){
                break;
            }
        }
        while(true);
        sc.close();
        System.out.println(colors);
    }

}
```

EXERCISE 3:

```
import java.util.Scanner;
import java.util.ArrayList;
public class lab02task3 {
    public static void main(String[] args){
        ArrayList<String> colors = new ArrayList<String>();
        Scanner sc= new Scanner(System.in);
        Scanner sd= new Scanner(System.in);
        String i;
        String change;
        int choice;
        do{
            System.out.println("Enter the color name:");
            i=sc.next();
            colors.add(i);
            System.out.println("If you are done with adding colors type 1 or else type anyother
number.");
            choice=sd.nextInt();
            if(choice == 1 ){
                break;
            }
        }
        while(true);
        System.out.println(colors);
        System.out.println("You are now changing the 3rd element of the list!\nEnter anything that you
```

```
want to be in the 3rd place of the list.");
        change=sc.next();
        if (colors.size() >= 3){
            colors.set(2, change);
        }
        else{
            System.out.println("The 3rd element don't exist!");
        }
        System.out.println(colors);
        sc.close();
        sd.close();

    }

}
```

EXERCISE 4:

```
import java.util.Scanner;
public class lab02task4 {
    public static void main(String[] args) {
        String word1,find;
        boolean check;
        Scanner sc= new Scanner(System.in);
        System.out.println("Enter the word which you want to be tested:");
        word1= sc.next();
        System.out.println("Enter the string you want to test:");
        find= sc.next();
        sc.close();
        if(check=word1.contains(find)){
            System.out.println("Yes it exist.");
        }
        else{
            System.out.println("No it does'nt exist.");
        }


     }

}
```

Lab Session 03

Design layouts

An activity displays the user interface of your application, which may contain widgets like buttons, labels, text boxes, and so on. Typically, you defi ne your UI using an XML fi le (e.g., the main.xml fi le located in the res/layout folder), which may look like this:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="vertical"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
>
<TextView
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="@string/hello"
/>
</LinearLayout>
```

During run time, you load the XML UI in the onCreate() event handler in your Activity class, using the setContentView() method of the Activity class:

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

Views and Viewgroups

An activity contains Views and ViewGroups. A view is a widget that has an appearance on screen. Examples of views are buttons, labels, and text boxes. A view derives from the base class android.view.View.

Android supports the following ViewGroups:

- LinearLayout
- AbsoluteLayout
- TableLayout
- RelativeLayout
- FrameLayout
- ScrollView
- AdapterView
-

LinearLayout

The LinearLayout arranges views in a single column or a single row. Child views can be arranged either vertically or horizontally. To see how LinearLayout works, consider the following elements typically contained in the main.xml file:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="vertical"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
>
<TextView
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="@string/hello"
/>
</LinearLayout>
```

In the main.xml file, observe that the root element is <LinearLayout> and it has a <TextView> element contained within it. The <LinearLayout> element controls the order in which the views contained within it appear.

Each View and ViewGroup has a set of common attributes, some of which are described as following.

Attribute	**Description**
layout_width	Specifies the width of the View or ViewGroup
layout_height	Specifies the height of the View or ViewGroup
layout_marginTop	Specifies extra space on the top side of the View or ViewGroup
layout_marginBottom	Specifies extra space on the bottom side of the View or ViewGroup
layout_marginLeft	Specifies extra space on the left side of the View or ViewGroup
layout_marginRight	Specifies extra space on the right side of the View or ViewGroup
layout_gravity	Specifies how child Views are positioned
layout_weight	Specifies how much of the extra space in the layout should be allocated to the View
layout_x	Specifies the x-coordinate of the View or ViewGroup
layout_y	Specifies the y-coordinate of the View or ViewGroup

Table 3.1

For example, the width of the <TextView> element fi lls the entire width of its parent (which is the screen in this case) using the fill_parent constant. Its height is indicated by the wrap_content constant, which means that its height is the height of its content (in this case, the text contained within it). If you don't want to have the <TextView> view occupy the entire row, you can set its layout_width attribute to wrap_content, like this:

```
< TextView
android:layout_width=”wrap_content
android:layout_height=”wrap_content”
android:text=”@string/hello”/>
```

AbsoluteLayout

The AbsoluteLayout enables you to specify the exact location of its children. AbsoluteLayout has been deprecated since Android 1.5 (although it is still supported in the current version). You should avoid using the AbsoluteLayout in your UI, as it is not guaranteed to be supported in future versions of Android.

TableLayout

The TableLayout groups views into rows and columns. You use the <TableRow> element to designate a row in the table. Each row can contain one or more views. Each view you place within a row forms a cell. The width of each column is determined by the largest width of each cell in that column.
Consider the content of main.xml shown here:

```
<?xml version=”1.0” encoding=”utf-8”?>
<TableLayout
xmlns:android=”http://schemas.android.com/apk/res/android”
android:layout_height=”fill_parent”
android:layout_width=”fill_parent”
>
<TableRow>
<TextView
android:text=”User Name:”
android:width =”120px”
/>
<EditText
android:id=”@+id/txtUserName”
android:width=”200px” />
</TableRow>
<TableRow>
<TextView
android:text=”Password:”
/>
<EditText
android:id=”@+id/txtPassword”
android:password=”true”
/>
</TableRow>
<TableRow>
<TextView />
<CheckBox android:id=”@+id/chkRememberPassword”
android:layout_width=”fill_parent”
android:layout_height=”wrap_content”
android:text=”Remember Password”
/>
</TableRow>
<TableRow>
<Button
android:id=”@+id/buttonSignIn”
android:text=”Log In” />
</TableRow>
</TableLayout>
```

RelativeLayout

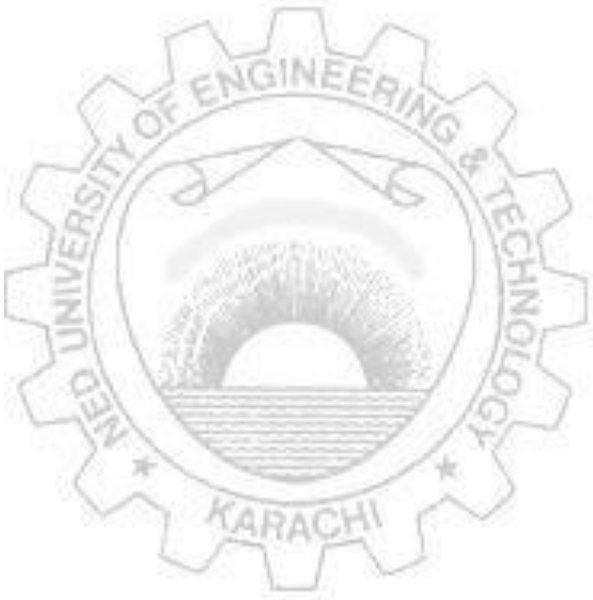
The RelativeLayout enables you to specify how child views are positioned relative to each other. Each view embedded within the RelativeLayout has attributes that enable it to align with another view. These attributes are as follows:

- layout_alignParentTop
- layout_alignParentLeft
- layout_alignLeft
- layout_alignRight
- layout_below
- layout_centerHorizontal

Consider the following main.xml file:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
android:id="@+id/RLayout"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
xmlns:android="http://schemas.android.com/apk/res/android"
>
<TextView
android:id="@+id/lblComments"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Comments"
android:layout_alignParentTop="true"
android:layout_alignParentLeft="true"
/>
<EditText
android:id="@+id/txtComments"
android:layout_width="fill_parent"
android:layout_height="170px"
android:textSize="18sp"
android:layout_alignLeft="@+id/lblComments"
android:layout_below="@+id/lblComments"
android:layout_centerHorizontal="true"
/>
<Button
android:id="@+id/btnSave"
android:layout_width="125px"
android:layout_height="wrap_content"
android:text="Save"
android:layout_below="@+id/txtComments"
android:layout_alignRight="@+id/txtComments"
/>
<Button
android:id="@+id/btnCancel"
android:layout_width="124px"
android:layout_height="wrap_content"
android:text="Cancel"
android:layout_below="@+id/txtComments"
android:layout_alignLeft="@+id/txtComments"
/>
```

```
</RelativeLayout>
```



FrameLayout

FrameLayout is designed to block out an area on the screen to display a single item.You use a FrameLayout to stack child views on top of each other, with the most recent child on top of the stack.

In the example below, the TextView is the most recent, so it is automatically placed on top of the ImageView.

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <ImageView
        android:id="@+id/backgroundImage"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:scaleType="centerCrop"
        android:src="@drawable/bitmapie" />

    <TextView
        android:id="@+id/descTextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_vertical"
        android:layout_marginTop="70dp"
        android:background="@android:color/holo_blue_light"
        android:padding="10dp"
        android:text="TextView placed at the top of the Imageview"
        android:textColor="@android:color/white"
        android:textSize="22sp" />

</FrameLayout>
```

Exercise

1) Using Constraint Layout, design an email login screen in xml that contains following views:
    1) Edit boxes for email id and password
    2) Button for Login
    3) Image view for logo

EXERCISE:

```
package com.example.lab03;

import androidx.appcompat.app.AppCompatActivity;

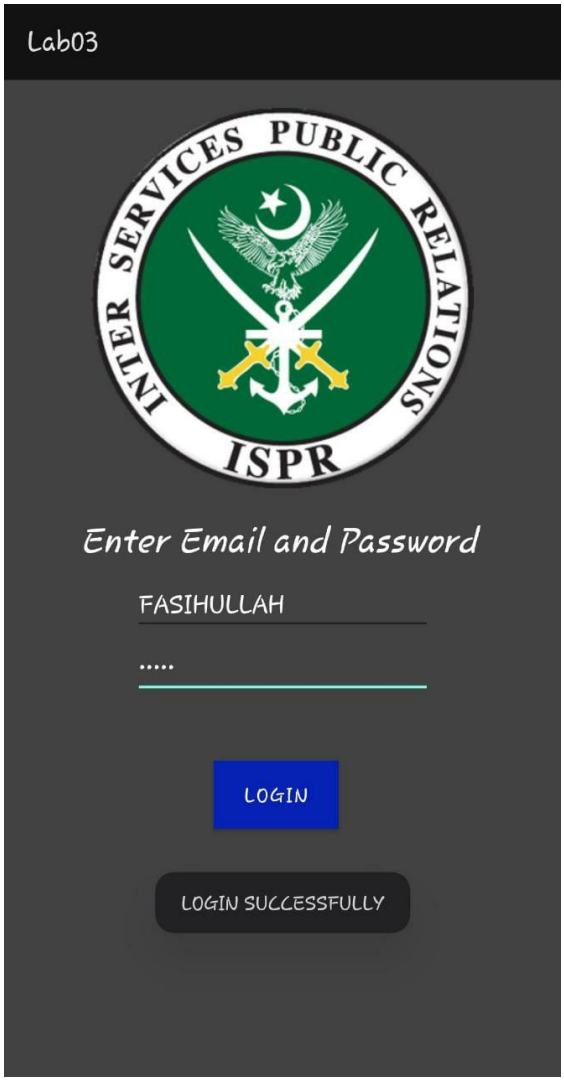
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

