NITTE EDUCATION TRUST | NITTE MEENAKSHI INSTITUTE OF TECHNOLOGY

AN AUTONOMOUS INSTITUTION AFFILIATED TO VTU, BELGAUM, ACCREDIATED by NAAC ('A+' Grade)

YELAHANKA, BANGALORE-560064

A Lab Manual on

HYBRID APP DEVELOPMENT LAB

(18ISL77)

Section	Description
Part-A	Android App Development
Part-B	Flutter App Development

Compiled by,

Mr. Prashanth BS

Assistant Professor

Department of Information Science and Engineering

Nitte Meenakshi Institute of Technology, Bengaluru - 560064

DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING

(Accredited by NBA Tier-1)

2019-20

Contents

PART-A: ANDROID APP DEVELOPMENT

Chapter 1

Inheritance in JAVA

Inheritance in Java is an OOP(Object Oriented Programming) paradigm by which one class is allowed to inherit the features of another class. Some of the terminologies involved in are,

- **Super Class**: The class from which features are inherited from. Also called as "base" or "parent" class
- **Derived Class**: The class that inherits the traits of the other class is called as Derived Class or "Sub class" or "Child Class"
- The Derived class will inherit the traits of Base class by using **extends** keyword

```
// General Structure
class Derived-class extends Base-class{
   // methods and fields
}
```

1.1 Types of Inheritance

Below are the different types of inheritance which are supported by Java[1] as shown in Figure1.1.

1. **Single**: In Single inheritance, sub-classes inherit the features of one super-class.
2. **Multi-level**: In Multilevel Inheritance, a derived class will be inheriting a base class and as well as the derived class also act as the base class to other class
3. **Hierarchical**: In Hierarchical Inheritance, one class serves as a superclass (base class) for more than one subclass

1.2 Uses

The uses of inheritance is as follows,

1. To achieve **run-time polymorphism**
2. **Code Re-usability**
3. Cleaner Code and Higher readability

[1] https://www.geeksforgeeks.org/inheritance-in-java/

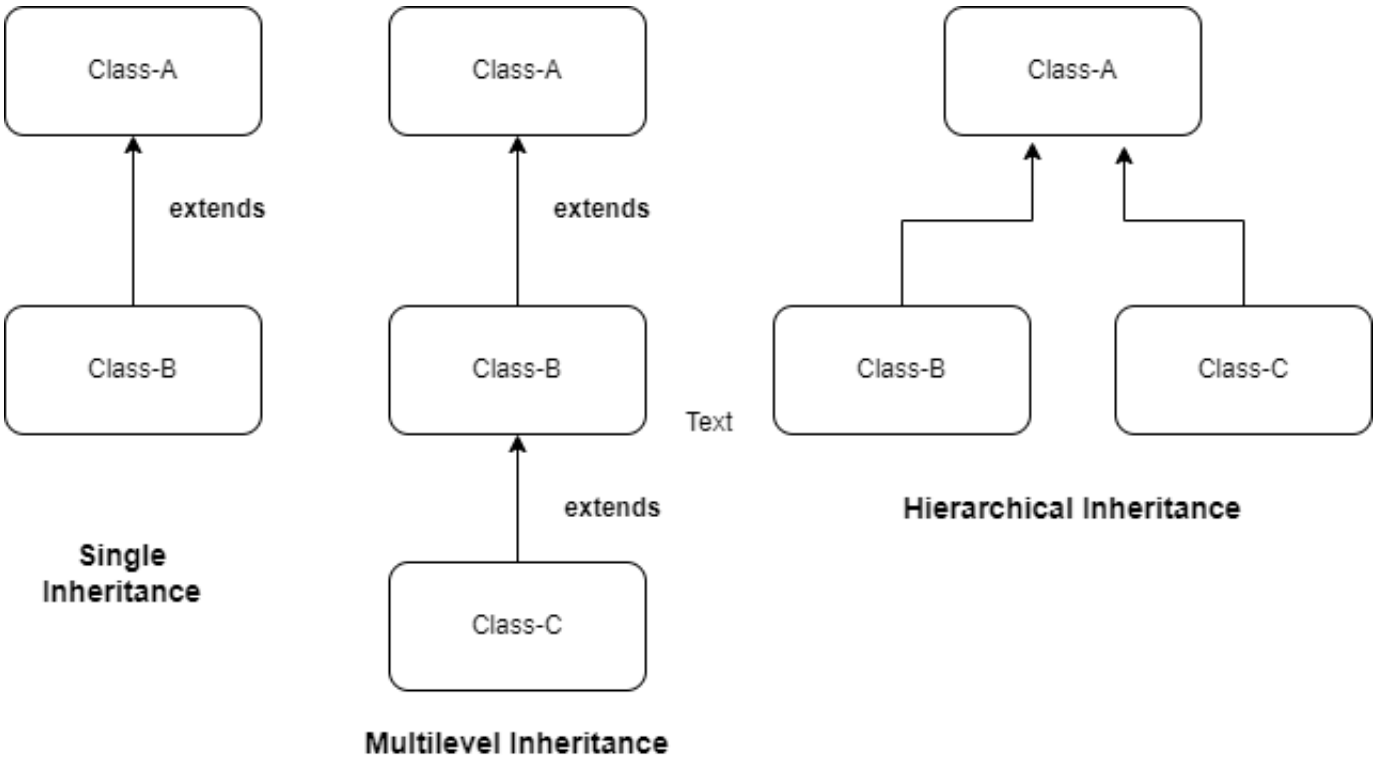


Figure 1.1: Types of Inheritance

1.3 Other Noteworthy Points

1. In Java, inheritance is an **is-a** relationship. That is, we use inheritance only if there exists an is-a relationship between two classes[2]
   - Car is a Vehicle
   - Orange is a Fruit
   - Surgeon is a Doctor
   - Dog is an Animal
2. Inheritance allows Derived-class to **override** the methods of Super-Class. That is, if the same method is present in both the superclass and subclass, then the method in the subclass overrides the method in the superclass. This concept is known as **Overriding**.
3. The overriding of methods from superclass can also be done with the help of **Super** keyword
4. In order to make the code cleaner, the data members can be defined using **protected** access modifiers, restricting the access to the variables only to the subclasses.
5. A subclass inherits all the members (fields, methods, and nested classes) from its superclass. **Constructors are not members**, so they are not inherited by subclasses, but the constructor of the superclass can be invoked from the subclass
6. **Multiple** and **Hybrid** inheritance is not supported in JAVA, but can be done using **Interfaces**

[2] https://www.programiz.com/java-programming/inheritance

1.4 Sample Program

A simple program demonstrating the usage of super keyword, inheritance, overriding is shown in the below code snippet for reference and structure is shown in figure 1.2

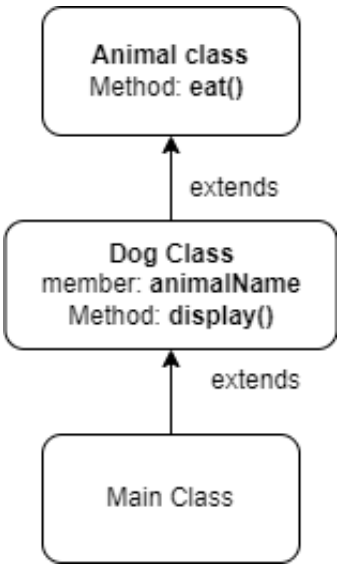


Figure 1.2: Program Structure

```
// Base Class
public class Animal {
   public void eat(){
         System.out.println("Animal Eat");
   }
}
// Derived Class
public class Dog extends Animal {
   String animalName = "Dog";
   public void display(){
      // calling the Animal class "eat method" using Super keyword
      super.eat();
      System.out.println("Animal name is: "+animalName);
   }
   // Overriding the eat method from Animal class
   @Override
   public void eat(){
      System.out.println("This is overriden Method");
   }
}

// Main Class
public class Main {

   public static void main(String[] args) {
      // write your code here
      Dog goldenRetreiver = new Dog();
      // Modifing the data member of the Dog class using Dog Operator
      goldenRetreiver.animalName ="Golden Retriever" ;
      goldenRetreiver.display(); // calling the Display method in Dog class
      goldenRetreiver.eat();
```

```
    }
}
```

1.5 Exercises

1. Write a Java Program to demonstrate Hierarchical Inheritance and document the code
2. What is the usage of Static keyword and Private keyword in Java. Demonstrate a program for the same
3. Illustrate the usage of Super Keyword for data member and member function access with a program

1.6 Submission Guidelines

- All the exercises must be executed, and a report comprising the Exercise number and the output should be presented in the document along with the name and USN (Neatly formatted)
- Don't take the snapshot of the code
- Font size : Heading - 16pt, Query title: 14 pt, Solution: 12 pt, Font: Times New Roman, Text aligned(Body Justified)
- The document should be submitted for evaluation in the google classroom Link and it will be evaluated for 10 Marks each

Chapter 2

Interfaces in JAVA

An Interface is an abstract class that comprises of **abstract/partially implemented** methods and data members. An interface in Java is a blueprint of a class. It has static constants and abstract methods.

1. The interface in Java is a mechanism to achieve abstraction
2. There can be **only abstract method**s in the Java interface, not method body
3. It is used to **achieve abstraction** and **multiple inheritance** in Java
4. Java Interface also represents the **IS-A** relationship
5. It is the duty of the class that implements the interface to implement the abstract methods of interfaces and it is must
6. Java provides "implements" method to implement and interface
7. An interface is declared by using the **interface** keyword

```
public interface <Interface Name> {
   // abstract value;
   // abstract method

}
```

2.1 Classes and Interfaces

A class extends another class, an interface extends another interface, but a class implements an interface as shown in figure 2.1.

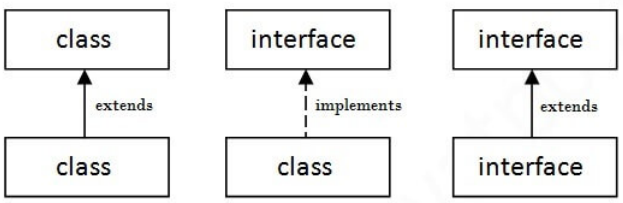


Figure 2.1: Classes and Interfaces

A sample program depicting the multiple inheritance via interfaces is as shown below figure 2.2,

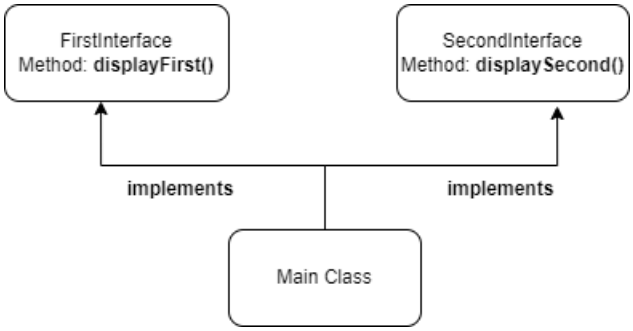


Figure 2.2: Program Structure

```
// First Interface
public interface FirstInterface {
   // abstract value
   public static String firstInterfaceValue = "ABC";
   // abstract method
   public void displayFirst() ;
}

// Second Interface
public interface SecondInterface {
   // abstract value
   public static String secondInterfaceVariable = "DEFD";
   // abstract method
   public void displaySecond();
}

// Main Program
public class Main implements FirstInterface, SecondInterface {

   public static void main(String[] args) {
      // write your code here
      Main m = new Main();
      m.displayFirst();
      m.displaySecond();
      System.out.println("FirstValue:"+FirstInterface.firstInterfaceValue);
      System.out.println("SecondValue:"+SecondInterface.secondInterfaceVariable);
   }
   // Implementing the Interface-I
   @Override
   public void displayFirst() {
      System.out.println("Implementing the First Interface");
   }
   // Implementing the Interface-II
   @Override
   public void displaySecond() {
      System.out.println("Implementing the Second Interface");
   }
}
```

2.2 Exercises

1. Write a Java Program to demonstrate Multiple Inheritance using 4 Interfaces
2. Try to write a program where one tries to modify the variable in the interface from the Main class. Check if it works or not, if not, Justify why

2.3 Submission Guidelines

- All the exercises must be executed, and a report comprising the Exercise number and the output should be presented in the document along with the name and USN (Neatly formatted)
- Don't take the snapshot of the code
- Font size : Heading - 16pt, Query title: 14 pt, Solution: 12 pt, Font: Times New Roman, Text aligned(Body Justified)
- The document should be submitted for evaluation in the google classroom Link and it will be evaluated for 10 Marks each

Chapter 3

Views and View-Group

In Android Layout is used to describe the user interface for an app or activity, and it stores the UI elements that will be visible to the user. An android app's user interface is made up of a series of View and ViewGroup elements. In most cases, android apps will have one or more operations, each of which is a single screen of the app. Multiple UI components will be present in the operations, and those UI components will be instances of the View and ViewGroup subclasses[1]. View is a basic building block of UI (User Interface) in android. A view is a small rectangular box that responds to user inputs. Eg: EditText, Button, CheckBox, etc. ViewGroup is an invisible container of other views (child views) and other ViewGroup. Eg: LinearLayout is a ViewGroup that can contain other views in it. ViewGroup is a special kind of view that is extended from View as its base class. ViewGroup is the base class for layouts.

3.1 View

The View class is the base class or we can say that it is the superclass for all the GUI components in android. View refer to the android.view.View class, which is the base class of all UI classes. android.view.View class is the root of the UI class hierarchy. So from an object point of view, all UI objects are View objects.

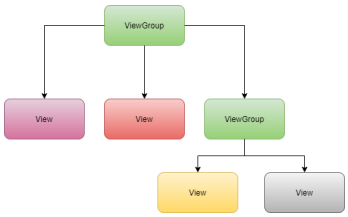
3.2 ViewGroup

The ViewGroup class is a subclass of the View class. And also it will act as a base class for layouts and layouts parameters. The ViewGroup will provide an invisible container to hold other Views or ViewGroups and to define the layout properties. For example, Linear Layout is the ViewGroup that contains UI controls like Button, TextView, etc., and other layouts also. A Viewgroup can also contain another viewgroups as well. The following figure 3.1a shows the relationship between the view and viewgroups.