    public void login(View view){
        Toast.makeText(this, " LOGIN SUCCESSFULLY ", Toast.LENGTH_SHORT).show();
    }
```

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

OUTPUT:



Lab Session 04

Implementing Activities and Intents

Activity

An activity represents a single screen in your app with an interface the user can interact with. For example, an email app might have one activity that shows a list of new emails, another activity to compose an email, and another activity for reading individual messages. Your app is a collection of activities that you either create yourself, or that you reuse from other apps.

An app usually consists of multiple activities that are loosely bound to each other. Typically, one activity in an application is specified as the "main" activity, which is presented to the user when the app is launched. Each activity can then start other activities in order to perform different actions.

Each time a new activity starts, the previous activity is stopped, but the system preserves the activity in a stack (the "back stack"). When a new activity starts, that new activity is pushed onto the back stack and takes user focus. The back stack abides to the basic "last in, first out" stack mechanism, so, when the user is done with the current activity and presses the Back button, that current activity is popped from the stack (and destroyed) and the previous activity resumes.

Activity Lifecycle

The Activity lifecycle is the set of states an activity can be in, from when it is first created, to each time it is stopped or resumed, to when the system destroys it.

.

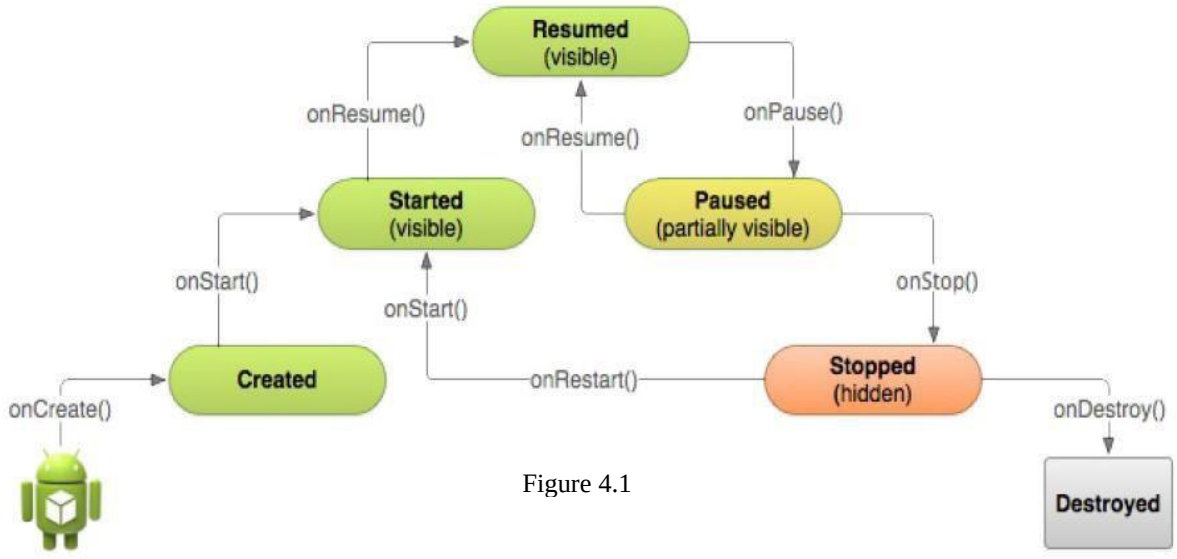


Figure 4.1

Intents

Android activities are started or activated with intent. Intents are asynchronous messages that you can can use in your activity to request an action from another activity (or other app component). You use intents to start one activity from another and to pass data between activities.
There are two kinds of intents: ***explicit*** and ***implicit***.

- An explicit intent is one in which you know the target of that intent, that is, you already know the fully-qualified class name of that specific activity.
- An implicit intent is one in which you do not have the name of the target component, but have a general action to perform.

In this practical you'll learn about explicit intents.

Creating activities

To implement an activity in your app, do the following:

1. Create an activity Java class.
2. Implement a user interface for that activity.
3. Declare that new activity in the app manifest.

When you create a new project for your app, or add a new activity to your app, in Android Studio (with File > New >Activity), template code for each of these tasks is already provided.

Create the activity class

Activities are subclasses of the Activity class, or one of its subclasses. When you create a new project in Android Studio, your activities are, by default, subclasses of the AppCompatActivity class. The AppCompatActivity class is a subclass of Activity that lets you to use up-to-date Android app features such as the action bar and material design, while still enabling your app to be compatible with devices running older versions of Android.
Here is a skeleton subclass of AppCompatActivity:

```
public class MainActivity extends AppCompatActivity {
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
}
}
```

The first task for you in your activity subclass is to implement the standard activity lifecycle callback methods (such as OnCreate()) to handle the state changes for your activity. These state changes include things such as when the activity is created, stopped, resumed, or destroyed.

Implement a user interface

The user interface for an activity is provided by a hierarchy of views, which controls a particular space within the activity's window and can respond to user interaction.

The most common way to define a user interface using views is with an XML layout file stored as part of your app's resources. Defining your layout in XML enables you to maintain the design of your user interface separately from the source code that defines the activity's behavior.
You can also create new views directly in your activity code by inserting new view objects into a ViewGroup, and then passing the root ViewGroup to setContentView(). After your layout has been inflated -- regardless of its source -- you can add more views in Java anywhere in the view hierarchy.

Declare the activity in the manifest

Each activity in your app must be declared in the Android app manifest with the <activity> element, inside <application> . When you create a new project or add a new activity to your project in Android Studio, your manifest is created or updated to include skeleton activity declarations for each activity. Here's the declaration for the main activity.

```
<activity android:name=".MainActivity" >
<intent-filter>
<action android:name="android.intent.action.MAIN" />
<category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>
```

Starting an activity with an explicit intent

To start a specific activity from another activity, use an explicit intent and the startActivity() method. Explicit intents include the fully-qualified class name for the activity or other component in the Intent object. All the other intent fields are optional, and null by default.

For example, if you wanted to start the MySecondActivity to show a specific message in an app, use code like this.

```
Intent messageIntent = new Intent(this, MySecondActivity.class);
startActivity(messageIntent);
```

The Intent constructor takes two arguments for an explicit intent.

1. An application context. In this example, the activity class provides the content (here, this).
2. The specific component to start (MySecondActivity.class).

Use the startActivity() method with the new intent object as the only argument. The startActivity() method sends the intent to the Android system, which launches the MySecondActivity class on behalf of your app. The new activity appears on the screen, and the originating activity is paused.
The started activity remains on the screen until the user taps the back button on the device, at which time that activity closes and is reclaimed by the system, and the originating activity is resumed. You can also manually close the started activity in response to a user action (such as a button click) with the finish() method:

```
public void closeActivity (View view) {
finish();
}
```

Exercises

1. Create an app that sends a message from first activity to second activity. Attach the printout of the two activities.
2. Create an android application that opens a defined url in browser within the device.

EXERCISE 1:

```
package com.example.lab4;

import androidx.appcompat.app.AppCompatActivity;

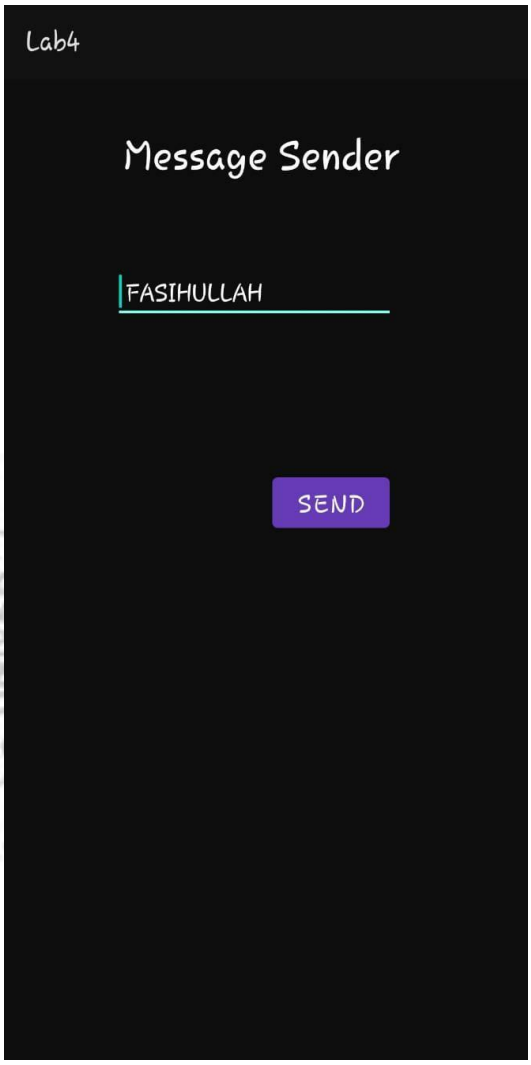
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

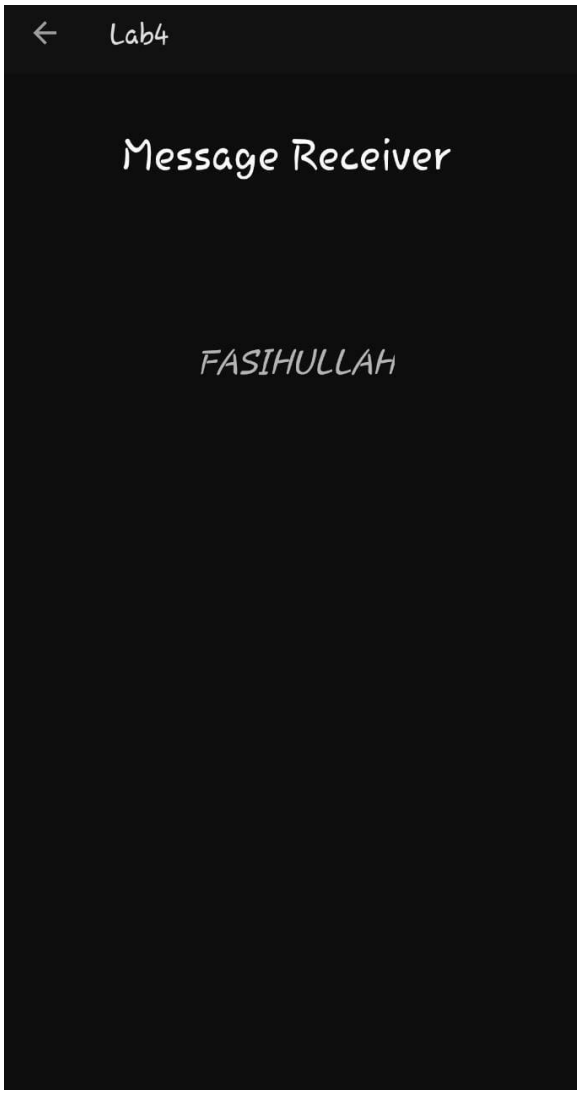
public class MainActivity extends AppCompatActivity {
    public static final String MSG = "lab04.excerciseno1.message.SHOW";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    public void Send(View view){
        Intent intent = new Intent(this,Showmessage.class);
        EditText editText = findViewById(R.id.usermessage);
        String message = editText.getText().toString();
        intent.putExtra(MSG,message);
        startActivity(intent);
        Toast.makeText(this, "Sending...", Toast.LENGTH_SHORT).show();

    }
}
```

OUTPUT:




Lab4
Message Receiver
FASIHULLAH

EXERCISE 2:

```
package com.example.lab04_ex02;

import androidx.appcompat.app.AppCompatActivity;