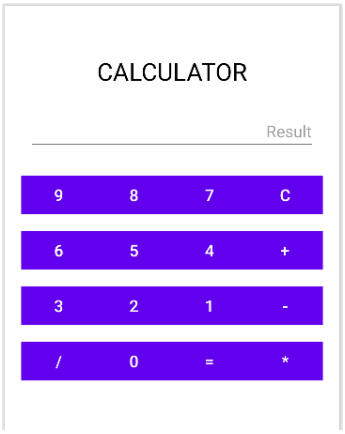
3.3 Exercise-I

Create a Basic Calculator App(Binary) to demonstrate the usage of views and view-groups.
The application Basic calculator works only for binary operators and is not suited for the complex expressions. To evaluate the expression we use custom logic in conjunction with regex. The app is implemented using LinearLayout(ViewGroup) which is a parent for the Views such as Button,

[1] https://www.geeksforgeeks.org/difference-between-view-and-viewgroup-in-android/



(a) View and ViewGroups



(b) Calculator Design

Figure 3.1: Views for Calculator

EditText, TextView and another LinearLayout as well. The front end of the calculator is as shown in the figure 3.1b,

The following are the steps,

- Follow the design shown in figure 3.2 for the activity_main.xml file and change the root element of the layout to LinearLayout and set the orientation to vertical as shown below,

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

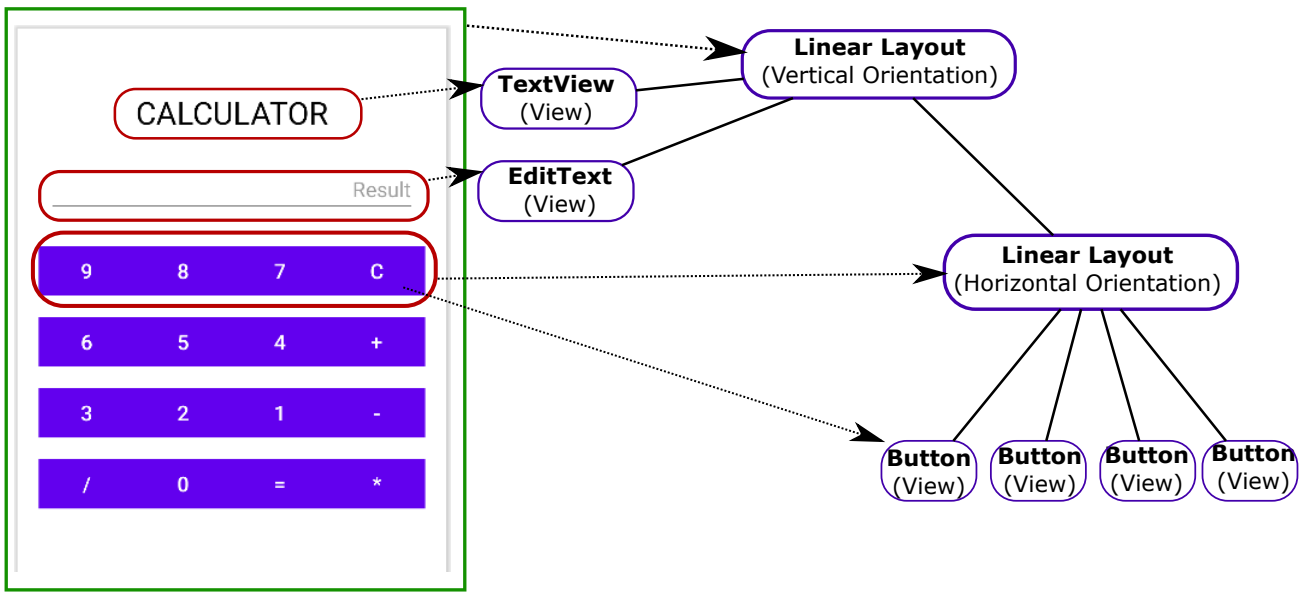


Figure 3.2: Layout Visualized

- Add an TextView(To display the name of the app), EditText(where the result of the calculation is shown as shown below,

```
<TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="60dp"
        android:id="@+id/label"
        android:text="CALCULATOR"
        android:textSize="30dp"
        android:textColor="@color/black"
        android:gravity="center"
        />

<EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="30dp"
        android:hint="Result"
        android:gravity="end"
        android:textSize="20dp"
        android:id="@+id/res"
        />
```

- To implement the next set of widgets, we define another LinearLayout with horizontal orientation under the parent layout after EditText which will act as a container for the first row buttons such as 9,8,7 and C Buttons. Define the buttons and use *layout_weight* attribute to share the width of the layout to four buttons(0.25 each). The implementation is as shown below,

```
<LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        android:layout_marginLeft="20dp"
        android:layout_marginRight="20dp"
        android:layout_marginBottom="20dp">

        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="9"
            android:textSize="20dp"
            android:id="@+id/nine"
            android:background="@color/white"
            android:layout_weight="0.25"
            />

        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="8"
            android:textSize="20dp"
            android:background="@color/white"
            android:id="@+id/eight"
            android:layout_weight="0.25"
```

```
            />
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="7"
            android:textSize="20dp"
            android:background="@color/white"
            android:id="@+id/seven"
            android:layout_weight="0.25"
            />
        <Button
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="C"
            android:textSize="20dp"
            android:background="@color/white"
            android:id="@+id/clear"
            android:layout_weight="0.25"
            />
</LinearLayout>
```

- On the similar lines, Repeat the above steps to implement the rest of the rows along with the respective buttons complying to the UI design shown in 5.4b.
- Open the Mainactivity.java make it implement View.OnClickListener, for which one needs to override a function that is onClick(View v).

```
public class MainActivity extends AppCompatActivity implements View.OnClickListener{
// logic of onCreate()
//....
@Override
public void onClick(View view)
{
   //Logic to Handle various buttons
}
//...
```

- In the Mainactivity.java, Instantiate JAVA object class for each of the views defined in the layout as follows,

```
Button one, two, three, four, five, six, seven, eight,nine;
Button plus, minus, div, sub, clear, equals;
EditText result;
String operatorPressed = " "; // keeps track of the operator pressed
```

- Inside the onCreate(), for each of the button defined above, equate the XML object with JAVA object using $findViewById(R.id.x)$ function as shown below,

```
result = findViewById(R.id.expression); //EditText
one = findViewById(R.id.one) ;
two = findViewById(R.id.two) ; //Buttons
 // do it for the rest of the buttons for numbers and operators as well
```

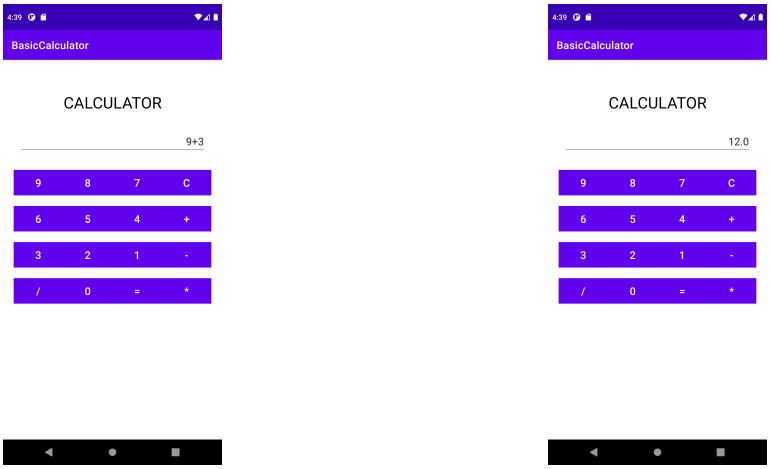
- Set the event listener for all of the buttons using this pointer as shown below,

```
// Buttons
one.setOnClickListener(this);
two.setOnClickListener(this);
three.setOnClickListener(this);
four.setOnClickListener(this);
five.setOnClickListener(this);
six.setOnClickListener(this);
seven.setOnClickListener(this);
eight.setOnClickListener(this);
nine.setOnClickListener(this);
// operators
plus.setOnClickListener(this);
minus.setOnClickListener(this);
mult.setOnClickListener(this);
div.setOnClickListener(this);
equals.setOnClickListener(this);
clear.setOnClickListener(this);
```

- Inside on the onClick override function, use the View.getID() and switch it to address the various buttons and perform the relevant actions(self commented)

```
double finalResult = 0.0;
      switch(view.getId())
      {
          case R.id.one: res.append("1");
              break;
          case R.id.two: res.append("2");
              break;
          // case for rest of the button 3-9 here
          case R.id.plus: res.append("+");
              operatorPressed="+";
              break;
          case R.id.minus: res.append("-");
              operatorPressed="-";
              break;
          case R.id.mult: res.append("*");
              operatorPressed="*";
              break;
          case R.id.div: res.append("/");
              operatorPressed="/";
              break;
          case R.id.equals: finalResult=
              evaluateExpression(res.getText().toString(),operatorPressed);
              res.setText(String.valueOf(finalResult));
              break;
              // evaluateExpression function is the function we use to compute the
                  expression it takes the expression and operatorpressed as input and
                  returns a string
          default:return;
      }
```

- Implement te evaluateExpression as follows, explanation is in comments



(a) Calculator Expression input (b) Result obtained on computation

Figure 3.3: Results of the Basic Calculator App

```
private double evaluateExpression(String res, String operatorPressed)
   {
       String[] tokens = res.split("\\+|-|\\*|\\/"); // split for +, -, *,/ operator
       // After split tokens[0] = first half of the string, tokens[1] = second half
           of the string
       double firstOperand = Double.parseDouble(tokens[0]); //convert string to double
       double secondOperand = Double.parseDouble(tokens[1]);
       switch(operatorPressed) // Switch the operator, compute and returs the result
           back to onclick that sets the editeext object to the result
       {
           case "+": return firstOperand + secondOperand;
           case "-": return firstOperand - secondOperand;
           case "*": return firstOperand * secondOperand;
           case "/": return firstOperand / secondOperand;
           default: return 0;
       }
   }
```

3.4 Output

The results of the app is shown in the figure 3.3. Figure 3.3a shows the landing page where as 3.3b shows the app screen after computing the expression.

Chapter 4

Android Activity Life-cycle

As a user navigates through, out of, and back to your app, the Activity instances in your app transition through different states in their lifecycle. The Activity class provides a number of callbacks that allow the activity to know that a state has changed: that the system is creating, stopping, or resuming an activity, or destroying the process in which the activity resides[1].Within the lifecycle callback methods, you can declare how your activity behaves when the user leaves and re-enters the activity.

4.1 Activity-lifecycle callbacks

To navigate transitions between stages of the activity lifecycle, the Activity class provides a core set of six callbacks: onCreate(), onStart(), onResume(), onPause(), onStop(), and onDestroy(). The system invokes each of these callbacks as an activity enters a new state.

1. *onCreate*()- You must implement this callback, which fires when the system first creates the activity. On activity creation, the activity enters the Created state. In the onCreate() method, you perform basic application startup logic that should happen only once for the entire life of the activity.

2. *onStart*()- When the activity enters the Started state, the system invokes this callback. The onStart() call makes the activity visible to the user, as the app prepares for the activity to enter the foreground and become interactive.

3. *onResume*()- When the activity enters the Resumed state, it comes to the foreground, and then the system invokes the onResume() callback. This is the state in which the app interacts with the user.

4. *onPause*()- The system calls this method as the first indication that the user is leaving your activity (though it does not always mean the activity is being destroyed); it indicates that the activity is no longer in the foreground (though it may still be visible if the user is in multi-window mode)

5. *onStop*()-When your activity is no longer visible to the user, it has entered the Stopped state, and the system invokes the onStop() callback. This may occur, for example, when a newly launched activity covers the entire screen. The system may also call onStop() when the activity has finished running, and is about to be terminated.

6. *onDestroy*()-onDestroy() is called before the activity is destroyed.

The overall process of using these callbacks are demonstrate in the figure 4.1,

[1] https://developer.android.com/guide/components/activities/activity-lifecycle

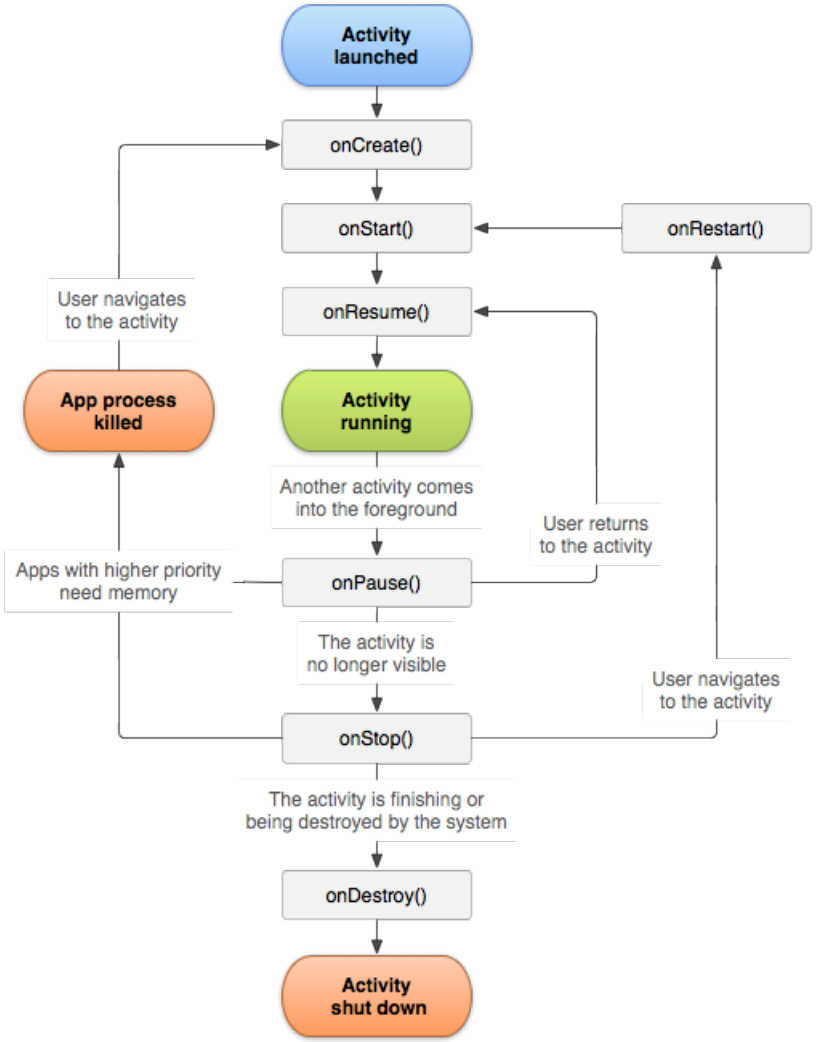


Figure 4.1: Android Activity Lifecycle

4.2 Exercise-I

Create an Android App to demonstrate the Working of Android Activity LifeCycle. The following are the steps,

- Create an Empty Project, By default, onCreate() function will be created, Add a Toast message after the super keyword in onCreate as shown below,

```
@Override
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Toast.makeText(this, "ACTIVITY CREATED", Toast.LENGTH_SHORT).show();
}
```

- Press CTRL+O after the oncreate() Function, A menu will pop up. Choose the override methods such as onStart(), onResume(), onPause(), onStop(), and onDestroy() resepectively as shown in figure 4.2 ,

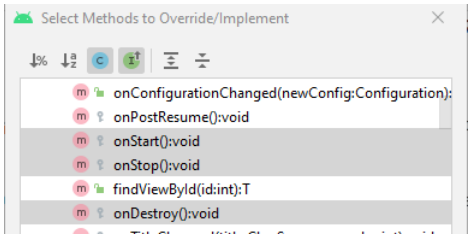


Figure 4.2: Selecting Override functions

- Add Toast messages in the respective callbacks such as onStart(), onResume(), onPause(), onStop(), and onDestroy() as shown below for onStart() and onPause().