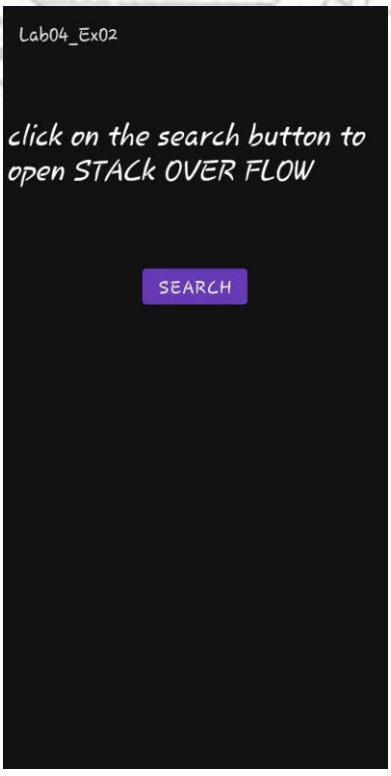
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;

import java.net.URI;
import java.net.URL;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    public void Search(View view){
        Intent intent = new Intent(Intent.ACTION_VIEW,Uri.parse("https://stackoverflow.com/"));
        if ((intent.resolveActivity(getPackageManager())) != null){
            startActivity(intent);
        }
    }
}
```

OUTPUT:




Products
Log in
Sign up
We <3 people who code
We build products that empower developers and connect them to solutions that enable productivity, growth, and discovery.
For developers
For businesses

Lab Session 05

Testing, Debugging and using Support Libraries

Debugging is the process of finding and fixing errors (bugs) or unexpected behavior in your code. All code has bugs, from incorrect behavior in your app, to behavior that excessively consumes memory or network resources, to actual app freezing or crashing.

Bugs can result for many reasons:

- Errors in your design or implementation.
- Android framework limitations (or bugs).
- Missing requirements or assumptions for how the app should work.
- Device limitations (or bugs)

Use the debugging, testing, and profiling capabilities in Android Studio to help you reproduce, find, and resolve all of these problems. Those capabilities include:

- The Android monitor (logcat)
- The Android Studio debugger
- Testing frameworks such as JUnit or Espresso
- Dalvik Debug Monitor Server (DDMS), to track resource usage

In this lab you'll learn how to debug your app with the Android Studio debugger, set and view breakpoints, step through your code, and examine variables.

START DEBUGGING

To start debugging, click Debug in the toolbar. Android Studio builds an APK, signs it with a debug key, installs it on your selected device, then runs it and opens the Debug window.

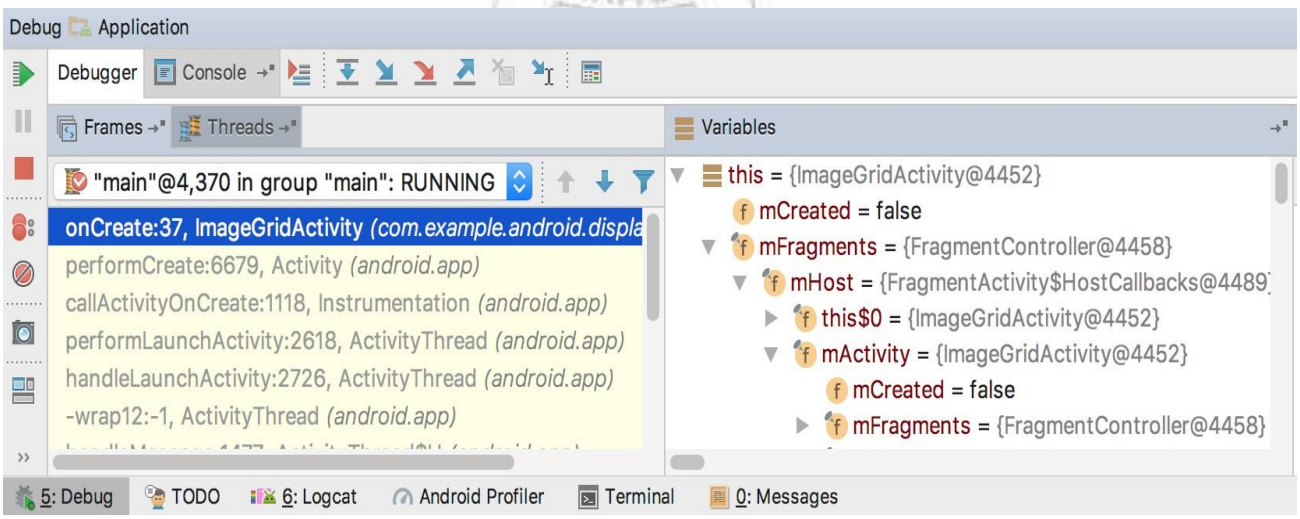


Figure 5.1

RESUME AND STOP DEBUGGING

To resume executing an app after debugging it, select **Run > Resume Program** or click the Resume icon. To stop debugging your app, select **Run > Stop** or click the Stop icon in the toolbar.

USING BREAKPOINTS

A breakpoint is a place in your code where you want to pause the normal execution of your app to perform other actions such as examining variables or evaluating expressions, or executing your code line by line to determine the causes of runtime errors.

Add breakpoints

To add a breakpoint to a line in your code, use these steps:
1. Locate the line of code where you want to pause execution.
2. Click in the left gutter of the editor window at that line, next to the line numbers. A red dot appears at that line, indicating a breakpoint.
You can also use **Run > Toggle Line Breakpoint** or Control-F8 (Command-F8 on OS X) to set a breakpoint at a line.
If your app is already running, you don't need to update it to add the breakpoint.
When your code execution reaches the breakpoint, Android Studio pauses execution of your app. You can then use the tools in the Android debugger to view the state of the app and debug that app as it runs.

After your app's execution has stopped because a breakpoint has been reached, you can execute your code from that point one line at a time with the Step Over, Step Into, and Step Out functions.
To use any of the step functions:
1. Begin debugging your app. Pause the execution of your app with a breakpoint.
Your app's execution stops, and the debugger shows the current state of the app. The current line is highlighted in your code.
2. Click the **Step Over** icon, select **Run > Step Over,** or type F8.
Step Over executes the next line of the code in the current class and method, executing all of the method calls on that line and remaining in the same file.
3. Click the **Step Into** icon, select **Run > Step Into,** or type F7.
Step Into jumps into the execution of a method call on the current line (versus just executing that method and
remaining on the same line). The **Frames** view (which you'll learn about in the next section) updates to show the new
stack frame (the new method). If the method call is contained in another class, the file for that class is opened and the
current line in that file is highlighted. You can continue stepping over lines in this new method call, or step deeper into
other methods.
4. Click the **Step Out** icon, select **Run > Step Out,** or type Shift-F8.
Step Out finishes executing the current method and returns to the point where that method was called.
5. To resume normal execution of the app, select **Run > Resume Program** or click the Resume icon.

ANDROID SUPPORT LIBRARIES

The Android SDK tools include several libraries collectively called the *Android Support Library*. This package of libraries provides several features that are not built into the standard Android framework, and provides backward compatibility for older devices. Include any of these libraries in your app to incorporate that library's functionality.

Features

The features of the Android Support Library include:

- Backward-compatible versions of framework components. These compatibility libraries allow you to use features and components available on newer versions of the Android platform even when your app is running on an older platform version. For example, older devices may not have access to newer features such as fragments, action bars, or Material Design elements. The support library provides access to those features on older devices.
- Additional layout and user interface elements. The support library includes views and layouts that can be useful for your app, but are not included in the standard Android framework. For example, the RecyclerView
- Support for different device form factors, such as TV or wearables: For example, the Leanback library includes components specific to app development on TV devices.
- Design support: The design support library includes components to support Material Design elements in your app, including floating action buttons (FAB).
- Various other features such as palette support, annotations, percentage-based layout dimensions, and preferences.

Setting up and using the Android Support Library

The Android Support Library package is part of the Android SDK, and available to download in the Android SDK manager.

To set up your project to use any of the support libraries, use these steps:

1. Download the support library with the Android SDK manager, or verify that the support libraries are already available.
2. Find the library dependency statement for the support library you're interested in.
3. Add that dependency statement to your build.gradle file.

Download the support library

In Android Studio, you'll use the Android Support Repository—the repository in the SDK manager for all support libraries to get access to the library from within your project.

You may already have the Android support libraries downloaded and installed with Android Studio. To verify that you have the support libraries available, follow these steps:

1. In Android Studio, select **Tools > Android > SDK Manager**, or click the SDK Manager icon. The SDK Manager preference pane appears.
2. Click the **SDK Tools** tab and expand Support Repository.
3. Look for **Android Support Repository** in the list.
    - If **Installed** appears in the Status column, you're all set. Click **Cancel**.
    - If **Not installed** or **Update Available** appears, click the checkbox next to Android Support Repository. A download icon should appear next to the checkbox. Click **OK.**
4. Click **OK** again, and then **Finish** when the support repository has been installed.

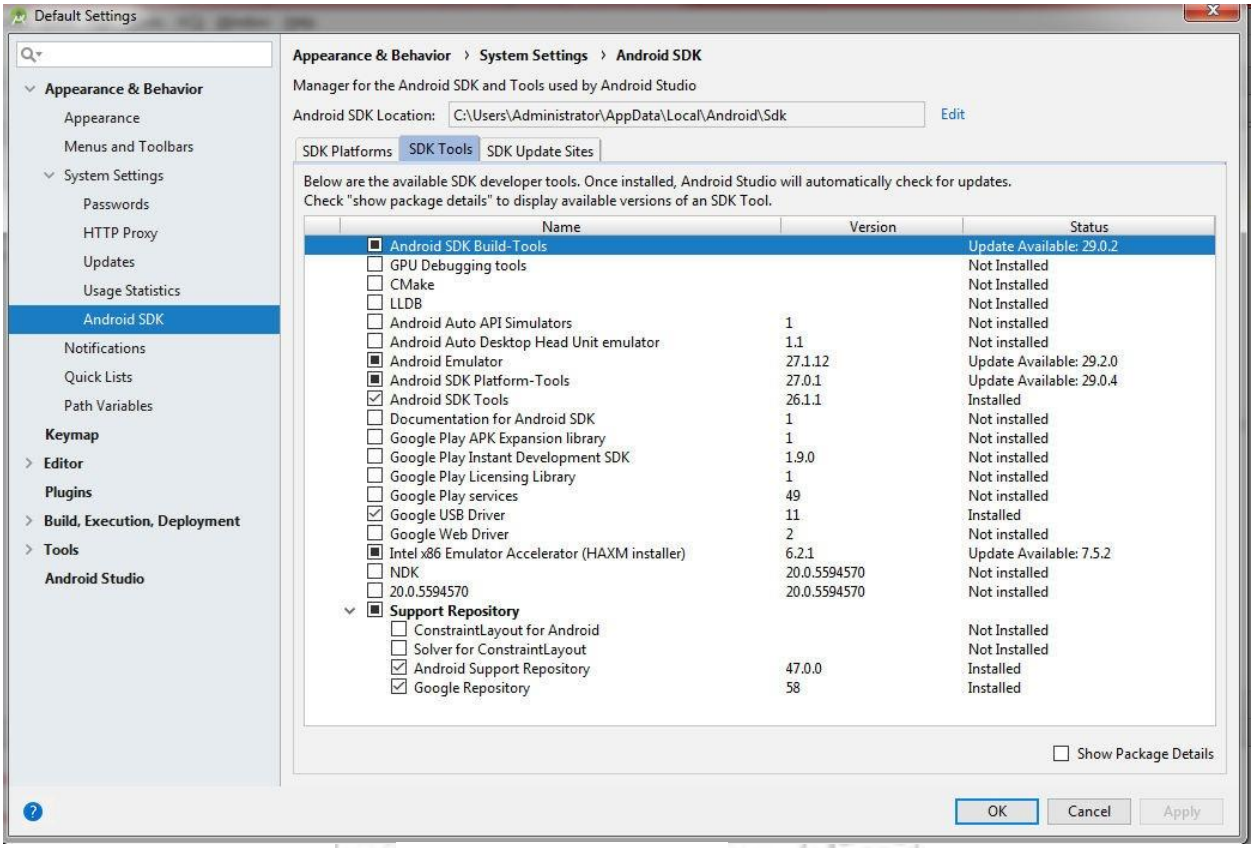


Figure 5.2

Find a library dependency statement

To provide access to a support library from your project, you add that library to your gradle build file as a dependency.

Dependency statements have a specific format that includes the name and version number of the library.

1. Visit the Support Library Features page on developer.android.com.
2. Find the library you're interested in on that page, for example, the Design Support Library for Material Design support.
3. Copy the dependency .

Add the dependency to your build.gradle file

The gradle scripts for your project manage how your app is built, including specifying the dependencies your app has on other libraries. To add a support library to your project, modify your gradle build files to include the dependency to that library.

1. In Android Studio, make sure the **Project** pane is open and the Android tab is clicked.
2. Expand **Gradle Scripts**, if necessary, and open the **build.gradle (Module: app)** file.

Note that build.gradle for the overall project (build.gradle (Project: app_name) is a different file from the build.gradle for the app module.

3. Locate the dependencies section of build.gradle, near the end of the file.

The dependencies section for a new project may already include dependencies several other libraries.

4. Add a dependency for the support library. For example, a dependency on the design support library looks like this:

```
compile 'com.google.zxing:core:3.0.1'
```

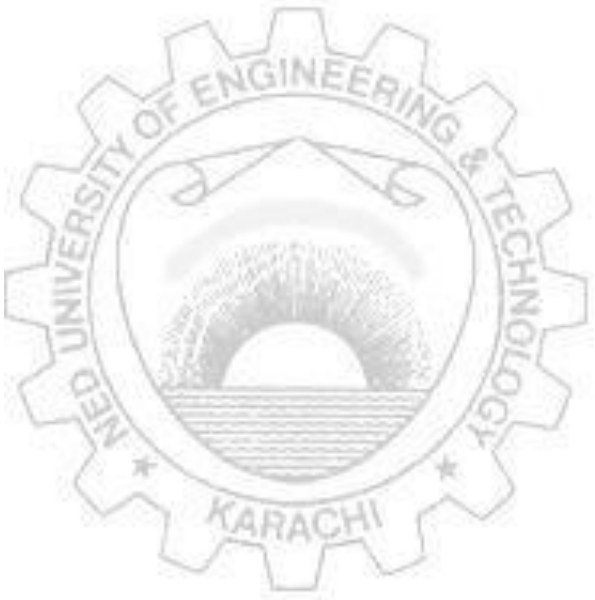
5. Update the version number, if necessary.

If the version number you specified is lower than the currently available library version number, Android Studio will warn you that an updated version is available. ("a newer version of com.android.support:design is available"). Edit the version number to the updated version, or type Shift+Enter and choose "Change to XX.X.X" where XX.X.X is theupdated version number.
6. Click **Sync Now** to sync your updated gradle files with the project, if prompted.

Using Support Libraries API

All the support library classes are contained in the android.support packages, for example, android.support.v7.app.AppCompatActivity is the fully-qualified name for the AppCompatActivity class, from which all of your activities extend.
Support Library classes that provide support for existing framework APIs typically have the same name as framework class but are located in the android.support class packages. Make sure that when you import those classes you use the right package name for the class you're interested in. For example, when applying the ActionBar class, use one of:

- android.support.v7.app.ActionBar when using the Support Library.
- android.app.ActionBar when developing only for API level 11 or higher.



Lab Session 06

Using Adapters and Adapter views

Adapter

In Android, Adapter is a bridge between UI component and data source that helps us to fill data in UI component. It holds the data and send the data to an Adapter view then view can takes the data from the adapter view and shows the data on different views like as ListView, GridView, Spinner etc. For more customization in Views base adapter or custom adapters are used.

Adapters in Android

There are the some commonly used Adapters in Android used to fill the data in the UI components.

1. BaseAdapter – It is parent adapter for all other adapters
2. ArrayAdapter – It is used whenever we have a list of single items which is backed by an array
3. Custom ArrayAdapter – It is used whenever we need to display a custom list
4. SimpleAdapter – It is an easy adapter to map static data to views defined in your XML file
5. Custom SimpleAdapter – It is used whenever we need to display a customized list and needed to access the child items of the list or grid

Creating an ArrayAdapter

To create an adapter, you need the following:

- a data set
- a resource file containing the layout of the generated `View` objects

Additionally, because the `ArrayAdapter` class can only work with strings, you need to make sure the layout of the generated `View` objects contains at least one `TextView` widget.

Step 1: Create the Data Set

The ArrayAdapter class can use both arrays and List objects as data sets. For now, let's use an array as the data set.

```
String[]
Dinosaurs={Ankylosaurus,”Brontosaurus”,”Stegosaurus”,”Triceratops”,”Tyrannosa
urus”};
```

Step 2: Create the Resource File

Create a new layout XML file whose root element is a LinearLayout and name it item.xml. Drag and drop a Large text widget in it and set the value of its id attribute to dinosaur_name. The layout XML file should look like this:

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

```
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="Large Text"
        android:id="@+id/dinosaur_name" />
</LinearLayout>
```

Step 3: Create the Adapter

In your activity, create a new instance of the ArrayAdapter class using its constructor. As its arguments, pass the name of the resource file, the identifier of the TextView, and a reference to the array. The adapter is now ready.

```
ArrayAdapter<String> dinosaurAdapter =
    new ArrayAdapter<String>(this,
        R.layout.item,
        R.id.dinosaur_name,
        Dinosaurs
    );
```

4. Creating a List

To display a vertically scrollable list of items, you can use the ListView widget. To add the widget to your activity, you can either drag and drop it inside the activity's layout XML file or create it using its constructor in your Java code. For now, let's do the latter.

```
ListView dinoList = new ListView(this);
```

Usually, no other user interface widgets are placed inside a layout that contains a ListView. Therefore, pass the ListView to the setContentView() method of your activity so that it takes up the entire screen.

```
setContentView(dinoList);
```

To bind the ListView to the adapter we created in the previous step, call the setAdapter() method as shown below.

```
dinoList.setAdapter(dinosaurAdapter);
```

Run your app now, you should be able to see the contents of the array in the form of a list.

Extending the Array Adapter

An ArrayAdapter can handle only one TextView widget inside the layout of the View objects it generates. To broaden its capabilities you must extend it. Before we do that, however, let's create a slightly more complex data set.

Instead of strings, let's say our data set contains objects of the following class:

```
static class Dinosaur {
    String name;
    String description;
```

```
    public Dinosaur(String name, String description) {
        this.name = name;
        this.description = description;
    }
}
```

This is the data set we will be using:

```
Dinosaur[] Dinos = {
        new Dinosaur("Ankylosaurus ", "Stiff Dino"),
        new Dinosaur ("Brontosaurus ", "Thunder Dino"),
        new Dinosaur ("Stegosaurus ", "Roofed Dino"),
        new Dinosaur ("Triceratops ", "Three Horned dino"),
        new Dinosaur ("Tyrannosaurus ", "a large meat eating dino"),
};
```

As you can see, the Dinosaur class contains two fields, name and description. To display both fields in a lists or grid, the layout of the items must contain two TextView widgets.

Create a new layout XML file and name it **custom_item.xml**. Add a **Large text** and a **Small text** widget to it. Set the id attribute of the first widget to **dinosaur_name** and that of the second one to **dinosaur_description**. The contents of the layout XML file should now look like this:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="@dimen/activity_horizontal_margin">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="Large Text"
        android:id="@+id/dinosaur_name" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceSmall"
        android:text="Small Text"
        android:id="@+id/dinosaur_description" />
</LinearLayout>
```

he ArrayAdapter must also be capable of handling two TextView widgets. Revisit your activity, create a new anonymous class that extends the ArrayAdapter class, and override its getView() method. Make sure that you pass the array as an argument to its constructor.

```
ArrayAdapter<Dinosaur> dinoAdapter =
        new ArrayAdapter<Dinosaur>(this, 0, dinosaurs) {
            @Override
            public View getView(int position,
                                View convertView,
                                ViewGroup parent) {

            }};
```

Inside the getView() method, you must use the position parameter as an index of the array and fetch the item at that index.

```
Dinosaur currentDino = Dinosaurs[position];
```

The second parameter of the getView() method is what enables us to reuse View objects. If you ignore it, the performance of your adapter view will be poor. When the getView() method is called for the first time, convertView is null. You must initialize it by inflating the resource file that specifies the layout of the list items. To do so, obtain a reference to a LayoutInflater using the getLayoutInflater() method and invoke its inflate() method.

```
// Inflate only once
if(convertView == null) {
    convertView = getLayoutInflater()
                    .inflate(R.layout.custom_item, null, false);
}
```

At this point, you can use findViewById() to get a reference to the TextView widgets inside the layout and call their setText() methods to initialize them using data from the array.

```
TextView dinoName =
    (TextView)convertView.findViewById(R.id.dinosaur_name);
TextView dinoDescription =
    (TextView)convertView.findViewById(R.id.dinosaur_description);

dinoName.setText(currentDino.name);
dinoDescription.setText(currentDino.description);
```

Finally, return convertView so that it can be used to populate any adapter view associated with the adapter.

```
return convertView;
```

Using a View Holder

The getView() method is called repeatedly by the adapter view to populate itself. Therefore, you must try to minimize the number of operations you perform in it.

In the previous step, you might have noticed that, even though we made sure that the layout of the list items is inflated only once, the findViewById() method, which consumes many CPU cycles, is called every time the getView() method is called.

To avoid this and to improve the performance of the adapter view, we need to store the results of the findViewById() method inside the convertView object. To do so, we can use a **view holder** object, which is nothing more than an object of a class that can store the widgets present in the layout.

Because the layout has two TextView widgets, the view holder class must also have two TextView widgets. I have named the class **ViewHolder**.

```
static class ViewHolder{
    TextView dinoName;
    TextView dinoDescription;
}
```

In the getView() method, after you inflate the layout, you can now initialize the view holder object using the findViewById() method.

```
ViewHolder viewHolder = new ViewHolder();
viewHolder.dinoName =
        (TextView)convertView.findViewById(R.id.dinosaur_name);
viewHolder.dinoDescription =
        (TextView)convertView.findViewById(R.id.dinosaur_description);
```

To store the view holder object in convertView, use its setTag() method.

```
// Store results of findViewById
convertView.setTag(viewHolder);
```

And now, every time getView() is called, you can retrieve the view holder object from convertView using the getTag() method and update the TextView widgets inside it using their setText() methods.

```
TextView dinoName =((ViewHolder)convertView.getTag()).dinoName;
TextView dinoDescription
=    ((ViewHolder)convertView.getTag()).dinoDescription;

dinoName.setText(currentDino.name);
dinoDescription.setText(currentdino.description);
```

If you run your app now, you can see the GridView displaying two lines of text in each cell.

Creating a Grid

To display a vertically scrollable two-dimensional grid of items, you can use the GridView widget. Both ListView and GridView are subclasses of the abstract AbsListView class and they share many similarities. Therefore, if you know how to use one, you know how to use the other as well.

Use the constructor of the GridView class to create a new instance and pass it to the setContentView() method of your activity.

```
GridView dinoGrid = new GridView(this);
setContentView(dinoGrid);
```

To set the number of columns in the grid, call its setNumColumns() method. I'm going to make this a two-column grid.

```
dinoGrid.setNumColumns(2);
```

Usually, you'd want to adjust the width of the columns and the spacing between them using the setColumnWidth(), setVerticalSpacing(), and setHorizontalSpacing() methods. Note that these methods use pixels as their units.

```
dinoGrid.setColumnWidth(60);
```

```
dinoGrid.setVerticalSpacing(20);
dinoGrid.setHorizontalSpacing(20);
```

You can now bind the GridView to the adapter we created earlier using the setAdapter() method.

```
dinoGrid.setAdapter(dinoAdapter);
```

Run your app again to see what the `GridView` looks like.