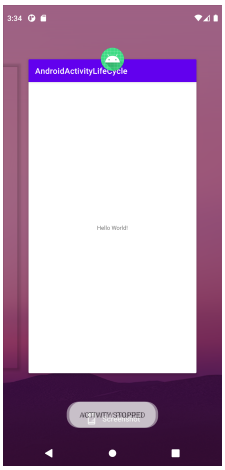
```
@Override
protected void onStart() {
        super.onStart();
        Toast.makeText(this, "ACTIVITY STARTED", Toast.LENGTH_SHORT).show();
}

@Override
protected void onPause() {
        super.onPause();
        Toast.makeText(this, "ACTIVITY PAUSED", Toast.LENGTH_SHORT).show();
}
```

- Similarly, Implement the Toast messages for onResume(), onStop(), and onDestroy() respectively.

4.3 Output

Sample output window of the app is as shown below figure 4.3,



(a) Activity Paused

(b) Activity Resumed

Figure 4.3: Activity CALLBACKS

Chapter 5

Intent

An Intent is a messaging object you can use to request an action from another app component. Although intents facilitate communication between components in several ways, there are three fundamental use cases, They are,

- Starting an activity
- Starting a service
- Delivering a broadcast

An intent is to perform an action on the screen. It is mostly used to start activity, send broadcast receiver,start services and send message between two activities. There are two intents available in android as *Implicit* Intents and *Explicit* Intents. The implicit intents are used to invoke an activity which is part of the Android system(Ex: Dialer, Messaging Activity). The explicit intents are used to invoke an activity which is user defined. At the same time the intents are used to transfer the data between activities wrapped in the container called *Bundle*. The overall process is visualized as shown in figure 5.1

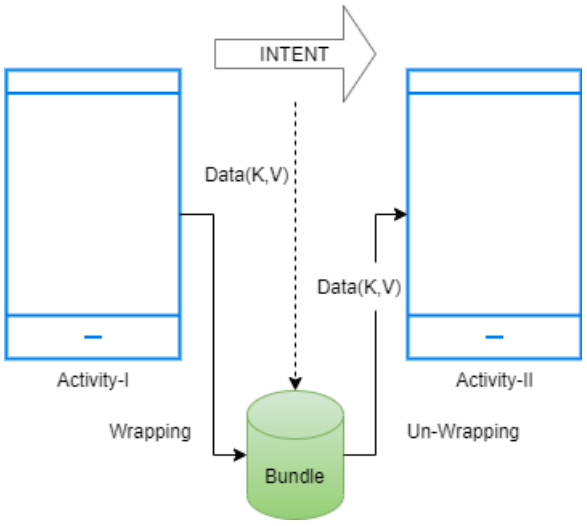


Figure 5.1: Intent overview

The Activity-II can be an activity either a user-defined activity or an implicit android activity, the choice differentiates the explicit and implicit intent. To create an implicit intent, the following code snippet is used,

```
Intent it = new Intent(Context ctx, Activity Destination_Activity);
startActivity(it);
```

where the destination activity is the activity that will be invoked from the current activity using context object. To Start the Activity via Intent, *startActivity*(*intent*) is called. The intent can carry the data with the help of bundle. To do the first create *Bundle* class object and then using that object call the function *putString*(*key*, *value*) to pass the value in the form of key-value pair. The following code snippet shows the same

```
Bundle b = new Bundle() ;
b.putString("name",name.getText().toString());
it.putExtras(b) ;
```

5.1 Exercise-I:Explicit Intent

Create an Android application to demonstrate the working of Explicit Intent
The following are the steps followed,

- Open the activity_main.xml, Create a LinearLayout with vertical orientation. Add an EditText and Button laid out one below the other. The following code snippet shows the same,

```
<EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/name"
        android:hint="Enter your name"/>
<Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="CLICK"
        android:id="@+id/click"/>
```

- Create a SecondActivity by right clicking on the project "app" and select new->Activity->Empty Activity, set the name as SecondActivity.java and the xml file activity_second.xml as shown in figure 5.2.

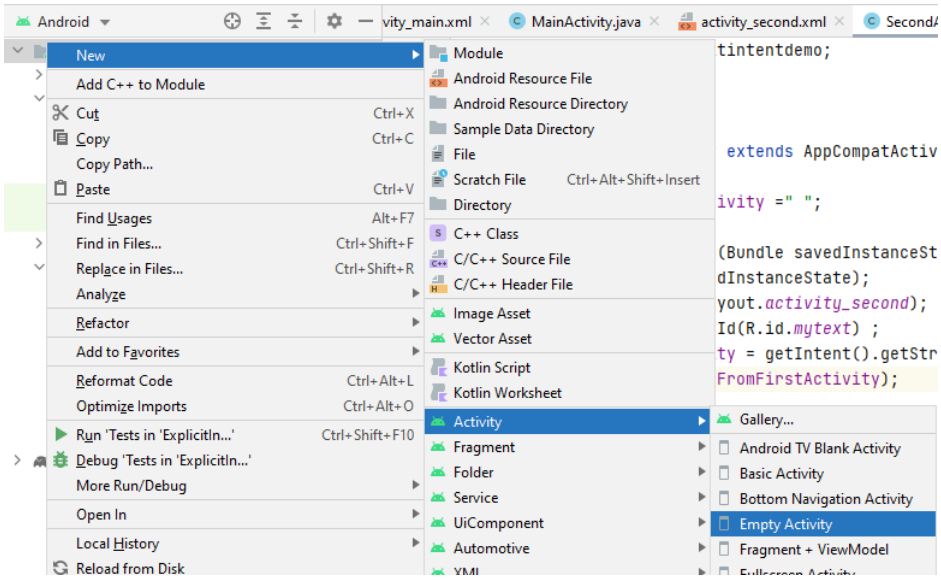


Figure 5.2: Creating a Second Activity

- In the activity_secondẋml, Set the layout to LinearLayout and add a textview as a component as shown below,

```
<TextView
       android:layout_width="match_parent"
       android:layout_height="wrap_content"
       android:id="@+id/mytext"
       android:text="NAME COMES HERE"
       android:textSize="40dp"
       android:textAlignment="center"
       />
```

- In the MainActivity Java file, Instantiate the java objects for the ListView and Button as follows, Instantiate the objects in the *oncreate* function too.

```
EditText name ;
Button click ;
// inside onCreate function
name = findViewById(R.id.name) ;
click = findViewById(R.id.click);
```

- Add an event *onClickListener* to the button "click". Upon click, use the Intent object to navigate from *MainActivity* to *SecondActivity* class with a string retrieved from the *EditText*(*MainActivity*) as shown below. The retrieved string is passed along with the key "name" into the Bundle object before passing the bundle to Intent. To pass the bundle to intent object use putExtra(Bundle b).

```
click.setOnClickListener(new View.OnClickListener() {
           @Override
           public void onClick(View v) {
               Intent it = new Intent(MainActivity.this, SecondActivity.class);
               Bundle b = new Bundle() ;
               b.putString("name",name.getText().toString());
               it.putExtras(b) ;
               startActivity(it);
           }
       });
```

- In the SecondActivity define the JAVA equivalent objects for the XML TextView object. In the oncreate function, equate the XML object with JAVA object. Use *getIntent*(). *getStringExtra*(*Stringkey*) function to get the retreive the string using the key("name") passed with Intent. Once the string is retrieved, set the text to the TextView object using *setText*(*string*) function as shown below,

```
TextView myName ;
String nameFromFirstActivity =" ";

//inside oncreate
myName = findViewById(R.id.mytext) ;
nameFromFirstActivity = getIntent().getStringExtra("name");
myName.setText(nameFromFirstActivity);
```

5.2 Exercise-II: Implicit Intent

Create an Android application to demonstrate the working of Implicit Intent
The app demonstrated here uses CALL_PHONE permission to let the user to enter the Phone number and puts the number to the inbuilt dialer activity of android system. The steps are as shown below,

- Navigate to manifests directory, open the AndroidManifest XML file and add the following uses-permission(before the application tag) as shown below,

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

- Go to the activity_main XML, change the layout to LinearLayout with vertical orientation. Add an EditText to receive a phone number and Button for handling event click as shown below,

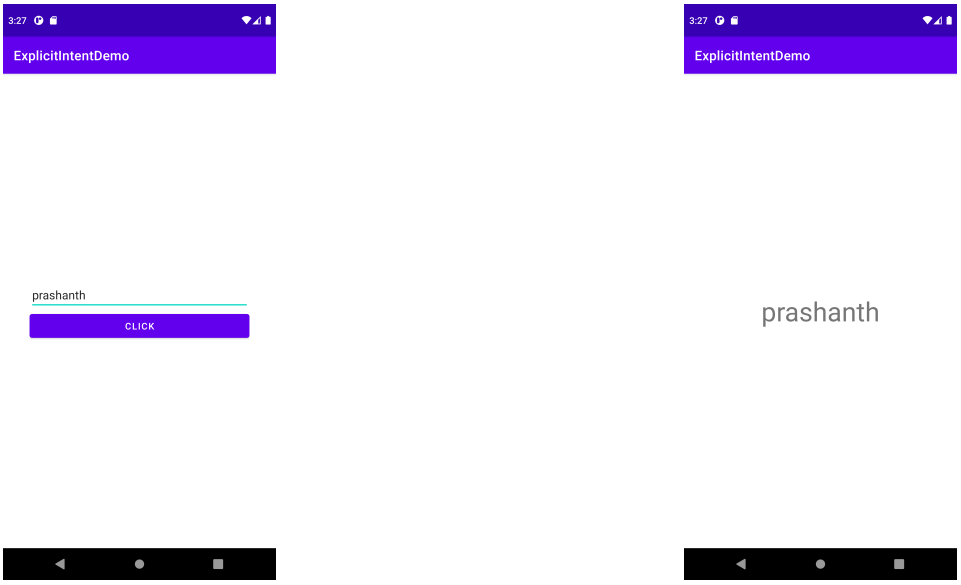
```
<EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/myphone"
        android:hint="Enter the Phone Number"
        android:layout_margin="20dp"
        />
<Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/call"
        android:text="CALL"
        android:layout_margin="20dp"/>
```

- In the MainActivity Java file, Instantiate the java objects for the EditText and Button as follows, Instantiate the objects in the *oncreate* function too.

```
EditText phoneNumberToCall ;
Button call ;
// inside onCreate function
call = findViewById(R.id.call) ;
phoneNumberToCall = findViewById(R.id.myphone) ;
```

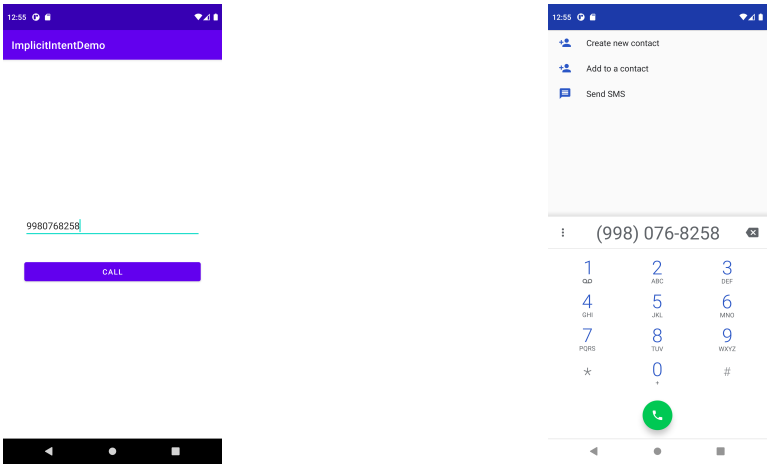
- Add an event *onClickListener* to the button "click". Upon click, use the Intent object to navigate from *MainActivity* to Implicit Activity that is "Dialer" in this case. The inbuilt activity can be invoked using "Intent.ACTION_DIAL" action. The intent constructor takes two arguments they are, the *intent.ACTION*, and the URI resource(PhoneNumber as URI) which is as shown below.

```
call.setOnClickListener(new View.OnClickListener() {
          @Override
          public void onClick(View v) {
              Uri myUri = Uri.parse("tel:"+phoneNumberToCall.getText().toString());
              Intent it = new Intent(Intent.ACTION_DIAL, myUri) ;
              startActivity(it);
          }
      });
```



(a) Explicit Intent Front Activity (b) Explicit Intent Second Activity

Figure 5.3: Explicit Intent



(a) Implicit Intent Front Activity (b) Implicit Intent Second Activity

Figure 5.4: Implicit Intent

5.3 Output for Exercise-I/II

Refer the figure 5.3 for the output of the Explicit Intent, where in the 5.3a shows the front page of the App and 5.3b shows the SecondActivity created in the app.

Refer the figure 5.4 for the output of the Implicit Intent, where in the 5.4a shows the front page of the App and 5.4b shows the Dialer Activity of android system to which the Phone number is passed via Intent.

Chapter 6

ImageView

ImageView class is used to display any kind of image resource in the android application either it can be android.graphics.Bitmap or android.graphics.drawable.Drawable. [1]. ImageView is a form of container which holds the image from a particular source, usually from the drawable folder. The sample XML code for adding the ImageView is as shown below,

```
<ImageView
        android:id="@+id/myImg"
        android:layout_width="match_parent"
        android:layout_height="400dp"
        android:src="@drawable/img_name"
         />
```

The properties of the source can be dynamically changed with the help of JAVA controller code as well.

```
ImageView imageView.setImageResource(R.drawable.night); //night.png is the image in
    drawable directory
```

6.1 Exercise

Create an ImageSwitcher which switches the images from the Drawable folder controlled by the button. Following are the steps,

- Download two android background images from internet, save them as first.png and second.png respectively.
- Copy and Paste the 2 downloaded images into drawable folder under res directory.
- Open the activity_main.xml, Change the Layout to LinearLayout and add an ImageView widget.

  ```
  <ImageView
          android:id="@+id/myImg"
          android:layout_width="match_parent"
          android:layout_height="400dp"
          android:src="@drawable/first" // by default first.png is displayed
          android:layout_margin="20dp"/>
  ```

[1] https://www.geeksforgeeks.org/imageview-in-android-with-example/

- In the MainActivity.java, Instantiate the ImageView and Button object respectively and equate them with XML objects that forms the view as shown below. Also add an eventListener to the Button as well and override the onclick function.

```
ImageView imageView;
Button change ;
int flag = 1 ; // A flag that keep tracks of the image being displayed

// Inside onCreate()
imageView = findViewById(R.id.myImg) ;
change = findViewById(R.id.click);
change.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
                // Code to change the image
    }
});
```

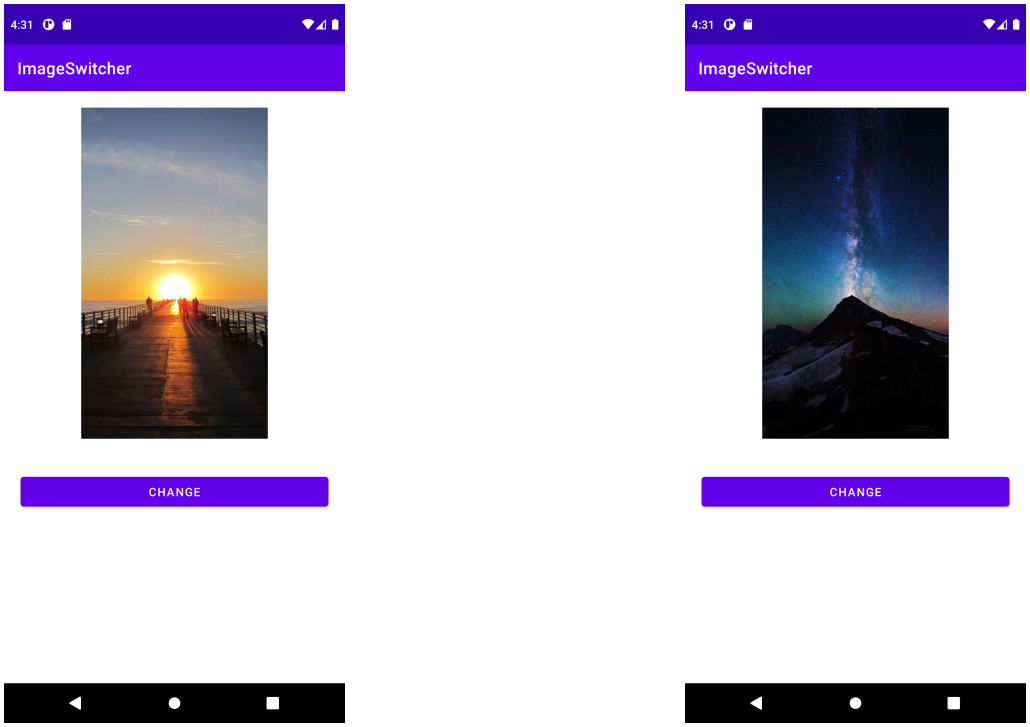
- Inside the onClick function control the image with the help of flag(flag = 1/2, depending on the image being displayed). Use the ImageView.setImgResource(drawable resource) function to change the image from the controller code as shown below,

```
change.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if(flag==1){
                    imageView.setImageResource(R.drawable.night);
                    flag=2 ;
                }else{
                    imageView.setImageResource(R.drawable.morning);
                    flag=1 ;
                }

            }
        });
```

6.2 Output

The following 6.1a shows the App Landing Screen on button click the image gets changed into second as shown in figure 6.1b.



(a) Imageview displaying the first Image (b) Imageview displaying the Second Image

Figure 6.1: ImageFlipper App

Chapter 7

AlertDialog

A Dialog is small window that prompts the user to a decision or enter additional information[1]. To create an alert Dialog, Android system provides an Interface called as, *AlertDialogBuilder* which an inner class of *AlertDialog*. AlertDialog.Builder provides wide range of properties to tweak on how the Alert Message to be presented to the user. Some of the properties that can be customized are,

1. Title of the Alert using setTitle(String title) function
2. Icon of the Alert using setIcon(Drawble_resource) function
3. Cancellable or not feature that lets user to be able to cancel the dialog or not using setCancellable(boolean) function
4. Set a Message using setMessage(CharSequence message)
5. To set positive and negative button to address YES/NO type input using *setPositiveButton*()& *setNegativeButton*() respectively.
6. finish() closes the AlertDialog
7. AlertDialog.show() shows the Alert Window

To Create an alertdialog, One needs to instantiate AlertDialog.Builder class as follows,

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

Once the AlertDialog object is created, we can now set the attributes such as title, message, positive/negative button as follows,

```
builder.setIcon(R.drawable.ic_launcher_background) ; //Sets the icon from Drawable folder
builder.setTitle("AlertDemo") ; // Sets the Title Message
builder.setMessage("You want to Stay, Y/N?") ;
builder.setPositiveButton("Yes", new DialogInterface.OnClickListener() {
     @Override // Positive Button with Event Listener
     public void onClick(DialogInterface dialog, int which) {
        Toast.makeText(MainActivity.this, "You have pressed yes",
            Toast.LENGTH_SHORT).show();
     }
});
```

[1] https://www.tutorialspoint.com/android/android_alert_dialoges.htm

7.1 Exercise-I

Create an Android App to demonstrate the Working of AlertDialog. The following are the steps,

- Open up activity_main.xml file, Change the Layout to LinearLayout and set the orientation to vertical. Add a Button that can generate an alert under the LinearLayout as shown below,

```
<Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/alert"
        android:layout_marginRight="30dp"
        android:layout_marginLeft="30dp"
        android:text="CLICKTOALERT"
    />
```

- Create the necessary JAVA objects in the MainActivity.java file and equate the JAVA object with XML object and add an onClick Event Listerner as well.

```
Button alertButton ;
// inside onCreate()
alertButton = findViewById(R.id.alert) ;
alertButton.setOnClickListener(new View.OnClickListener() {
     @Override
     public void onClick(View v) {
      //logic for Alert
     }
});
```

- Inside the onClick() function, Create the AlertDialog.Builder object using the constructor and pass the context as an argument for it.

```
AlertDialog.Builder builder = new AlertDialog.Builder(MainActivity.this) ;
```

- Customize the AlertDialog and its properties as shown below,

```
builder.setIcon(R.drawable.ic_launcher_background) ;
builder.setTitle("Alert Title") ;
builder.setMessage("Shall I Close this window, Y/N?") ;
builder.setPositiveButton("Yes", new DialogInterface.OnClickListener() {
@Override
public void onClick(DialogInterface dialog, int which) {
      Toast.makeText(MainActivity.this, "You have pressed yes",
          Toast.LENGTH_SHORT).show();
}
});
builder.setNegativeButton("No", new DialogInterface.OnClickListener() {
@Override
public void onClick(DialogInterface dialog, int which) {
      finish() ;
}
});
```

- Finally, To Create an AlertDialog we use the AlertDialog.Builder class to customize the AlertWindow and then we use AlertDialog.builder.create() to create an AlertDialog object which can be equated accordingly. this is shown below,

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

7.2 Output

The alert window of the result is as shown below figure 7.1,

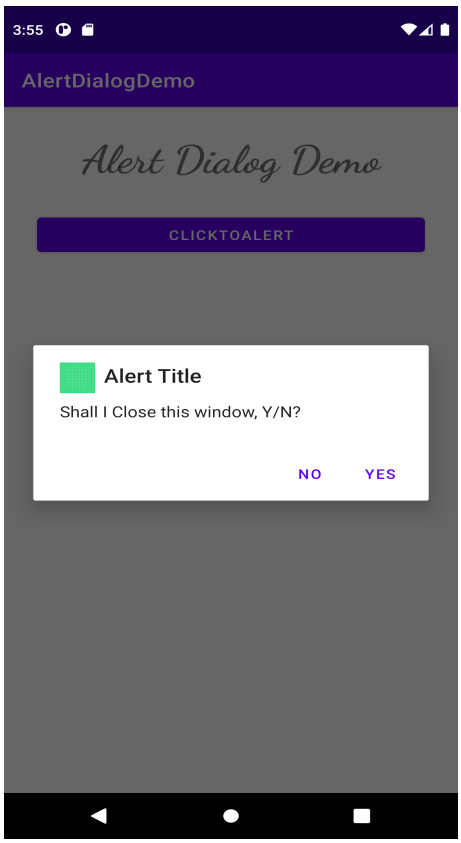


Figure 7.1: Alert Dialog

Chapter 8

ListView-Android Widgets(View)

Listview is a view which groups several items and displays them in the vertical scrollable list.The list items are automatically inserted to the list using an entity called as Adapters. Adapters can be of many types, some of the commonly used adapters are

- ListAdapters
- ArrayAdapters

An array adapter recieves entries from the data source which can be a string array or database and creates a view for each of the elements with a specified layouts. Adapter acts as an interface between the data source and the UI component. The common way to define the Array adapter is as follows,

```
ArrayAdapter<type> adapter= new ArrayAdapter<type>(context, layout_to_inflate,
    datasource) ;
listview.setAdapter(adapter);
```

8.1 Exercise

Create an Android Application to demonstrate the working of the Listview by creating a country List. On each item click, Display the Item clicked as a Toast Message.
Following are the steps followed to implement the ListView

1. *Create the Data Source* Navigate to the resource folder, under which go to values folder, open the string.xml and add the following lines

```
<string-array name="country">
        <item>India</item>
        <item>China</item>
        <item>Pakistan</item>
        <item>USA</item>
        <item>UK</item>
</string-array>
```

2. Create a Listview in activity _mainẋml Change the Layout of the Main activity as LinearLayout and add the following code snippet.

```
<ListView
        android:layout_width="match_parent"
```

```
        android:layout_height="wrap_content"
        android:id="@+id/list"/>
```

3. Create a Layout for each of the view component of the List as textview_layout.xml. Add the following code snippet.

```
<?xml version="1.0" encoding="utf-8"?>
<TextView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:text="Content"
    android:textSize="40dp"
    />
```

4. Instantiate the ListView object in the Mainactivity.java and create a String array as follows,

```
ListView myList ;
String countryList[] ;
```

5. On the oncreate(), assign the XML object of the listview to the Listview object of the JAVA. At the same time, fetch the resource array from string.xml and assign it to the countryList[] array as follows,

```
myList = findViewById(R.id.list) ;
countryList = getResources().getStringArray(R.array.country) ;
```

6. Create a Array Adapter class and pass the context, layout for each view, data-source as arguments.

```
ArrayAdapter<String> arrayAdapter = new ArrayAdapter<String>(this,
    R.layout.textviewlayout, countryList);
```

7. Add the array adapter to the Listview using setAdapter() function,

```
myList.setAdapter(arrayAdapter);
```

8. Finally add the event Listener called onItemClickedListener to the List items as follows,

```
myList.setOnItemClickListener(new AdapterView.OnItemClickListener() {
@Override
public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
Toast.makeText(MainActivity.this, "You have
    clicked:"+parent.getItemAtPosition(position), Toast.LENGTH_SHORT).show();
}
});
```

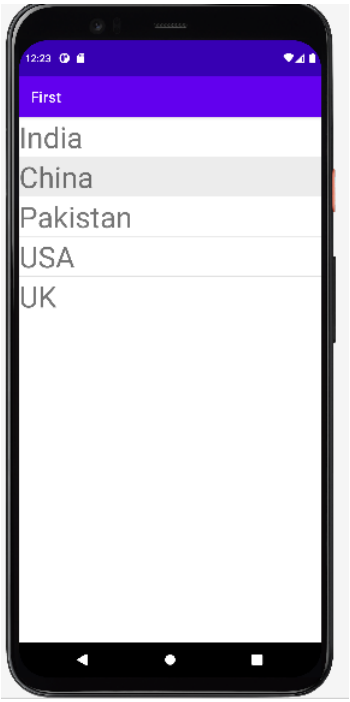


Figure 8.1: Output

Chapter 9

MediaPlayer-Playing Audio File

The Android multimedia framework includes support for playing variety of common media types, so that you can easily integrate audio, video and images into your applications. You can play audio or video from media files stored in your application's resources (raw resources), from standalone files in the filesystem, or from a data stream arriving over a network connection, all using MediaPlayer APIs. One of the most important components of the media framework is the MediaPlayer class. An object of this class can fetch, decode, and play both audio and video with minimal setup. It supports several different media sources such as,

- Local resources
- Internal URIs, such as one you might obtain from a Content Resolver
- External URLs (streaming)

Some of the APIs which comes under the MediaPlayer class are,

- To use the existing mediaplayer in the android, Android system provides MediaPlayer Class. Instantiate the object of this class as follows, it takes two arguments, they are the context and source of the file in the raw folder.

```
MediaPlayer mediaPlayer = MediaPlayer.create(context, source_file_mp3) ;
```

- *MediaPlayer* class provides a bunch of APIs Which can be used with mediaplayer object, they are
  - *mediaPlayer.play*(): To play the audio file
  - *mediaPlayer.pause*(): To pause the audio file
  - *mediaPlayer.reset*(): To move to the start of the audio file
  - *mediaPlayer.stop*(): To stop the audio file from playing
  - *mediaPlayer.seekTo*(*position_in_millisecond*): To move to the location specified in the audio file in millisecond
  - *mediaPlayer.getCurrentPostion*(): To retrieve the current position of the audio file while playing
  - *mediaPlayer.release*(): Releases the audioplayer resources that is the audiofile
  - *mediaPlayer.getDuration*(): To get the full duration of the audio file

9.1 Exercise

Create a media player application that will play media file saved on Android Phone. Demonstrate application with play, pause, fast forward, and rewind functionality

The following are the steps followed,

1. Create a raw folder under res directory and add the MP3 file with user specific name as shown in the figure 9.1,

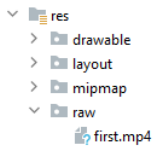


Figure 9.1: Folder Hierarchy

2. Create the layout as shown below in Figure 9.2 using LinearLayout with vertical orientation with buttons for play, pause, forward, rewind,restart and stop operation. The sample code for

Figure 9.2: Folder Hierarchy

one button in activity_mainẋml is as shown below,

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity"
    android:orientation="vertical"
    android:background="#7712">
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/songname"
        android:id="@+id/songname"
        android:layout_margin="30dp"
        android:fontFamily="cursive"
        android:textSize="40dp"
        android:gravity="center"
        android:textStyle="bold"
        />
    <Button
```

```
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginLeft="30dp"
    android:layout_marginRight="30dp"
    android:text="Play"
    android:id="@+id/play"/>
   </LinearLayout>
```

3. In the Mainactivityjava file, define the equivalent java objects as follows,

```
Button play, forward, rewind, pause, stop, reset;
MediaPlayer mediaPlayer ;
int starttime = 0 ; // startime is 0s
int stopttime = 0; // stoptime is 0s by default
int forwardtime = 5000 ; // 5s for forward
int backwardtime = 5000 ; // 5s for backwardtime
```

4. Instantiate *MediaPlayer* class and refer to the media file in the raw folder(Suppose the name is first.mp3) refer it below. Add an event click listener for all of the buttons as follows, Implement the same for pause, stop using play code as a template