Adding Event Listeners

It is possible to listen for click and long click events on the items inside an adapter view. As an example, let's add a click event listener to the GridView.

Create a new instance of an anonymous class that implements the AdapterView.OnItemClickListener interface and pass it to the setOnItemClickListener() method of the GridView object. Android Studio automatically generates a stub for the onItemClick() method of the interface. You'll notice that the method's parameters include an integer specifying the position of the list item. You can use this integer to find out which item in the data set the user clicked.

The following code illustrates how to display a simple message as a snackbar every time an item in the GridView is clicked.

```
dinoGrid.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> adapterView,
                            View view, int position, long rowId) {

        // Generate a message based on the position
        String message = "You clicked on " + dinosaurs[position];

        // Use the message to create a Snackbar
        Snackbar.make(adapterView, message, Snackbar.LENGTH_LONG)
                .show(); // Show the Snackbar
    }
});
```

If you run the app and click any item in the grid, a message appears at the bottom of the screen. Note that you can use the same code to listen for click events on items inside a ListView.

Exercise

1) Build an application that consists of list of rows displaying text descriptions and an info icon. Clicking the row should display the SnackBar with the text details of that row.

EXERCISE:

```
package com.example.lab06;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.View;
```

```
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.TextView;

import com.google.android.material.snackbar.Snackbar;

import java.util.ArrayList;

public class MainActivity extends AppCompatActivity {

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

        ListView myListView = findViewById(R.id.myListView);
        ArrayList<String> sportsitems = new ArrayList<>();
        sportsitems.add("Cricket");
        sportsitems.add("Football");
        sportsitems.add("Basketball");
        sportsitems.add("Table Tennis");
        sportsitems.add("Base ball");
        sportsitems.add("Badminton");
        sportsitems.add("Rugby");
        sportsitems.add("Gulf");

        ArrayAdapter<String> arrayAdapter=new ArrayAdapter<String>(this,
R.layout.single_row,R.id.textView,sportsitems);
        myListView.setAdapter(arrayAdapter);
        myListView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
                String text = sportsitems.get(position) + " is a type of sport";
                Snackbar.make(parent,text,Snackbar.LENGTH_LONG).show();
            }
        });

    }
```

OUTPUT:


Lab 06
SPORTS ITEMS
Cricket
Football
Basketball
Table Tennis
Base ball
Badminton
Rugby
Gulf
Cricket is a type of sport

Lab Session 07

Learning Menus, Alerts and Pickers

Android Menus

Menus are a common user interface component in many types of applications.
Beginning with Android 3.0 (API level 11), Android-powered devices are no longer required to provide a dedicated *Menu* button. With this change, Android apps should migrate away from a dependence on the traditional 6-item menu panel and instead provide an app bar to present common user actions.
Although the design and user experience for some menu items have changed, the semantics to define a set of actions and options is still based on the Menu APIs.
In android, we can define a Menu in separate XML file and use that file in our activities or fragments based on our requirements.

There are three fundamental types of menus in android

- Option menu
- Context menu
- Pop up menu

1. Option menu

In android, **Options Menu** is a primary collection of menu items for an activity and it is useful to implement actions that have a global impact on the app, such as Settings, Search, etc.

i. Create Android Options Menu in XML File

In android, to define options menu, we need to create a new folder menu inside of our project resource directory (res/menu/) and add a new XML (menu_example) file to build the menu.

Following is the example of defining a menu in XML file (**menu_example.xml**).

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/mail"
        android:icon="@drawable/ic_mail"
        android:title="@string/mail" />
    <item android:id="@+id/upload"
        android:icon="@drawable/ic_upload"
        android:title="@string/upload"
        android:showAsAction="ifRoom" />
    <item android:id="@+id/share"
        android:icon="@drawable/ic_share"
        android:title="@string/share" />
</menu>
```

ii. Load Android Options Menu from an Activity

To specify the options menu for an activity, we need to override onCreateOptionsMenu() method and load the defined menu resource using MenuInflater.inflate() like as shown below.

```
 @Override
public void onCreateOptionsMenu(ContextMenu menu, View v,
ContextMenuInfo menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.menu_example, menu);
}
```

iii. Handle Android Options Menu Click Events

In android, we can handle a options menu item click events using **onOptionsItemSelected()** event method. Following is the example of handling a options menu item click event using **onOptionsItemSelected()**.

```
 @Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.mail:
            // do something
            return true;
        case R.id.share:
            // do something
            return true;
        default:
            return super.onContextItemSelected(item);
    }
}
```

2. Context Menu

In android, **Context Menu** is like a floating menu and that appears when the user performs a long press or click on an element and it is useful to implement an actions that effect the selected content or context frame.

i. Create Android Context Menu in Activity

In android, the Context Menu offers an actions that effect a specific item or context frame in the UI and we can provide a context menu for any view. The context menu won't support any item shortcuts and item icons.

The views which we used to show the context menu on long press, we need to register that views using registerForContextMenu(View) in our activity and we need to override onCreateContextMenu() in our activity or fragment.

When the registered view receives a long click event, the system calls our onCreateContextMenu() method. By using onCreateContextMenu() method, we can create our menu items like as shown below.

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

```
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    Button btn = (Button) findViewById(R.id.btnShow);
    registerForContextMenu(btn);
}
@Override
public void onCreateContextMenu(ContextMenu menu, View v,
ContextMenu.ContextMenuInfo menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);
    menu.setHeaderTitle("Context Menu");
    menu.add(0, v.getId(), 0, "Upload");
    menu.add(0, v.getId(), 0, "Search");
}
```

If you observe above code, we registered our Button control using registerForContextMenu() to show the context menu on button long-click and binding the Context Menu items using onCreateContextMenu() method.

ii. Handle Android Context Menu Click Events

In android, we can handle a context menu item click events using onContextItemSelected() method. Following is the example of handling a context menu item click event using onContextItemSelected() method.

```
@Override
public boolean onContextItemSelected(MenuItem item) {
    if (item.getTitle() == "Save") {
        // do your coding
    }
    else {
        return  false;
    }
    return true;
}
```

Note: If you are using Android 3.0 +, the Context Menu won't support any item shortcuts and item icons in the menu.

3. Popup menu

In android, to define **popup menu**, we need to create a new folder **menu** inside of our project resource directory (**res/menu/**) and add a new XML (menu_example) file to build the menu.

Create Android Popup Menu in XML File

Following is the example of defining a menu in XML file (**menu_example.xml**).

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/mail"
        android:icon="@drawable/ic_mail"
        android:title="@string/mail" />
    <item android:id="@+id/upload"
        android:icon="@drawable/ic_upload"
        android:title="@string/upload"
        android:showAsAction="ifRoom" />
  <item android:id="@+id/share"
```

```
        android:icon="@drawable/ic_share"
        android:title="@string/share" />
</menu>
```

Once we are done with the creation of menu, we need to create a view element which anchored the menu.

```
<Button
    android:id="@+id/btnShow"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Show Popup Menu"
    android:onClick="showPopup" />
```

Now in our activity we need to implement showPopup method to show the popup menu.

Load Android Popup Menu from an Activity

To show the popup menu for the view, we need to instantiate **Popup** constructor and use **MenuInflater** to load the defined menu resource using **MenuInflater.inflate()** like as shown below.

```
public void showPopup(View v) {
    PopupMenu popup = new PopupMenu(this, v);
    MenuInflater inflater = popup.getMenuInflater();
    inflater.inflate(R.menu.menu_example, popup.getMenu());
    popup.show();
}
```

Handle Android Popup Menu Click Events

To perform an action when the user selects a menu item, we need to implement the PopupMenu.OnMenuItemClickListener interface and register it with our PopupMenu by calling setOnMenuItemclickListener(). When the user selects an item, the system calls the onMenuItemClick() callback in your interface.

Following is the example of handling a popup menu item click event using **onMenuItemClick()**.

```
public void showMenu(View v) {
    PopupMenu popup = new PopupMenu(this, v);
    popup.setOnMenuItemClickListener(this);
    popup.inflate(R.menu.actions);
    popup.show();
}

@Override
public boolean onMenuItemClick(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.archive:
            archive(item);
            return true;
        case R.id.delete:
            delete(item);
            return true;
        default:
            return false;
```

```
    }
}
```

Note: If you are using Android 3.0 +, the Popup Menu won't support any item shortcuts and item icons in the menu.

AutoCompleteTextView

In android, AutoCompleteTextView is an editable text view which is used to show the list of suggestions based on the user typing text. The list of suggestions will be shown as a dropdown menu from which the user can choose an item to replace the content of textbox.

Generally, the dropdown list of suggestions can be obtained from the data adaptor and those suggestions will be appeared only after giving the number characters defined in the Threshold limit.

The Threshold property of AutoCompleteTextView is used to define the minimum number of characters the user must type to see the list of suggestions.

The dropdown list of suggestions can be closed at any time in case if no item is selected from the list or by pressing the back or enter key.

In android, we can create a AutoCompleteTextView control in two ways either in XML layout file or create it in Activity file programmatically.

Create AutoCompleteTextView in Layout File

Following is the sample way to define AutoCompleteTextView control in XML layout file in android application.

```
<?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" >
    <AutoCompleteTextView
        android:id="@+id/autoComplete_Country"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content" />
</LinearLayout>
```

If you observe above code snippet, here we defined **AutoCompleteTextView** control in xml layout file.

Create AutoCompleteTextView Control in Activity File

In android, we can create AutoCompleteTextView control programmatically in activity file to show the list of suggestions based on user entered text.

Following is the example of creating AutoCompleteTextView control dynamically in activity file.

```
LinearLayout l_layout =  (LinearLayout)
findViewById(R.id.linear_Layout);
AutoCompleteTextView actv = new AutoCompleteTextView(this);
l_layout.addView(actv);
```

Set the Text of Android AutoCompleteTextView

In android, we can set the text of AutoCompleteTextView control by using setAdapter() method in Activity file.

Following is example of binding data AutoCompleteTextView in activity file using **setAdapter()** method.

```
String[] Countries = { "Pakistan", "USA", "Australia", "UK", "Italy",
"Ireland", "Africa" };
ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
android.R.layout.simple_dropdown_item_1line,Countries);
AutoCompleteTextView actv =
AutoCompleteTextView)findViewById(R.id.autoComplete_Country);
actv.setAdapter(adapter);
```

Spinner

In android, **Spinner** is a view which allows a user to select one value from the list of values. The spinner in android will behave same like dropdown list in other programming languages.

Generally, the android spinners will provide a quick way to select one item from the list of values and it will show a dropdown menu with a list of all values when we click or tap on it.

By default, the android spinner will show its currently selected value and by using **Adapter** we can bind the items to spinner object.

Create Android Spinner in XML Layout File

In android, we can create **Spinner** in XML layout file using **<Spinner>** element with different attributes like as shown below.

```
<Spinner android:id="@+id/spinner1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
```

Populate Android Spinner with Values

To populate spinner with list of values, we need to specify spinner adapter, such as an **ArrayAdapter** in activity file like as shown below.

```
String[] users = { "Sindh”,”Punjab”,”Balochistan”,”Khyber
Pakhtunkhwa”};
Spinner spin = (Spinner) findViewById(R.id.spinner1);
ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
android.R.layout.simple_spinner_item, users);
adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdo
wn_item);
spin.setAdapter(adapter);
```

This is how we can define and bind data to Spinner control in android applications. Now we will see complete example of using spinner control android applications.

Android DatePicker with Calendar Mode

We can define android DatePicker to show only calendar view by using DatePicker android:datePickerMode attribute.