```
play.setOnClickListener(new View.OnClickListener() {
   @Override
   public void onClick(View v) {
   Toast.makeText(MainActivity.this, "Playing Media now", Toast.LENGTH_SHORT).show();
   mediaPlayer = MediaPlayer.create(this, R.raw.first) ;
   mediaPlayer.start();
   }
});
reset.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
        mediaPlayer.seekTo(starttime);
        mediaPlayer.start();
}
});
```

5. To implement the forward operation, first we need to determine the current position in which the audioplayer is playing which can be accessed using $mediaPlayer.getCurrentPostion()$ call.The full duration of the audiofile can be obtained by calling $mediaPlayer.getDuration()$. The audio can be forwarded only when the $(currentTime+forwardTime) < mediaPlayer.getDuration()$ which is as shown below,

```
forward.setOnClickListener(new View.OnClickListener() {
   @Override
   public void onClick(View v) {
   int currentpos = mediaPlayer.getCurrentPosition() ;
   if((currentpos+forwardtime) <= (stopttime = mediaPlayer.getDuration())){
          mediaPlayer.seekTo(currentpos+forwardtime);
    }}
});
```

6. Implement the backward operation on the similar lines.

Chapter 10

Introduction to SQLite Database

SQLite Databases are regular Relational DBMS designed for lower or mid range devices. SQLite databases are used when one wants a database to leave very low memory imprint and with fewer basic operations only. Some of the features of SQLite databases are listed below,

1. SQLite database along with the packages sums up to the size of 600Kb which is suitable for mobile devices
2. SQLite databases are serverless unlike MySQL or Oracle databases
3. In SQLite datbases, the data is encrypted and stored on to regular files and the read-write operations are directly performed as if one accessing the files which makes the database run very agile.
4. SQLite is ACID complaint, does not support advanced concepts such as Triggers
5. SQLite databases are written in C and is part of android libraries, there is no need to install the package explicitly

10.1 SQLiteOpenHelper Class

The APIs related to SQLite database are managed by an Android component called SQLite Manager. It is the component which provides the functions, methods, & classes for accessing, manipulating and versioning of SQLite databases. The SQLite Manager provides a class to do all the operations on the database called as *SQLiteOpenHelper* Class. To use the features of the SQLiteOpenHelper class, the model class must extend and make the class as inheritor of the features from Super class SQLiteOpenHelper as shown below,

```
public class DBHelper extends SQLiteOpenHelper{}
```

Once the custom class DBHelper extends SQLiteOpenHelper, One must add a Constructor, and override functions such as shown below,

1. DBHelper constructor takes context, Database name, cursor factory, Database version as arguments, and override the functions. The constructor should override the constructor of the super class taking same params as the constructor as arguments as shown below,

```
public DBHelper(@Nullable Context context,@Nullable String name,@Nullable
    SQLiteDatabase.CursorFactory factory,int version) {
        super(context, dbName, null, dbVersion);
    }
```

The context specifies the current class context, The second argument is a Database name which should be defined earlier as String and the CursorFactory specifies whether we are using default cursor object provided by SQLiteOpenHelper class or using custom cursor. By default, this object is set to NULL to tell that we are using default cursor object. The final argument is used for the database version.

2. The second function to be override by extending the SQLiteOpenHelper class is called as onCreate(SQLiteDatabase db) that takes SQLiteDatabase object pointing to the current database. It is called only once when the DB is created for the first time. The override function is as shown below,

```
@Override
public void onCreate(SQLiteDatabase db) {
   // Logic to create Database
}
```

3. Another default override function is onUpgrade(SQLiteDatabase db, int prev_version, int new version), which is used to upgrade database to another version or any preliminary sanitisation function like dropping tables if already exists, etc.

```
@Override
public void onUpgrade(SQLiteDatabase db, int prev_version, int new version) {
   // Logic to upgrade Database & perform preliminary setups
}
```

4. Similar to onUpgrade, the database can be downgraded using onDowngrade function which recieves the same arguments as onUpgrade and syntax is similar as well. It is shown below.

```
@Override
public void onDowngrade(SQLiteDatabase db, int old_version, int new version) {
   // Logic to downgrade Database
}
```

5. Another function that is provided by SQLiteOpenHelper is close(). which is used to close the database upon the completion of operations on database.

10.2 SQLiteDatabase Class

The Android System also provides SQLiteDatabase class that facilitates the process of table creation, and basic CRUD operation on the database. The following are some of the member functions of SQLiteDatabase class that are often used,

1. execSQL function is used to execute raw SQL statements which is called by SQLiteDatabase object

```
SQLiteDatabase db ; //instance of SQliteDatabase object
db.execSQL(String SQL_statement) ; // SQL statement
```

2. getReadableDatabase() and getWritableDatabase() functions are used to open the Database file in Read and Write/ mode respectively. Both are demonstrated here

```
SQLiteDatabase db = = context.getWritableDatabase(); // DB is opened in Write mode
SQLiteDatabase db = = context.getReadableeDatabase(); // DB is opened in Read mode
```

3. To insert data into database, data is actually inserted in the form of (Key,value) pair which is prepared by the ContentValue object. The default values can also be made null with the help of nullColumnHack argument. The API is as shown below,

```
long insert(String tablename, String nullColumnHack, ContentValues values);
```

4. To delete, we use delete API, which takes three arguments, they are tablename, where clause and arguments for the where arguments. It is as shown below,

```
int delete(String table, String whereClause, String[] whereArgs);
```

5. To update, we use update API, which takes three arguments, they are tablename, where clause and arguments for the where arguments. It is as shown below,

```
int update(String table, ContentValues values, String whereClause, String[]
    whereArgs);
```

6. to execute a query we use query function which returns a cursor object. A cursor object points to the first row output of the SQL statement that got executed and its iteratable. The query functions uses arguments such as tablename, column name, selection and its arguments, groupby, having and orderby arguments respectively depending upon the user choices. Not all the arguments are usable at the same time and they can be set to null if its not being used.

```
Cursor cur = query(String table, String[] columns, String selection, String[]
    selectionArgs, String groupBy, String having, String orderBy, String limit);
```

7. rawQuery function is also used to execute the SQL query that returns a cursor object to the ResultSet.

```
Cursor cur = rawQuery(String sql, String[] selectionArgs);
```

10.3 Exercise

Create a Login and Registration App using SQLite database which provides basic CRUD operations such as,

- Insert a new user
- Delete an existing user upon inputting the name
- Update the existing user
- Display all the users in the form of a String

The following are the steps followed to create the Application.

1. Create 4 different Activities for the operations Register(insert), Delete, Update and Display operation respectively. This is done by right clicking on project and selecting new->Activity->Empty Activity. The following files are created upon creating 4 activities.

- activity_registration.xml & Registration.java for Registration page
- activity_display_page.xml & DisplayPage.java for Display Activity
- activity_updatepage.xml & UpdatePage.java for Update Activity
- activity_deletepage.xml & DeletePage.java for Delete Activity

2. Prepare a Separate class encapsulating all the functions on the database. Call this class as DBHelper Class. Do the following steps,

   (a) Make the DBHelper Class extend the SQLiteOpenHelper Class

   (b) Define the variables such as tablename, database name, version.

   (c) Add the constructor matching the SQLiteOpenHelper Class and add the user defined database name and version

   (d) Override the methods of SQLiteOpenHelper such as onCreate() and onUpgrade() which are mandatory

   After completing all the above steps, the code looks as follows,

```
public class DBHelper extends SQLiteOpenHelper{
private static final String dbName="studentdb"; // Database Name
private static final String tbName="student" ; // Table Name
private static final int dbVersion = 1 ; // Version - 1 indicating first version

// Constructor matching the SQLiteOpenHelper class
public DBHelper(@Nullable Context context,@Nullable String name,@Nullable
    SQLiteDatabase.CursorFactory factory,int version) {
        super(context, dbName, null, dbVersion);
    }
//  onCreate()
@Override
public void onCreate(SQLiteDatabase db) {
   // Logic to create Database
}
// onUpgrade()
@Override
public void onUpgrade(SQLiteDatabase db, int prev_version, int new version) {
   // Logic to upgrade Database & perform preliminary setups
}
}
```

3. Create the table in the onCreate function using execSQL function as follows,

```
@Override
public void onCreate(SQLiteDatabase db) {
       db.execSQL("CREATE TABLE "+tbName+"(uname VARCHAR(10),passw
           VARCHAR(10))"+";");
       //SQL - CREATE TABLE tbName(uname VARCHAR(10),passw VARCHAR(10));
}
```

4. Implement on onUpgrade function. If the table already exists, then we will drop the table and then we recreate the table again by calling onCreate function as follows,

```
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
```

```
        db.execSQL("DROP TABLE IF EXISTS "+tbName);
        onCreate(db);
}
```

5. Implement insert function as addUser function that takes name and usn as arguments. We perform the following steps

   (a) Create a SQLiteDatabase and open it in Writeable mode using getWritableDatabase()

   (b) Prepare the Data for insert as (key,value) pair. The key would be the column name and value will be the value for the name. This is done by instantiating the ContentValues.

   (c) Call the SQLiteDatabase.insert function passing tablename, nullcolumnHack and content values. The return value is in the long type, store it to a long variable

   (d) close the database and return the long value. The steps are shown below,

```
public long adduser(String name, String pass){
        SQLiteDatabase sqLiteDatabase = this.getWritableDatabase(); //5(a)
        ContentValues cv = new ContentValues() ;// 5(b)
        cv.put("uname", name);
        cv.put("passw ",pass) ;
        long result = sqLiteDatabase.insert(tbName, null,cv); // 5(c)
        sqLiteDatabase.close(); // 5(d)
        return result ;
}
```

6. Implement the update function on the similar lines, the steps are,

   (a) Create a SQLiteDatabase and open it in Writeable mode using getWritableDatabase()

   (b) Use the execSQL to which pass the update SQL Statement

   (c) Close the database

```
public void update(String name, String pass){
        SQLiteDatabase sqLiteDatabase = this.getWritableDatabase(); //6(a)
        sqLiteDatabase.execSQL("UPDATE "+tbName+" SET passw='"+pass+"'"+" WHERE
            uname='"+name+"';");//6(b)
        // UPDATE tbName SET passw='pass' WHERE uname='name' ; SQL statement
        sqLiteDatabase.close(); // 6(c)
    }
```

7. Implement the delete function that takes the name as the input argument and follow these steps,

   (a) Create a SQLiteDatabase and open it in Writeable mode using getWritableDatabase()

   (b) Use the execSQL to which pass the DELETE SQL Statement using name as matching argument

   (c) Close the database

```
public void delete(String name){
        SQLiteDatabase sqLiteDatabase = this.getWritableDatabase(); // 7(a)
        sqLiteDatabase.execSQL("DELETE FROM "+tbName+" WHERE uname='" +name+"';" );
            // 7(b)
        // SQL statement : DELETE FROM tbName WHERE uname='name';
```

```
        sqLiteDatabase.close(); // 7(c)
    }
```

8. Implement the Display function by following the steps below,

   (a) Create a SQLiteDatabase and open it in Readable mode using getReadableDatabase()

   (b) Execute the SELECT statement using SQLiteDatabase.rawQuery() function and store the resultset into Cursor Object

   (c) iterate through the Cursor Object using Cursor.moveToNext() to parse the resultset

   (d) use CursorObject.getString(int Columnnumber) to retrieve the relevant fields from the table for all rows. Columns can be referred by using index that starts from 0.

   The steps are shown below,

```
public String display(Context ctx){
        SQLiteDatabase sqLiteDatabase = this.getReadableDatabase(); //8(a)
        Cursor cursor = sqLiteDatabase.rawQuery("SELECT * FROM "+tbName, null); //8(b)
        String finalres = " " ;
        while(cursor.moveToNext()){ //8(c)
            finalres += cursor.getString(0)+":"+cursor.getString(1); // 8(d)
        }
        return finalres;
}
```

9. Design the Landing Page by tweaking the activity_main.xml which is shown in figure 10.1

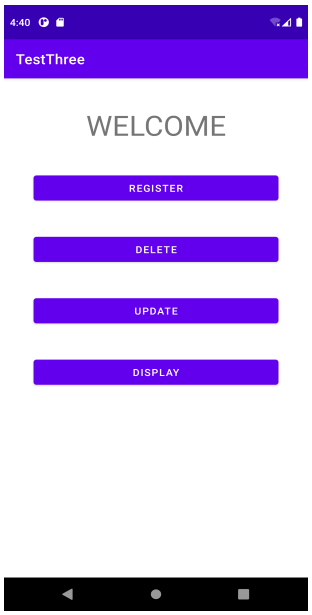
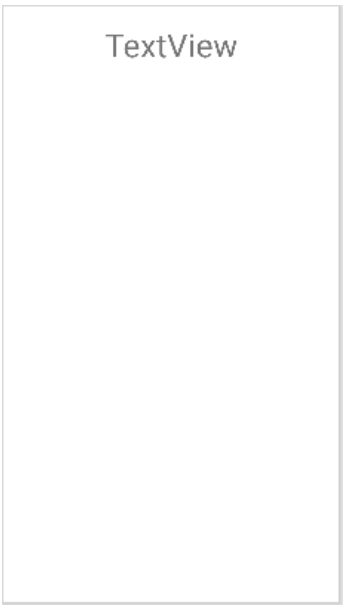


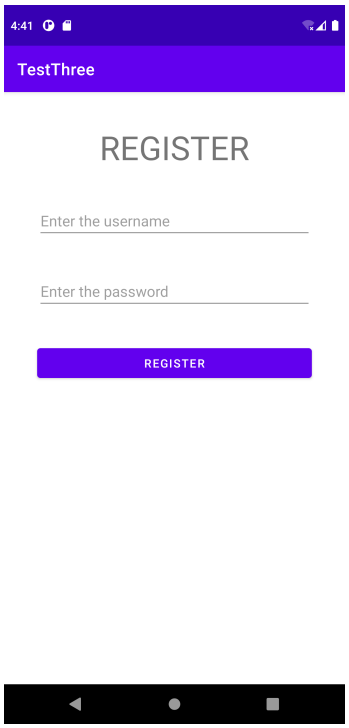
Figure 10.1: Landing Page

10. Design the pages for activity_display_page.xml, activity_registration.xml as shown in the figure 10.2a, 10.2b respectively.

11. Design the pages for activity_deletepage.xml & activity_updatepage.xml as shown in the figure 10.3a & 10.3b respectively.

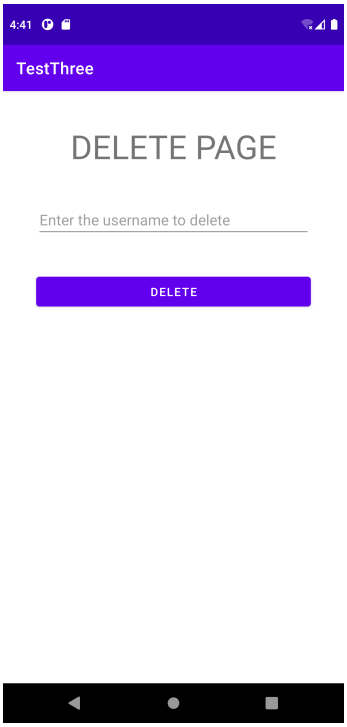


(a) Display Page

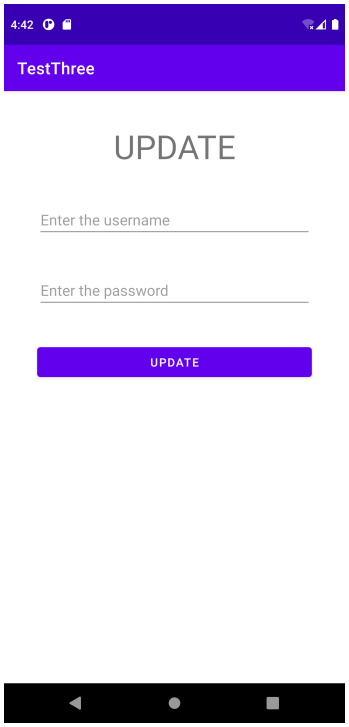


(b) Registration Page

Figure 10.2: Display & Registration Page Design



(a) Delete Page



(b) Update Page

Figure 10.3: Delete & Update Page Design

12. Modify the MainActivity.java and instantiate the respective buttons. Add onClickListerner to the buttons and navigate from MainActivity to the respective action activities(DeletePage, Registration, UpdatePage, DisplayPage) using Anonymous onClick definition respectively. Sample Code for navigation to Registration Page is shown below,