Following is the example of showing the DatePicker in **Calendar** mode.

```
<DatePicker
    android:id="@+id/datePicker1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:datePickerMode="calendar"/>
```

Android TimePicker

In android, **TimePicker** is a widget for selecting the time of day, in either 24-hour or AM/PM mode.
If we use **TimePicker** in our application, it will ensure that the users will select a valid time for the day.

```
<TimePicker android:id="@+id/timePicker1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:timePickerMode="clock" />
```

Android Alert Dialog

In android, **Dialog** is a small window that prompts messages to the user to make a decision or enter additional details. AlertDialog is used to prompt a dialog to the user with message and buttons to perform an action to proceed further.

In order to make an alert dialog, you need to make an object of AlertDialogBuilder which an inner class of AlertDialog. Its syntax is given below

```
AlertDialog.Builder alertDialogBuilder = new
AlertDialog.Builder(this);
```

Now you have to set the positive (yes) or negative (no) button using the object of the AlertDialogBuilder class. Its syntax is

```
alertDialogBuilder.setPositiveButton(CharSequence text,
   DialogInterface.OnClickListener listener)
alertDialogBuilder.setNegativeButton(CharSequence text,
   DialogInterface.OnClickListener listener)
```

After creating and setting the dialog builder , you will create an alert dialog by calling the create() method of the builder class. Its syntax is

```
AlertDialog alertDialog = alertDialogBuilder.create();
alertDialog.show();
```

This will create the alert dialog and will show it on the screen.

Exercise

1) Open the list view app that you created in the last lab ,add a floating context menu to show three menu options: **Edit**, **Share**, and **Delete**. The menu should appear when the user performs a long click on the TextView.

EXERCISE:

```
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.TextView;

import com.google.android.material.snackbar.Snackbar;

import java.util.ArrayList;

public class MainActivity extends AppCompatActivity {


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

        ListView myListView = findViewById(R.id.myListView);
        ArrayList<String> sportsitems = new ArrayList<>();
        sportsitems.add("Cricket");
        sportsitems.add("Football");
        sportsitems.add("Basketball");
        sportsitems.add("Table Tennis");
        sportsitems.add("Base ball");
        sportsitems.add("Badminton");
        sportsitems.add("Rugby");
        sportsitems.add("Gulf");
        ArrayAdapter<String> arrayAdapter=new ArrayAdapter<String>(this,
R.layout.single_row,R.id.textView,sportsitems);
        registerForContextMenu(myListView);
        myListView.setAdapter(arrayAdapter);
        myListView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
                String text =sportsitems.get(position) +  " is a type of sport";
                Snackbar.make(parent,text,Snackbar.LENGTH_LONG).show();
            }

        });



    }
    @Override
```

```
    public void onCreateContextMenu(ContextMenu menu, View v, ContextMenu.ContextMenuInfo menuInfo) {
        super.onCreateContextMenu(menu, v, menuInfo);
        getMenuInflater().inflate(R.menu.context_menu,menu);


    }

}
```

OUTPUT:

Lab Session 08

Supporting Localization, multiple Screen sizes and Screen Orientation

Localization

Android runs on many devices in many regions. To reach the most users, your app should handle text, audio files, numbers, currency, and graphics in ways appropriate to the locales where your app is used.

You can localize your application according to different regions etc. We will localize the strings used in the application.

Localizing Strings

In order to localize the strings used in your application , make a new folder under **res** with name of **values-local** where local would be the replaced with the region.

For example, in the case of italy, the **values-it** folder would be made under res.

Once that folder is made, copy the **strings.xml** from default folder to the folder you have created. And change its contents. For example the hello world string will be changed as follows.

Italy, res/values-it/strings.xml

```
<;?xml version="1.0" encoding="utf-8"?>
<resources>
   <string name="hello_world">Ciao mondo!</string>
</resources>
```

Spanish, res/values-it/strings.xml

```
<;?xml version="1.0" encoding="utf-8"?>
<resources>
   <string name="hello_world">Hola Mundo!</string>
</resources>
```

French, res/values-it/strings.xml

```
<;?xml version="1.0" encoding="utf-8"?>
<resources>
   <string name="hello_world">Bonjour le monde !</string>
</resources>
```

Multiple Screen Sizes Support in Android

Android runs on a variety of devices that offer different screen sizes and densities. That's why handling the multiple screen size is most important. We should design the user interface of our application in such a way so that it appears correctly on the widest possible range of devices.

To simplify the way that you design your user interfaces for multiple screens, Android divides the range of actual screen sizes and densities into:

- A set of four generalized **sizes**: *small*, *normal*, *large*, and *xlarge*
- A set of four generalized **densities**: *ldpi* (low), *mdpi* (medium), *hdpi* (high), and *xhdpi* (extra high)

✓ Dimensions

For managing different dimensions

- Always avoid hard-coded layout sizes.
- Use only dp (density-independent pixels), match_parent or wrap_content for layout elements dimensions.
- Use only sp (scale-independent pixels) for text size.
- Keep dimensions in dimens.xml files (Not hardcoded in app code or xml file).
- Provide dimensions for different values folder for different screen resolutions.

Values folder for different screen resolutions.

```
values-sw720dp          10.1" tablet 1280x800 mdpi
values-sw600dp          7.0" tablet 1024x600 mdpi
values-sw480dp          5.4" 480x854 mdpi ,
                        5.1" 480x800 mdpi

values-xxhdpi           5.5"  1080x1920 xxhdpi
values-xxxhdpi          5.5"  1440x2560 xxxhdpi
values-xhdpi            4.7"   1280x720 xhdpi,
                        4.65"  1280x720 xhdpi
values-hdpi             4.0" 480x800 hdpi,
                        3.7" 480x854 hdpi
values-mdpi             3.2" 320x480 mdpi
values-ldpi             3.4" 240x432 ldpi,
                        3.3" 240x400 ldpi,
```

✓ Image and icon

- Provide different images and icon in different drawable folder for different screen resolutions**.**

```
drawable-ldpi        //240x320
drawable-mdpi        //320x480
drawable-hdpi        //480x800
drawable-xhdpi       //720x1280
```

```
drawable-xxhdpi       //1080X1920
drawable-xxxhdpi      //1440X2560
```

- Don't apply fixed values everywhere for image. Fixed dimensions can be vary with different device and image will be looked stretch. So use wrapcontent.
- Recommended minimum size for icons is 32 dp and you need 8 dp free space between another icon.
- You should put all your app icons in mipmap directories instead of drawable directories. Beause all mipmap directories are retained in the APK even if you build density-specific APKs. This allows launcher apps to pick the best resolution icon to display on the home screen.

✓ Layout Design

The configuration qualifiers you can use to provide size-specific resources are small, normal, large, and xlarge. For example, layouts for an extra large screen should go in layout-xlarge/.

Screen Orientation

The **screenOrientation** is the attribute of activity element. The orientation of android activity can be portrait, landscape, sensor, unspecified etc. You need to define it in the AndroidManifest.xml file.

```
<activity android:name="package_name.Your_ActivityName"

      android:screenOrientation="orientation_type">

</activity>
```

Adapting to Screen Orientation

As with almost all smartphones, Android supports two screen orientations: portrait and landscape. When the screen orientation of an Android device is changed, the current activity being displayed is destroyed and re-created automatically to redraw its content in the new orientation. In other words, the onCreate() method of the activity is fired whenever there is a change in screen orientation.

`Portrait` mode is longer in height and smaller in width, whereas `landscape` mode is wider but smaller in height. Being wider, `landscape` mode has more empty space on the right side of the screen. At the same time, some of the controls don't appear because of the smaller height. Thus, controls needs to be laid out differently in the two screen orientations because of the difference in the height and width of the two orientations.

There are two ways to handle changes in screen orientation:

- **Anchoring controls**—Set the controls to appear at the places relative to the four edges of the screen. When the screen orientation changes, the controls do not disappear but are rearranged relative to the four edges.

- **Defining layout for each mode**—A new layout file is defined for each of the two screen orientations. One has the controls arranged to suit the `Portrait` mode, and the other has the controls arranged to suit the `Landscape` mode.

Anchoring controls

Following attributes are used for anchoring the views on the four edges of the screen :

- layout_alignParentLeft — Aligns the view to the left of the parent view
- layout_alignParentRight — Aligns the view to the right of the parent view
- layout_alignParentTop — Aligns the view to the top of the parent view
- layout_alignParentBottom — Aligns the view to the bottom of the parent view
- layout_centerVertical — Centers the view vertically within its parent view
- layout_centerHorizontal — Centers the view horizontally within its parent view

Defining layout for each mode(Resize and Repositioning)

Apart from anchoring your views to the four edges of the screen, an easier way to customize the UI based on screen orientation is to create a separate res/layout folder containing the XML files for the UI of each orientation. To support landscape mode, you can create a new folder in the res folder and name it as layout-land (representing landscape).

Basically, the main.xml file contained within the layout folder defines the UI for the activity in portrait mode, whereas the main.xml file in the layout-land folder defines the UI in landscape mode.

```
if(getResources().getDisplayMetrics().widthPixels>getResources().getDisplayMe
trics(). heightPixels)
        {
            Toast.makeText(this,"Screen switched to Landscape
mode",Toast.LENGTH_SHORT).show();
        }
        else
        {
            Toast.makeText(this,"Screen switched to Portrait
mode",Toast.LENGTH_SHORT).show();
        }
```

Exercise

1. Create a single-language application which displays a country's image and a text view according to the language set in the device

EXERCISE:

```
package com.example.lab8;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

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

OUTPUT:

Lab Session 09

Exploring Android services

In android, **Service** is a component which keeps an app running in the background to perform long running operations based on our requirements. For **Service**, we don't have any user interface and it will run the apps in background like playing the music in background or handle network operations when the user in different app.

Android Service Life Cycle

In android, the life cycle of service will follow two different paths Started or Bound.

Started Service

A service is **Started** when an application component, such as an activity calls startService() method. Once it started, it will run indefinitely in background even if the component that started is destroyed.

We can stop the **Started** service by using stopService() method or the service can stop itself by calling stopSelf() method. In android, the **Started** service component will performs a single operation and it won't return any result to the caller.

Bound Service

A service is **Bound** when another application component calls bindService() method. The bound service runs as long as another application component is bound to it.

We can unbind the service by calling unbindService() method based on our requirements. In android, we can bind multiple components to single service at once, but the service will be destroyed in case all the components unbind.

Create a Service

Generally, in android to create a service we must create a subclass of **Service** or use one of existing subclass. In android the application component such as an activity can start the service by calling `startService()` which results in calling the service's `onStartCommand()` method.

Following is the simple example of creating a service in android application.

```
public class SampleService extends Service {
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        //TODO write your own code       return
Service.START_NOT_STICKY;
    }
```

```
    @Override
    public IBinder onBind(Intent intent) {
        //TODO for communication return IBinder implementation
        return null;
    }
}
```

Register a Service in Manifest File

Once we create a service, we must need to register that in android manifest file using **<service>** element like as shown below.

```
<manifest ... >   ...
  <application ... >
     <service android:name=".SampleService" />
  </application>  ...
</manifest>
```

Start a Service

In android, the component such as an activity, service or receiver can start the service using startService() method. Following is the sample code snippet of starting a service using **startService** method.

```
Intent intent = new Intent(this, MyService.class);
startService(intent);
```

Android Service Callback Methods

During the service implementation, following are the callback methods that we need to override in android application.

onStartCommand()

The system will invoke this method when an another component such as an activity requests the service to be started by calling **startService()**.In android, onStartCommand() method must return an **integer** and the integer is a value that describe how the system will continue the service in the event that the system kills it. The onStartCommand() method will return a value from one of the following constants.

Options	Description
START_STICKY	It will restart the service in case if it terminated and the Intent data which is passed to **onStartCommand()** method is NULL. This is suitable for the service which are not executing commands but running independently and waiting for the job.
START_NOT_STICKY	It will not restart the service and it is useful for the services which will run periodically. The service will restart only when there are a pending **startService()** calls. It's a best option to avoid running a service in case if it is not necessary.

START_REDELIVER_INTENT	It's same as **STAR_STICY** and it recreates the service, call `onStartCommand()` with last intent that was delivered to the service.

onBind()

The system will invoke this method when an another component wants to bind with the service by calling bindService(). During implementation of this method, we must need to provide an interface to the clients to communicate with the service by returning an **IBinder** object. In android, we must need to implement this method, in case if we don't need to allow binding, then we should return NULL.

onCreate()

The system will invoke this method when the service is created initially using onStartCommand() or onBind() methods to do one time setup procedures. In case, if the service is already running, then this method will not call.

onDestroy()

The system will invoke this method when the service is no longer used and is being destroyed. This is the final call that the service will receive and we need to implement this method in our service to clean up any unused resources such as threads, receivers or listeners.

Generally, in android if we start a service by calling startService() method, the service will run continuously even if the component that started a service is destroyed until we stop it by using stopService() or it stops itself with stopSelf().

Same way, if we create a service by calling bindService() method, the service will runs as long as the component is bound to it. After the service is unbound from all of its clients, the system will destroys it.

Following is the example of start playing a music in background when we start a service and that music will play continuously until we stop the service in android application.

Create a new android application using android studio and give names as **Services**.

MyService.java

```
public class MyService extends Service {
    private MediaPlayer player;
    @Override
    public IBinder onBind(Intent intent) {
        return  null;
    }
    @Override
    public void onCreate() {
        Toast.makeText(this, "Service was Created",
Toast.LENGTH_LONG).show();
    }
```

```
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        player = MediaPlayer.create(this,
Settings.System.DEFAULT_RINGTONE_URI);
        // This will play the ringtone continuously until we stop the
service.
        player.setLooping(true);
        // It will start the player
        player.start();
        Toast.makeText(this, "Service Started",
Toast.LENGTH_LONG).show();
        return START_STICKY;
    }
    @Override
    public void onDestroy() {
        super.onDestroy();
        // Stopping the player when service is destroyed
        player.stop();
        Toast.makeText(this, "Service Stopped",
Toast.LENGTH_LONG).show();
    }
}
```

Now open **activity_main.xml** file from \src\main\res\layout path and write the following code.

```
    <?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
android:layout_width="match_parent"
    android:layout_height="match_parent">
    <Button
        android:id="@+id/btnStart"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="startService"
        android:layout_marginLeft="130dp"
        android:layout_marginTop="150dp"
        android:text="Start Service"/>
    <Button
        android:id="@+id/btnstop"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:onClick="stopService"
        android:layout_marginLeft="130dp"
        android:layout_marginTop="20dp"
        android:text="Stop Service"/>
</LinearLayout>
```

Now open **MainActivity.java** file write following to implement custom broadcast intents.

MainActivity.java

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
    // Start the service
    public void startService(View view) {
        startService(new Intent(this, MyService.class));
    }
    // Stop the service
    public void stopService(View view) {
        stopService(new Intent(this, MyService.class));
    }
}
```

Now we need to register our service in android manifest file (**AndroidManifest.xml**) using **<service>** attribute like as shown below.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.tutlane.services">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <service android:name=".MyService" />
    </application>
</manifest>
```

Exercise

Create a background service that is going to do back ground operation without interaction with user interface and works even after activity is destroyed.

EXERCISE:

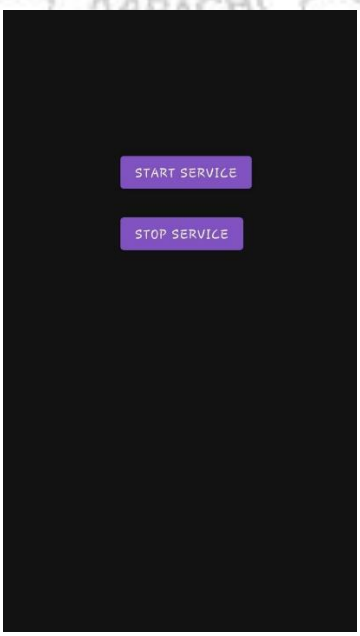
```
package com.example.lab9;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;

import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity  {
    @Override protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    } // Start the service
    public void startService(View view) {
        Intent intent=new Intent(this,MyService.class);
        startService(intent);}
    //startService(new Intent(this, MyService.class)); } // Stop the service
    public void stopService(View view) {
        stopService(new Intent(this, MyService.class)); }
}
```

OUTPUT:



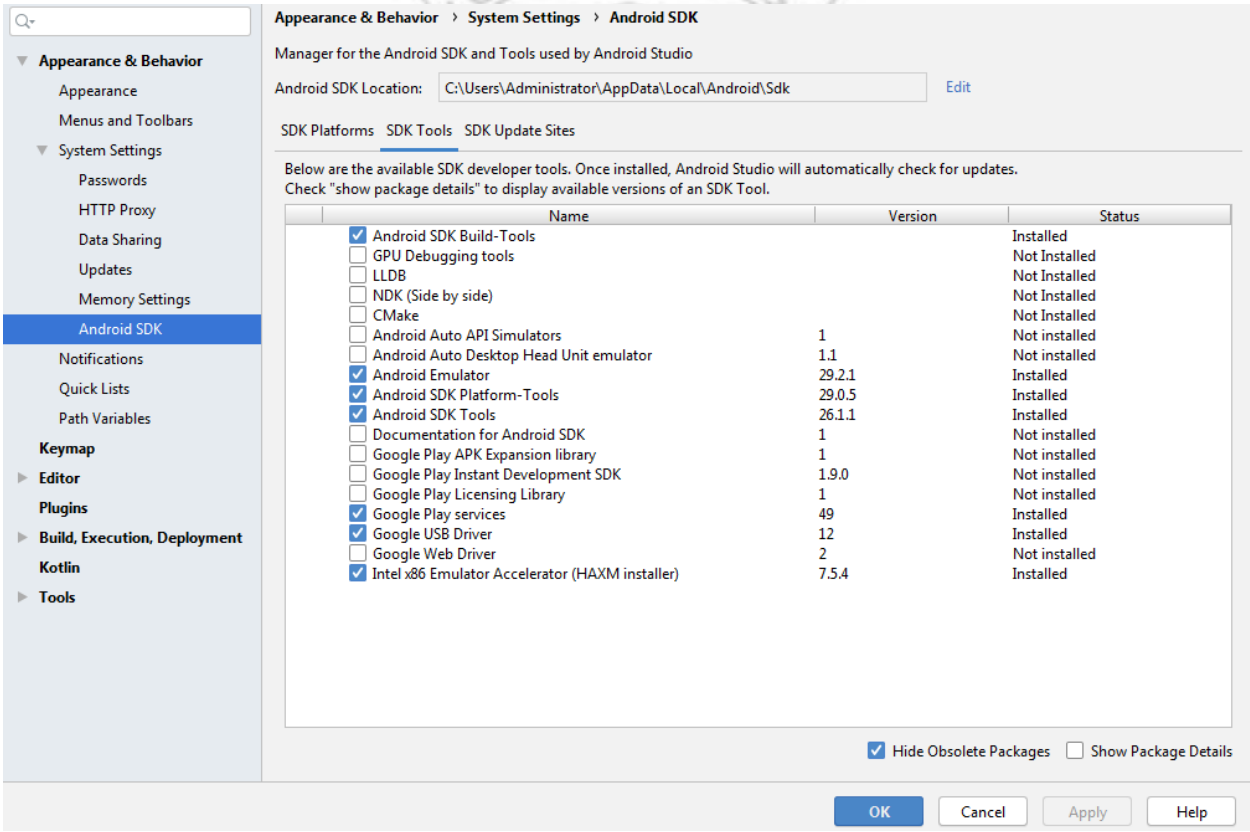
Lab Session 10

Integrating Google maps in Android application

Android allows us to integrate Google Maps in our application. By using **Google Maps** Android API we can integrate google maps in android applications to show the location details on map based on our requirements.

To use google maps in our android applications we need to install **Google Play Services** SDK in our Android Studio because google made **Google Map API** as a part of Google Play Services SDK.

To install Google Play Services, open **Android Studio** à Go to **Tools** menu à **Android** à click **SDK Manager**, then new window will open in that select **SDK Tools** tab à Select **Google Play Services** à click **OK** like as shown below.



Follow these steps to create a new app project including a map activity:

1. Start Android Studio.

2. Create a new project as follows:
3. In the **Choose your project** dialog, select the tab that corresponds to the platform you intended to develop for. Most users will want to keep the default **Phone and Tablet**.
4. Select **Google Maps Activity**, and then click **Next**.
5. Enter your app name, package name, and project location, programming language (Java or Kotlin), and the minimum Android API level supported by your app, then click **Finish**.

When the build is finished, Android Studio opens the google_maps_api.xml and the MapsActivity.java files in the editor. (Note that your activity may have a different name, but it will be the one you configured during setup.)

Your application needs an API key to access the Google Maps servers. The type of key you need is an API key with restriction for **Android apps**. The key is free. You can use it with any of your applications that call the Maps SDK for Android, and it supports an unlimited number of users.

The **google_maps_api.xml** file will contains an instructions to generate a Google Maps API key to access Google Maps servers. Copy the link provided in the **google_maps_api.xml** file.

Copy and paste the console URL in browser and it will take you to **Google API Console** like as shown below. Create an Android-restricted API key for your project. Copy the resulting API key, go back to Android Studio, and paste the API key into the <string> element in the google_maps_api.xml file.

Android Google Map Types

The Google Maps Android API provides a map in different types such as Normal, Hybrid, Satellite, Terrain and None.

Map Type	Description
Normal	Typical road map. Shows roads, some features built by humans, and important natural features like rivers. Road and feature labels are also visible.
Hybrid	Satellite photograph data with road maps added. Road and feature labels are also visible.
Satellite	Satellite photograph data. Road and feature labels are not visible.
Terrian	Topographic data. The map includes colors, contour lines and labels, and perspective shading. Some roads and labels are also visible.
None	No tiles. The map will be rendered as an empty grid with no tiles loaded.

In android, we can change the type of a map by calling the GoogleMap object's **setMapType()** method, by passing the type of constants defined in GoogleMap.

Following is the example of displaying the map type as **Satellite** in android application.

```
GoogleMap map;
......
// Set Map type as Satellite
map.setMapType(GoogleMap.MAP_TYPE_SATELLITE);

Same way we can set other type maps also like as shown below.
```

```
map.setMapType(GoogleMap.MAP_TYPE_NORMAL);
map.setMapType(GoogleMap.MAP_TYPE_HYBRID);
map.setMapType(GoogleMap.MAP_TYPE_TERRAIN);
map.setMapType(GoogleMap.MAP_TYPE_NONE);
```

Google Map Zoom, Rotation

The Maps API provides built-in zoom controls that appear in the bottom right hand corner of the map. These are disabled by default, but can be enabled by calling the following lines of codes:

```
mMap.getUiSettings().setZoomGesturesEnabled(true);
```

A user can rotate the map by placing two fingers on the map and applying a rotate motion. You can disable rotation by calling `UiSettings.setRotateGesturesEnabled(boolean)`.

Adding Information Window

The simplest way to add an info window is to set the title() and snippet() methods of the corresponding marker. Setting these properties will cause an info window to appear whenever that marker is clicked.