```
Button register, update, delete, display;
 @Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    register = findViewById(R.id.register);
   // equate the other buttons for update, delete and display here
   // setting onClickListener Anonymously
   register.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
   // Using Intent to Navigate to Registration.java and starting it upon register
       button click
            Intent it = new Intent(MainActivity.this, Registration.class);
            startActivity(it);
    }
});
// Similarly define Anonymous OnClickListerner for update, delete and display
    buttons as well here
}
```

13. Modify the Registration.java page with the following steps,

    (a) Define the Equivalent JAVA objects in Registration.java. Copy the dbName and dbVersion parameters from DBHelper class. Define an object of DBHelper class.

    (b) Equate the JAVA objects with XML objects using findViewByID function

    (c) Add an anonymous Event Listener to register button and override onClick function.

    (d) Inside the onClick function create the DBHelper Object by calling the default constructor and passing the context, dbName, cursorfactory and dbversion as arguments

    (e) Pass on the name and password from EditText to the addUser function called by using DBHelper object which returns -1 on error.

    (f) if it returns other value than -1, then it indicates the operation is successful and use Intent to navigate back to the main activity.

    The code snippet for the above steps are shown below indicating the step number

```
EditText uname; // 13(a)
EditText upass ;
DBHelper dbHelper;
Button register;
private static final String dbName="studentdb";
private static final String tbName="student" ;
private static final int dbVersion = 1 ;

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

```
uname = findViewById(R.id.rname) ; //13(b)
upass = findViewById(R.id.rpass) ;

register = findViewById(R.id.r_register) ;
register.setOnClickListener(new View.OnClickListener() { // 13(c)
    @Override
    public void onClick(View v) {
        dbHelper = new DBHelper(Registration.this, dbName, null, dbVersion) ;
            //13(d)
        long val = dbHelper.adduser(uname.getText().toString(),
            upass.getText().toString()) ; // 13(e)
        if(val == -1)
             Toast.makeText(Registration.this, "Error in adding user",
                 Toast.LENGTH_SHORT).show();
        else{
             Toast.makeText(Registration.this, "USER REGISTERED",
                 Toast.LENGTH_SHORT).show(); //13(f)
             Intent it = new Intent(Registration.this, MainActivity.class);
             startActivity(it);
             }
         }
     });
  }
```

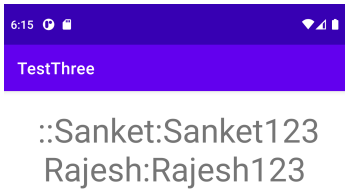
14. On the similar grounds, Implement the UpdatePage.java,The code inside the onClick function is as shown below,

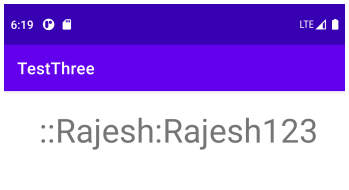
```
update.setOnClickListener(new View.OnClickListener() {
           @Override
           public void onClick(View v) {
               dbHelper = new DBHelper(Updatepage.this, dbName, null, dbVersion) ;
               dbHelper.update(uname.getText().toString(),
                   upass.getText().toString());
               Intent it = new Intent(Updatepage.this, MainActivity.class);
               startActivity(it);
           }
});
```

15. The delete function defined in DBHelper class takes in name as input and we have to pass that as an argument for the delete function.Modify tehe DeletePage.java. Follow the template for Registration page and the onClick function code for the Delete button is as shown below,

```
delete.setOnClickListener(new View.OnClickListener() {
           @Override
           public void onClick(View v) {
               dbHelper = new DBHelper(DeletePage.this, dbName,null, dbVersion) ;
               dbHelper.delete(duname.getText().toString());
               Intent it = new Intent(DeletePage.this, MainActivity.class);
               startActivity(it);
           }
});
```

16. Finally call the DBHelper.display() to retreive the username and update that in the TextView





(a) After Inserting Two Records

(b) After Deleting one Record

Figure 10.4: Results

by modifying DisplayPage.java class. The code snippet demonstrate the same.

```
TextView text ;
DBHelper dbHelper;
private static final String dbName="studentdb";
private static final int dbVersion = 1 ;
@Override
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_display_page);
        text = findViewById(R.id.textView) ;
        dbHelper = new DBHelper(DisplayPage.this, dbName, null, dbVersion);
        String out= dbHelper.display(DisplayPage.this);
        text.setText(out);
    }
```

10.4 Results

The Result obtained after inserting couple of users(Sanket & Rajesh) into the SQLiteDatabase is depicted in output snapshot shown in figure 10.4a, the second snapshot shows after deleting the first user Sanket how database looks like in figure 10.4b.

PART-B: FLUTTER APP DEVELOPMENT

Chapter 11

Flutter

Flutter is a powerful language packed with a powerful mobile framework that can be used in both iOS and Android applications. Flutter is often used with DART, which is an object-oriented programming language by Google[1]. Flutter was introduced in the year 2017 in the dart summit targetting app development for the lower or mid-range devices. Flutter is an open source framework by Google for building beautiful, natively compiled, multi-platform applications from a single codebase[2]. Some of the features of flutter are,

1. Flutter code compiles to ARM or Intel machine code as well as JavaScript, for **fast performance** on any device
2. Deploy to multiple devices from a **single codebase**: mobile, web, desktop, and embedded devices.
3. Flutter is supported and used **by Google**, trusted by well-known brands around the world, and maintained by a community of global developers.
4. Flutter uses its **own SDK and set of widgets** and is not dependent on the native components
5. Flutter supports hot reload function, typical of web
6. Flutter is an UI-Toolkit/Library which are used with the help of dart programming language

Some of the advantages of using flutter are,

- Apps can *run on both iOS and Android platform*, no additional configuration is needed
- The *development and NRE cost is very less* for product development
- Apps can be built with *less time* since dart has less learning curve
- *Code written* are comparatively less compared to Native counterpart

The downside of using flutter are,

- Lack of native feel
- Bit sluggish on older devices and the performance depends upon the CPU used
- Having its own components has its downside, if needed a new feature one has to develop it on its own or wait till the developers make it available as add-on packages
- Large APK size compared to Java and Kotlin code

Some of the apps build by flutter are, Google Ads, Google Assistant, Baidu, My Leaf, TopLine

[1] https://www.geeksforgeeks.org/what-is-flutter/
[2] https://flutter.dev/

11.1 Environment Setup

The flutter can be setup in windows, macOS, Linux and Chrome OS as well. Read the documentation for the required installation steps[3],

- Windows: https://docs.flutter.dev/get-started/install/windows
- Linux: https://docs.flutter.dev/get-started/install/linux
- MacOS: https://docs.flutter.dev/get-started/install/macos
- ChromeOS: https://docs.flutter.dev/get-started/install/chromeos

11.2 Setting up an IDE

To code applications using Flutter, make sure the relevant steps are followed to install the packages in section 11.1. The next step is to use an IDE for coding. The possible choice of IDEs available are,

1. Android Studio with Flutter plugin
2. IntelliJ IDE with flutter plugin
3. VSCode with flutter add-on

11.3 Flutter Architecture

The following figure 11.1[4] shows the architecture of the flutter and the components of Flutter SDK as well. Flutter comes with full fledged SDK for building apps. That includes tools to create UI,

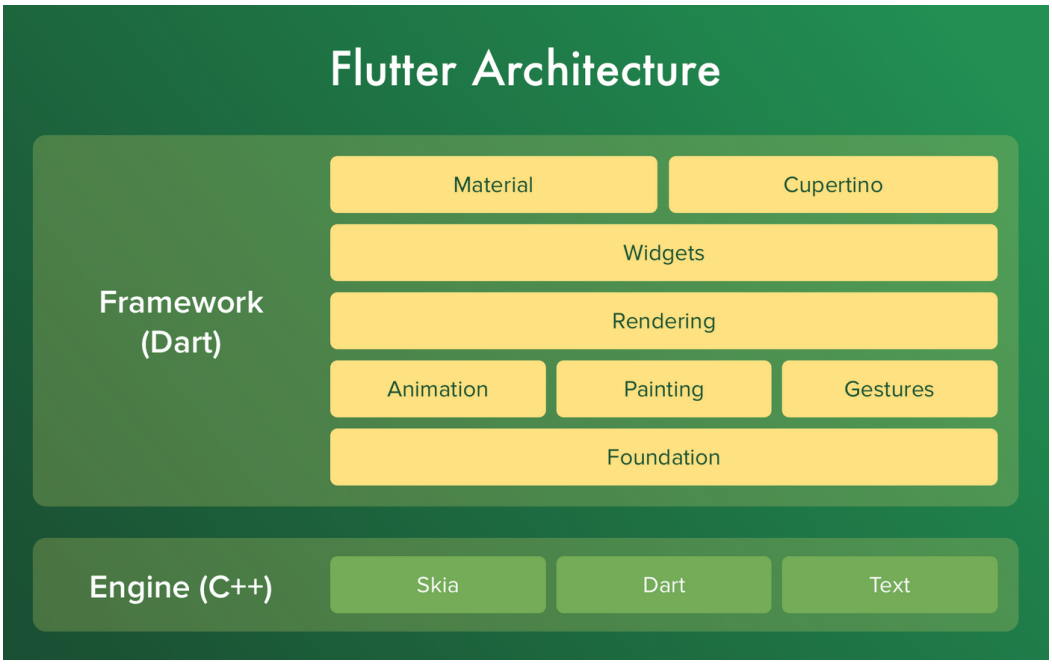


Figure 11.1: Flutter Architecture

widgets for both Android devices as well as iOS devices. The android apps uses *Material Design* based themes and Widgets, where as iOS apps uses *Cupertino* based themes and widgets. Flutter

[3] https://docs.flutter.dev/get-started/install
[4] https://www.cleveroad.com/blog/why-use-flutter

uses Dart for both server side and client side development and its completely open-source and object-oriented programming language. With the flexibility of dart language combined with C++ flutter core makes the app behave very close to the native ones in terms of performance.Flutter does not use native components in any form, so developers don't have to make bridges for communicating with them.Flutter draws the interface on its own. Buttons, text, media elements, background – all this is drawn inside the Skia graphics engine in Flutter[5].

11.4 Exercise

Create a sample Flutter application with a simple text at the center of the screen and apply styling to it.
The steps followed in Android Studio IDE are as follows,

1. Create a new flutter project and let the gradle build gets completed
2. Run the Emulator in the backend using AVD manager
3. Under the Project folder->lib->open the main.dart file which consist of the relevant template code which is by default a counter app which shows how many times the "+" button is pressed(Press the play button to run the app)
4. Delete everything in the main.dart file except the following lines,

```
import 'package:flutter/material.dart';

void main() {
  runApp(const MyApp());
}
```

5. Change the app name inside the runApp as Materialapp as shown below ,

```
import 'package:flutter/material.dart';
void main() {
  runApp(MaterialApp());
}
```

6. Navigate to test folder under project directory, open the widget_test.dart and change the name of the app as shown below,

```
// Build our app and trigger a frame.
await tester.pumpWidget(const MyApp()); //by default

// change this line to
await tester.pumpWidget(MaterialApp()); // pointing to the app created
```

7. Draw and visualize a widget tree consisting of components such as,
   (a) A Homepage widget which houses appBar & body Widget
   (b) Scaffold widget acts as a container that houses appBar, body
   (c) appBar widget houses title widget

[5] https://www.cleveroad.com/blog/why-use-flutter

(d) body widget houses a text widget

Everything is treated as widget in flutter. The overall widgets forms the app UI and are visualized using Widget Tree prior implementation as shown in figure 16.1

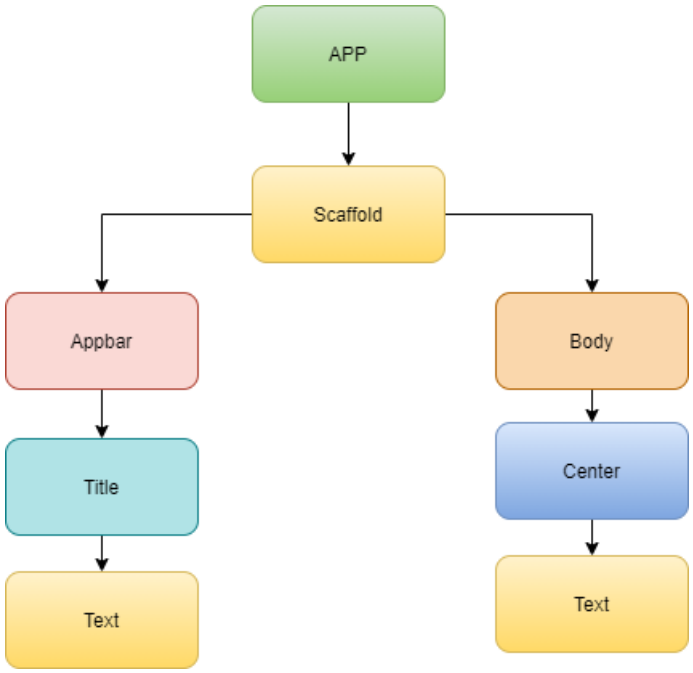


Figure 11.2: Widget Tree

8. Seeing the Widget Tree customize the MaterialApp with the following code snippet

```
import 'package:flutter/material.dart';

void main() {
  runApp(
    MaterialApp(
      home:Scaffold(
        appBar: AppBar(
          title: Text("First Flutter App"),
        ),
        body: Center(
          child: Text(
            'This is a first App',
            style: TextStyle(
              fontSize: 40.0,
              fontStyle: FontStyle.italic,
              fontWeight: FontWeight.bold,
            ),
          ),
        ),
      ),
    ),
  );
}
```

9. Run the App and test out the design

11.5 Results

The output of the app is as shown below figure 11.3

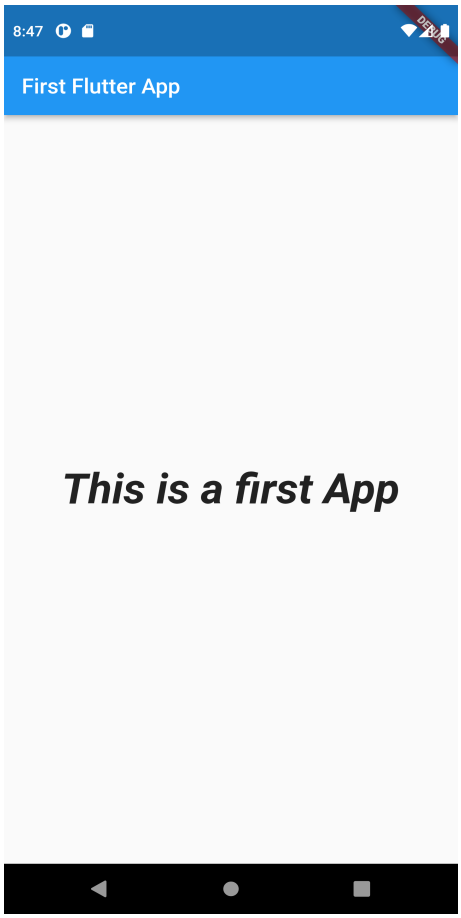


Figure 11.3: Output of the App

Chapter 12

Scaffold & SafeArea

The Scaffold widget is the base of the screen for a single page. It is used to implement the basic functional layout structure of an app. You can easily implement functional widgets like AppBar, FloatingActionButton, ButtonNavigationBar, Drawer, and many more widgets on the app using the Scaffold widget[1]. Some of the properties supported by the scaffold widgets are,

1. appBar: It is a horizontal bar displayed at the top of the screen. The app bar is one of the main components in your app, without it, your app may seem incomplete. The appBar widget has its own properties like elevation, title, actions, etc.

```
Scaffold(
appBar: AppBar(
title:Text("AppBar"), //title aof appbar
backgroundColor: Colors.redAccent, //background color of appbar
),
)
```

2. backgroundColor: This property on Scaffold is used to change the background color of the Scaffold screen.

```
Scaffold(
backgroundColor: Colors.blue, //set background color of scaffold to blue
)
```

3. body: This is the main content property on Scaffold. You have to pass the widget and it will be displayed on the screen.

```
Scaffold(
body: Center( //content body on scaffold
child: Text("Scaffold Widget")
),)
```

4. floatingActionButton: It is a floating button that is used for quick action.

```
Scaffold(
floatingActionButton:FloatingActionButton( //Floating action button on Scaffold
onPressed: (){
  //code to execute on button press
},
```

[1] https://www.fluttercampus.com/tutorial/9/flutter-scaffold/

```
child: Icon(Icons.send), //icon inside button
),)
```

SafeArea is basically a glorified Padding widget. If you wrap another widget with SafeArea, it adds any necessary padding needed to keep your widget from being blocked by the system status bar, notches, holes, rounded corners and other "creative" features by manufactures. Setting them all to false would be the same as not using SafeArea. The default for all sides is true.One can define SafeArea to any child and the properties that can be set is as shown below,

```
SafeArea(
left: false,
top: false,
right: false,
bottom: false,
child: Text('My Widget: ...'),
)
```

12.1 Exercise-I

Create a Flutter App to demonstrate the usage of SafeArea.
To build an app to demonstrate the usage of SafeArea, First we build an app without using SafeArea and then we build another using SafeArea. The following are the steps,

- Create a new flutter app, modify the main.dart file according to the widget tree shown in 12.1(a).

```
import 'package:flutter/material.dart';
void main() {
   runApp(MaterialApp(
   home: Text(
   'Without SafeArea', // Text to display
   textAlign: TextAlign.center, //align the text to center
   style: TextStyle(
   fontSize: 40, // fontSize in px
   fontWeight: FontWeight.w800, // font weight
   color: Colors.deepOrangeAccent, // Setting up the color font
   ),
   ),
   ));
}
```

- To see the usage of SafeArea, wrap the Text Widget around SafeArea Widget according to the figure 12.1(b) and make the Text Widget as child of SafeArea as shown below

```
import 'package:flutter/material.dart';
void main() {
   runApp(MaterialApp(
   home: SafeArea(
   child: Text(
   'Without SafeArea',
   textAlign: TextAlign.center,
   style: TextStyle(
```

```
        fontSize: 40,
        fontWeight: FontWeight.w800,
        color: Colors.deepOrangeAccent,
        ),
        ),
        ),
        ));
    }
```

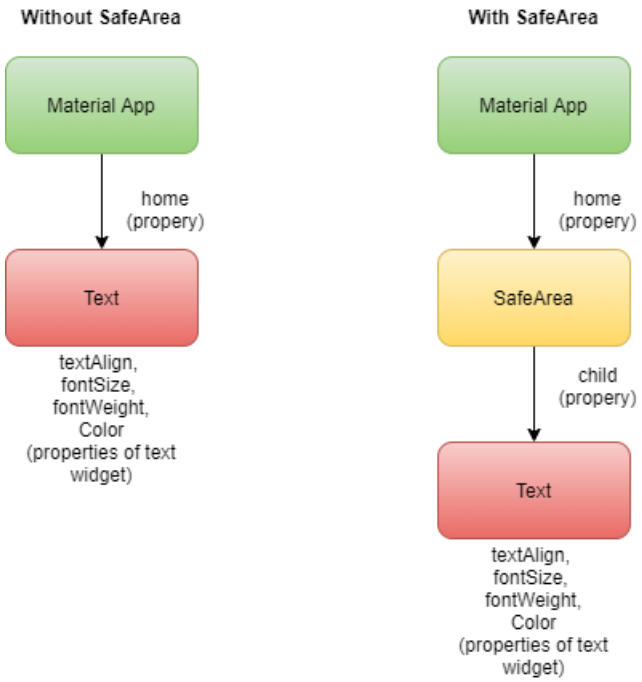


Figure 12.1: Widget Tree:SafeArea

12.2 Exercise-II

Create a Flutter App to demonstrate the usage of Scaffold.
To implement the app, Follow the widget tree as shown 12.2. The Root element of the app is a MaterialApp itself whose child points to Scaffold which acts as a container for the widgets such as AppBar, Text widgets. The code following the Widget Tree is as shown below,

```
import 'package:flutter/material.dart';

void main() {
   runApp(
   MaterialApp(
   home:Scaffold(
   appBar: AppBar(
   title: Text("First Flutter App"),
   ),
   body: Center(
   child: Text(
   'This is a first App',
   style: TextStyle(
   fontSize: 40.0,
   fontStyle: FontStyle.italic,
```

```
    fontWeight: FontWeight.bold,
    ),
    ),
    ),
    ),
    ),
    );
}
```

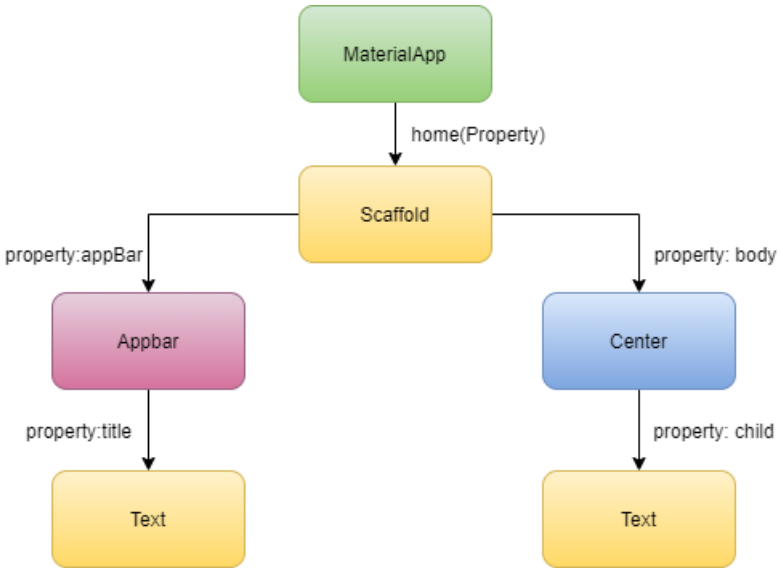
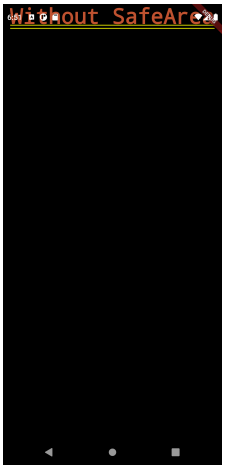


Figure 12.2: Widget Tree:Scaffold

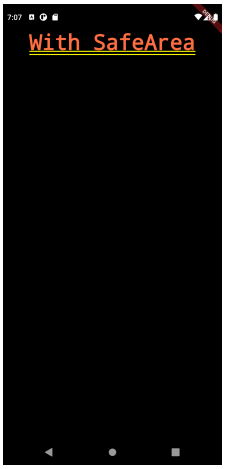
12.3 Additional Program:Combining Scaffold and SafeArea

The following program combines both the widgets in a common program,

```
void main() {
    runApp(MaterialApp(
    home: Scaffold( // Scaffold houses appBar and body property
    appBar:AppBar(
    title: Text(
    'Scaffold-SafeArea'
    ),
    ),
    body:Container(
    child: SafeArea( // Defining SafeArea as Child of Container
    child: Center(
    child: Text(
    'This is a Sample Text'
    ),
    ),
    ),
    ),
    ),
    ));
}
```

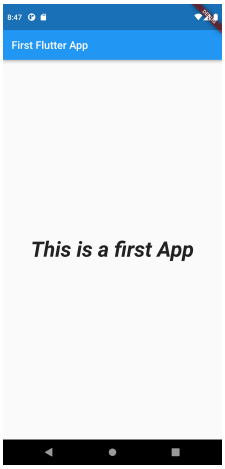


(a) Without the usage of SafeArea

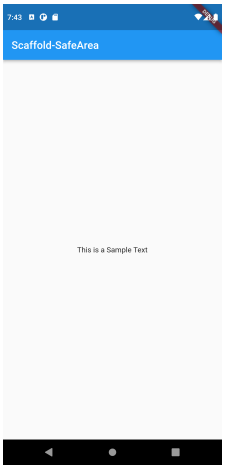


(b) Without the usage of SafeArea

Figure 12.3: Output of SafeArea



(a) Output of the Exercise-II



(b) Scaffold and SafeArea in action

Figure 12.4: Output of Exercise-II and Additional Program

12.4 Output

The output of the Exercise-I is as shown in the figure 12.3a & 12.3b , The output of the Exercise-II and Additional Program is as shown below figure 12.4a and 12.4b respectively,

Chapter 13

Layout Widgets

While designing apps using flutter, many inbuilt widgets from Flutter SDKs are used. Based on the number of childs a widget can possess, they are broadly classified as,

1. *Single Child Widgets* - These widgets have a single child property often seen as "child: ...".
2. *Multi Child Widgets*- These widgets can have more then one child, children in particular. often seen as "children[]" property

Some of the examples of the Single child widgets are container, padding, Center, Align, SizedBox Widget, etc. While for the Multi child widgets the most commonly used widgets are row and column widgets which can hold multiple widgets in bunch of rows and columns. The section aims at exploring few commonly used Layout Widgets, i.e, Single and Multi child Widgets.

13.1 Single-Child Widgets

Few commonly used Single-Child Widgets are discussed in this section. The focus is on Container, padding and Center Widgets.These widgets can house only one child widget.

1. *Container Widget*: Container Widgets houses one Widget, it provides two attributes for customization, they are sizeheight and width, painting & positioning[1]. The following code snippet shows the same,

```
body: Container(
    height: 200.0, // 200 logical pixel - depends on Device
    width: double.infinity,//expand to the width of screen
    color: Colors.blue, // Color attribute for the container
    child: Center( // Container ->child(Center->child(Text))
         child: Text(
         'Hello World',
         style: TextStyle(color: Colors.white, fontSize: 30),
            ),
        ),
    ),
```

2. *Center Widget*: Center Widget positions its child widget at the center of the screen.

```
body: Center(
        child: Text(
        'Hello World',
```

[1] https://chamithchathuka.medium.com/flutter-layout-widgets-b137758d9d43

```
        style: TextStyle(color: Colors.white, fontSize: 30),
        ),
)
```

3. *Align Widget*: Align Widgets places its child w.r.t to its parent using alignment property whose values can be topCenter, topRight, topLeft, center, bottomCenter, bottomRight, bottomLeft using the Align Object. The following code snippet has the parent Container which houses a child Aligment. The Align acts as a parent for another container.

```
body: Container(
    height: 200.0,
    width: double.infinity,
    color: Colors.blue, // Color attribute for the outer container
    child: Align(
    alignment: Alignment.bottomCenter, //places the child of align to bottomCenter
        of the outer container
    //change the topCenter, topRight, topLeft, center, bottomCenter, bottomRight,
        bottomLeft and observe the change
    child:Container( // inner container
    height: 20.0,
    width: 20.0,
    color: Colors.white, // Color attribute for the inner container
    ),
  ),
),
```

4. *Padding Widget*: This widget helps to add padding to its child widget. It provides a property called as padding which is set using an Object of *EdgeInset* Class. Some of the the options for padding using EdgeInset class are,

   (a) *EdgeInset.all(double value)*: Sets the padding from all direction using the value in terms of pixels

   (b) *EdgeInset.fromLTRB(double left, double top, double right, double bottom)*: sets the padding by adjusting the arguments of left, top, right and bottom

   (c) *EdgeInset.symmetric(double vertical=0.0, double horizontal=0.0)*: Sets the horizontal padding(left, right) and vertical padding(top,bottom)

   (d) *EdgeInset.only(double left, double top, double right, double bottom)*: same as fromLTRB

   The following code snippet demonstrate the usage of Padding,

```
Padding(
    padding: const EdgeInsets.all(20.0), // puts 20 dp padding from all sides
    // try setting the EdgeInset to other functions as well
    child: Container(
        height: 200.0,
        width: double.infinity,
        color: Colors.blue,
        ),
)
```

13.2 Multi-child Widgets

Multi child widgets are the widgets which supports more than one child widgets. Some of the commonly used multi-child widgets are, ScrollView, Row & Column Widgets. This section presents the discussion on Row and Column Widgets in particular.