```
mMap.addMarker(new MarkerOptions()
        .position(new LatLng(41.005745,  28.977114))
        .title("BlueMosque")
        .snippet("It is located in Turkey")
        .rotation((float) 33.5)
.icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_VIOLET)));
```

snippet() is used to display more data over the marker when it's tapped.

Exercises

1) Explore StreetView in Google Maps. Attach the code and output to display street view of any city.

EXERCISE:

```
package com.example.lab10;


import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Toast;

import com.google.android.gms.maps.OnStreetViewPanoramaReadyCallback;
import com.google.android.gms.maps.StreetViewPanorama;
import com.google.android.gms.maps.SupportStreetViewPanoramaFragment;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.StreetViewPanoramaCamera;
import com.google.android.gms.maps.model.StreetViewPanoramaLocation;
import com.google.android.gms.maps.model.StreetViewPanoramaOrientation;
import com.google.android.gms.maps.model.StreetViewSource;
import com.google.android.material.floatingactionbutton.FloatingActionButton;

public class MainActivity extends AppCompatActivity
        implements OnStreetViewPanoramaReadyCallback {

    private StreetViewPanorama mStreetViewPanorama;
    private boolean secondLocation = false;

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

        SupportStreetViewPanoramaFragment streetViewFragment =
                (SupportStreetViewPanoramaFragment) getSupportFragmentManager()
                        .findFragmentById(R.id.googleMapStreetView);
        streetViewFragment.getStreetViewPanoramaAsync(this);

        FloatingActionButton fab = findViewById(R.id.fab);
        fab.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                secondLocation = !secondLocation;
                onStreetViewPanoramaReady(mStreetViewPanorama);
            }
        });

    }

    @Override
    public void onStreetViewPanoramaReady(StreetViewPanorama streetViewPanorama) {
        mStreetViewPanorama = streetViewPanorama;

        LatLng Melbourne = new LatLng(37.754130, -122.447129);
        streetViewPanorama.setPosition(Melbourne);
    }
}
```

OUTPUT:

Lab Session 11

Using Shared preferences for data storage in Android

One of the most Interesting Data Storage option **Android** provides its users is **Shared Preferences**. **Shared Preferences** is the way in which one can store and retrieve small amounts of primitive data as key/value pairs to a file on the device storage such as String, int, float, Boolean that make up your preferences in an XML file inside the app on the device storage. Shared Preferences allows activities and applications to keep preferences that will persist even when the user closes the application.

The Shared Preferences file is managed by an android framework and it can be accessed anywhere within the app to read or write data into the file, but it's not possible to access the file from any other app so it's secured.

The Shared Preferences are useful to store the small collection of key-values such as user's login information, app preferences related to users, etc. to maintain the state of app, next time when they login again to the app.

Handle a Shared Preference

In android, we can save the preferences data either in single or multiple files based on our requirements.

In case if we use single file to save the preferences, then we need to use **getPreferences()** method to get the values from Shared Preferences file and for multiple files we need to call a **getSharedPreferences()** method and pass a file name as a parameter.

If we are using single shared preference file for our activity, then we need to initialize the **SharedPreferences** object by using **getPreferences()** method like as shown below.

```
SharedPreferences sharedPref = getPreferences(Context.MODE_PRIVATE);
```

In case, if we are using multiple shared preference files, then we need to initialize the **SharedPreferences** object by using **getSharedPreferences()** method like as shown below.

```
SharedPreferences sharedPref = getSharedPreferences("filename1",Context.MODE_PRIVATE);
```

Here, the name "**filename1**" is the preference file, which wants to read the values based on our requirements and the context mode **MODE_PRIVATE** will make sure that the file can be accessed only within our application.

Following are the operating modes applicable:

- **MODE_PRIVATE**: the default mode, where the created file can only be accessed by the calling application

- **MODE_WORLD_READABLE**: Creating world-readable files is very dangerous, and likely to cause security holes in applications
- **MODE_WORLD_WRITEABLE**: Creating world-writable files is very dangerous, and likely to cause security holes in applications
- **MODE_MULTI_PROCESS**: This method will check for modification of preferences even if the Shared Preference instance has already been loaded
- **MODE_APPEND**: This will append the new preferences with the already existing preferences
- **MODE_ENABLE_WRITE_AHEAD_LOGGING**: Database open flag. When it is set, it would enable write ahead logging by default

Write to Shared Preferences

To store a data in shared preference file, we need an **editor** to edit and save the changes in **SharedPreferences** object. Following is the code snippet to store the data in shared preference file using **editor**.

```
SharedPreferences sharedPref = getPreferences(Context.MODE_PRIVATE);
SharedPreferences.Editor editor = sharedPref.edit();
editor.putBoolean("keyname",true);
editor.putString("keyname","string value");
editor.putInt("keyname","int value");
editor.putFloat("keyname","float value");
editor.putLong("keyname","long value");
editor.commit();
```

Read from Shared Preferences

To read or retrieve a values from Shared Preferences file, we need to call a methods such as **getInt()**, **getString()**, etc. by providing the key for the value which we want to get like as shown below.

```
SharedPreferences pref = getPreferences(Context.MODE_PRIVATE);
pref.getString("keyname",null);
pref.getInt("keyname",0);
pref.getFloat("keyname",0);
pref.getBoolean("keyname",true);
pref.getLong("keyname",0);
```

Deleting from Shared Preferences

To delete a values from Shared Preferences file, we need to call a **remove()** method by providing the key for the value which we want to delete like as shown below.

```
SharedPreferences pref = getPreferences(Context.MODE_PRIVATE);
SharedPreferences.Editor editor = pref.edit();
editor.remove("keyname");
editor.commit();
```

Clearing from Shared Preferences

We can clear all the data from Shared Preferences file using **clear()** method like as shown below.

```
SharedPreferences pref = getPreferences(Context.MODE_PRIVATE);
SharedPreferences.Editor editor = pref.edit();
editor.clear();
editor.commit();
```

Exercise

Create an android application to save, retrieve and clear the data input from Edit Texts, using Shared Preferences.

EXAMPLE:

```
package com.example.lab11;

import androidx.appcompat.app.AppCompatActivity;

import android.content.SharedPreferences;
import android.os.Bundle;
import android.provider.ContactsContract;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
    EditText name,email;
    SharedPreferences sharedPreferences;
    public static final String myPreference="myfile";


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        name = (EditText)findViewById(R.id.etName);
        email = (EditText)findViewById(R.id.etEmail);
        sharedPreferences=getSharedPreferences(myPreference,0);

    }

    public void save(View view) {
        String n =name.getText().toString();
        String e=email.getText().toString();
        SharedPreferences.Editor editor=sharedPreferences.edit();
        editor.putString("NameKey",n);
        editor.putString("EmailKey",e);
        editor.commit();
        Toast.makeText(this, "Data has been saved", Toast.LENGTH_SHORT).show();

    }

    public void clear(View view) {
```

```
        name.setText("");
        email.setText("");
        Toast.makeText(this, "Data has been cleared", Toast.LENGTH_SHORT).show();

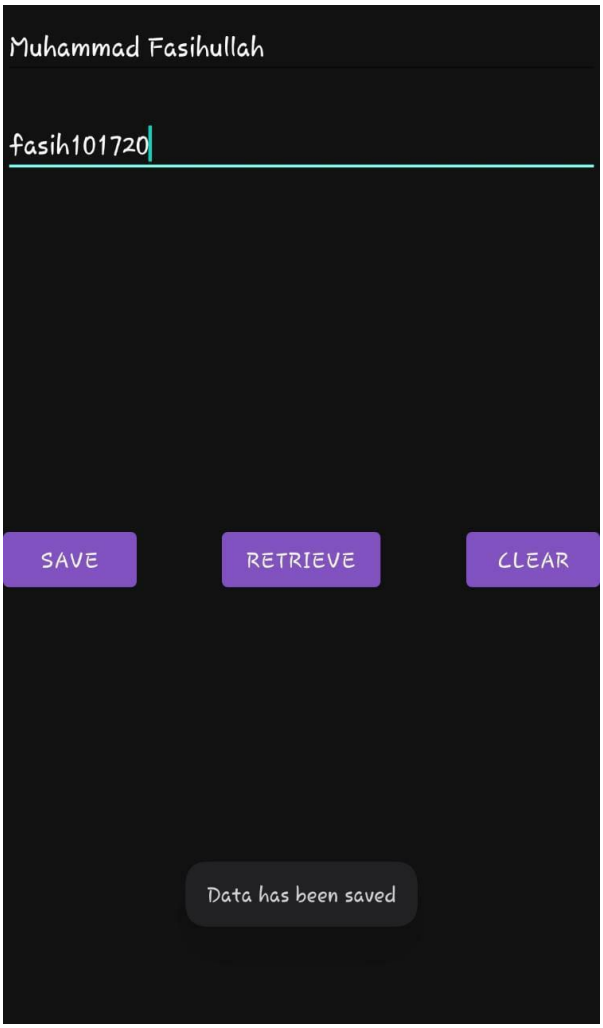
    }

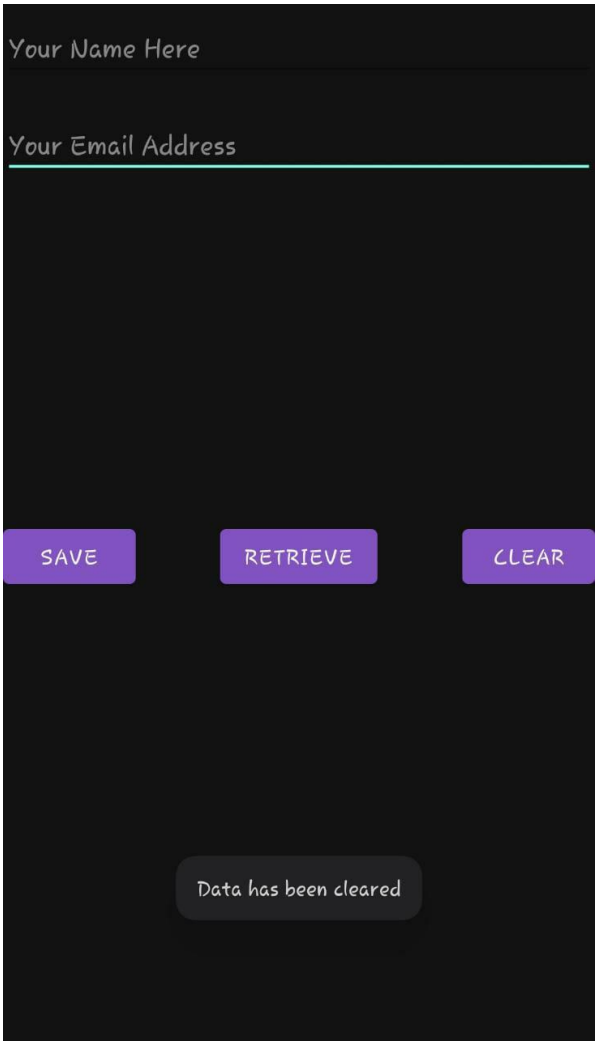
    public void get(View view) {

        name=(EditText)findViewById(R.id.etName);
        email=(EditText)findViewById(R.id.etEmail);
        sharedPreferences=getSharedPreferences(myPreference,0);
        if(sharedPreferences.contains("NameKey")){
            name.setText(sharedPreferences.getString("NameKey","abc"));
        }
        if(sharedPreferences.contains("EmailKey")){
            email.setText(sharedPreferences.getString("EmailKey","abc"));
        }

    }
}
```

OUTPUT:




Your Name Here
Your Email Address
SAVE
RETRIEVE
CLEAR
Data has been cleared