13.2.1 Row Widget

A row widget places his children horizontally in a form of row. The row widget supports children[child1_widget, child2_widget,...] property to let the user define childrens for the row. The horizontal axis of the row is called *MainAxis* and the perpendicular axis is called as *CrossAxis*. These properties can be set using MainAxisAlignment & CrossAxisAlignment property respectively. A row can have multiple row/column widgets as its childrens.

13.2.2 Column Widget

A column widget places his children vertically in a form of column. The column widget supports children[child1_widget, child2_widget,...] property to let the user define childrens for the column. The horizontal axis of the row is called *CrossAxis* and the perpendicular axis is called as *MainAxis* for the column. These properties can be set using MainAxisAlignment & CrossAxisAlignment property respectively.A column can have multiple row/column widgets as its childrens.

13.3 Exercise-I

Build a flutter app to demonstrate the usage of Row Widget and Experiment with its MainAxis and CrossAxis properties respectively.
The Steps followed are as follows,

1. Create a MaterialApp and add Custom RowFun() as Homepage of the flutter app as shown below.

```
void main() {
  runApp(MaterialApp(
    home: RowApp(),
  ));
}
```

2. Make the RowApp inherit the traits of Stateless widget to support hot-reload features.

```
class RowApp extends StatelessWidget { // Our RowApp page inherits the traits of the
    Stateless Widget
  const RowApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Container(); // here we build the Widget Tree
   }
}
```

3. Add the Row Widget to body of the App. Set the mainAxisAlignment as MainAxisAlignment.spaceEvenly(start,end,center, spaceAround, strech) and crossAxisAlignment as CrossAxisAlignment.start(end,stretch,baseline).

```
body:Row(
        mainAxisAlignment: MainAxisAlignment.spaceEvenly,
        crossAxisAlignment: CrossAxisAlignment.start,
        children:[ ],
)
```

4. Make three containers as a children of Row, and specify the children inside children:[ ]. This is demonstrated as shown in the below code snippet.

```
Container(
     height: 100,
     width: 100,
     color: Colors.blue, // change the color to red and green for other two
         containers
     child: Center(
     child: Text(
            'First Child'
      ),
   ),
),
// code for container 2
// code for container 3
```

13.4 Exercise-II

Build a flutter app to demonstrate the usage of Column Widget and Experiment with its MainAxis and CrossAxis properties respectively.
The Steps followed are as follows,

1. Create a MaterialApp and add Custom ColumnApp() as Homepage of the flutter app as shown below.

```
void main() {
  runApp(MaterialApp(
    home: ColumnApp(),
  ));
}
```

2. Make the ColumnApp inherit the traits of Stateless widget to support hot-reload features.

```
class ColumnApp extends StatelessWidget { // Our ColumnApp page inherits the traits
    of the Stateless Widget
  const ColumnApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Container(); // here we build the Widget Tree
   }
}
```

3. Add the Column Widget to body of the App. Set the mainAxisAlignment as MainAxisAlignment.spaceEvenly(start,end,center, spaceAround, strech) and crossAxisAlignment as CrossAxisAlignment.start(end,stretch,baseline).

```
body:Column(
        mainAxisAlignment: MainAxisAlignment.spaceEvenly,
        crossAxisAlignment: CrossAxisAlignment.start,
        children:[ ],
)
```

4. Make three containers as a children of Column, and specify the children inside children:[ ]. This is demonstrated as shown in the below code snippet.

```
Container(
     height: 100,
     width: 100,
     color: Colors.blue, // change the color to red and green for other two
         containers
     child: Center(
     child: Text(
           'First Child'
      ),
   ),
),
// code for container 2
// code for container 3
```

13.5 Exercise-III

Build a flutter app to demonstrate the usage of Row & Column Widget and Experiment with its MainAxis property respectively for the design shown in figure 13.1.

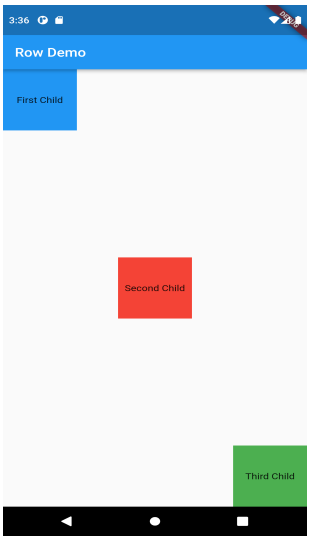
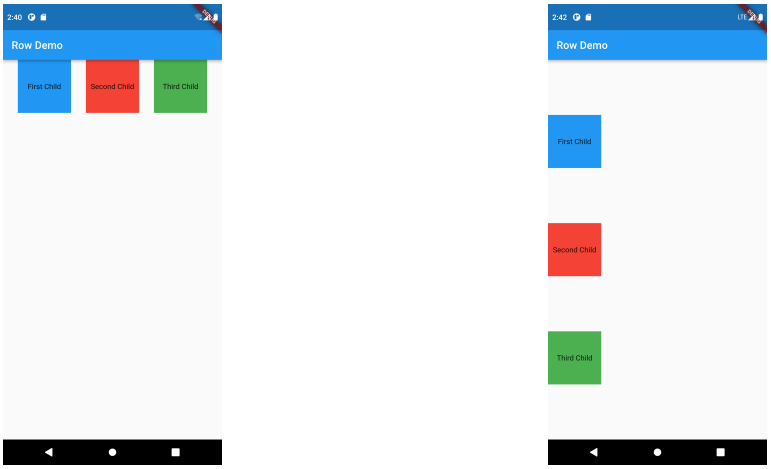


Figure 13.1: TO-DO Exercise



(a) Row Widget Demo (b) Column Widget Demo

Figure 13.2: Row & Column Widget Demo

13.6 Results

The results of Row and Column Widgets are shown in below. The figures 13.2a shows the output of Row Widget and 13.2b shows the Column Widget.

Chapter 14

Stateful Widgets v/s Stateless Widgets

The state of an app can be defined as anything that exists in the memory of the app while the app is running. This includes the properties of the object, the values each property holds.State is simply the information of a StatefulWidget. Every StatefulWidget has a State Object. This State Object keeps a track of the variables and functions that we define inside a StatefulWidget[1]. Based on the state property,the Widgets can also be classified into two types they are,

1. *Stateless Widgets*: The Widgets whose state can not be altered once they are built are called stateless widgets. These Widgets are immutable once they are built i.e, any amount off change in the variables, icon, buttons, or retrieving data can not change the state of the app

2. *Stateful Widgets*: The Widgets whose state can be altered once they are built are called stateful widgets. These states are mutable and can be changes multiple times in their app lifetime.

When an app is implemented as either stateful or stateless, then only user can use the *Hot-Reload* feature of the flutter. During Hot-reload the Dart compiler instead of recompiling the full app, looks for the code which has been changed from the previous state. This code is called as dirty code/changed code. The Hot-reload will only recompile the dirty code while running the app. This saves time and app will be loaded fastly.

The stateful widgets are immutable while building it. But can change the state over time. In order to update the UI change in a stateful widget, we use *setState*() function. The code inside the setState function is marked as dirty and will be updated on save. The setState function takes a function as an argument, but it can be an empty/void function too. The syntax of setState is as shown below,

```
void setState(VoidCallback function) {
  // code marked as dirty which will be redrawn
}
// typical syntax
setState((){
   // Dirty code
   }
);
```

14.1 Exercise

Implement a hybrid Dice rolling app to demonstrate the use of setState() method for marking part of the code as dirty, and refereshing the app must result in update of UI.

[1] https://dev.to/nicks101/when-to-use-setstate-in-flutter-380

The following are the steps to be followed,

1. Create a new flutter app

2. Download the Dicee App Stub file from this link https://github.com/londonappbrewery/dicee-flutter. Copy the images folder from the Stub project and place them in to your project directory created by you . The snapshot upon copy-pasting the images folder into your project should look like this figure 14.1.

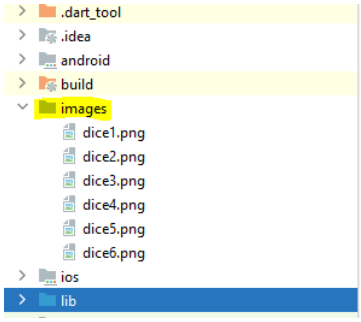


Figure 14.1: After copy-pasting the image folder into new Project

3. Open the pubspec.yaml file to reference the images we copied. Append the images folder under asset property as shown below . Click on pub-get to index the new resources. Return back to main.dart file and click on get-dependencies pop up to be in sync with the updated yaml file.

```
// pubspec.yaml file
flutter:
  uses-material-design: true
  assets:           // add these two line, careful about the spacing.
    - images/
```

4. Build a simple Material App with appBar and make the body points to a stateful Widget called DicePage() as shown below,

```
void main() {
  runApp(MaterialApp(
    home:Scaffold(
      appBar: AppBar(title: Text('DICEE'),centerTitle: true,), // Simple Appbar
      body: DicePage(), // body now points to a DicePage which is a stateful widget
    ),
  ));
}
class DicePage extends StatefulWidget {
  const DicePage({Key? key}) : super(key: key);

  @override
  _DicePageState createState() => _DicePageState();
}

class _DicePageState extends State<DicePage> {
    @override
    Widget build(BuildContext context) {
    return Container(); // Here we build our body widget Tree
```

```
    }
}
```

5. Build the Widget Tree for body as shown below in figure 14.2. The widget tree comprises of following widgets,

   (a) *Expanded*: A widget that expands a child of a Row, Column, or Flex so that the child fills the available space. Using an Expanded widget makes a child of a Row, Column, or Flex expand to fill the available space along the main axis. If multiple children are expanded, the available space is divided among them according to the flex factor.[2]. The following code snippet demonstrates the same,

```
Expanded(
      flex: 2,
      child: Container(
      color: Colors.amber,
      height: 100,
      ),
),
```

   (b) *TextButton*:Text Button is a Material Design's button that comes without border or elevation change by default. Therefore, it relies on the position relative to other widgets[3]. A simple definition of the TextButton is shown below,

```
TextButton(
    child: Text('Simple text Button'), // can be Text/Image button too
   style: TextButton.styleFrom(
           primary: Colors.black, // text color
           backgroundColor: Colors.blueAccent, // background color
    ),
    onPressed: () {
      print('Pressed');
    }
)
```

6. The Code after implementing the left half of the Widget tree is shown below,

```
@override
  Widget build(BuildContext context) {
    return Center(
      child: Container(
        child: Row(
          children: [
            Expanded(
              flex: 1,
              child: TextButton(
                style: TextButton.styleFrom(
                  backgroundColor: Colors.blueAccent,
                ),
              child:Image.asset('images/dice1.png'), // refer the first image of
                  dicee.
```

[2] https://api.flutter.dev/flutter/widgets/Expanded-class.html
[3] https://www.woolha.com/tutorials/flutter-using-textbutton-widget-examples

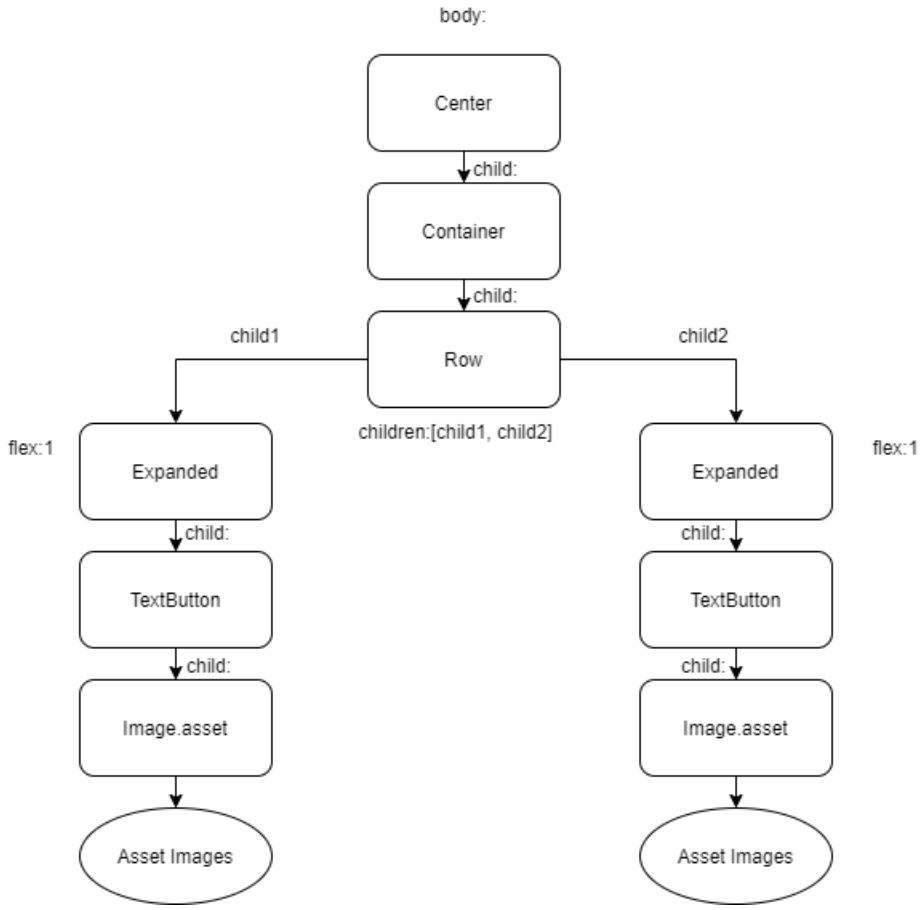


Figure 14.2: Widget Tree for body of the App

```
                onPressed: (){
                  //function to change the face
                },
              ),
              ),
              // Implement another child of the row with image asset dice2.png
              ],
          ),
        ),
      );
    }
}
```

7. Define a function called as changeface() which will use set state to change the UI and dice image number dynamically as shown below,

```
class _DicePageState extends State<DicePage> {
  int left = 1 ; // set the first image number to left
  int right = 2 ; // set the second image number to right

  void changeface(){
    setState(() { // marks the code below as dirty
      left = Random().nextInt(6)+1; // Random() belongs to Math library
      right = Random().nextInt(6)+1 ; // Random().nextInt(6) - generates random
          number between 1-6, excluding 6.
      // to include 6 as well, add 1 to it.
```

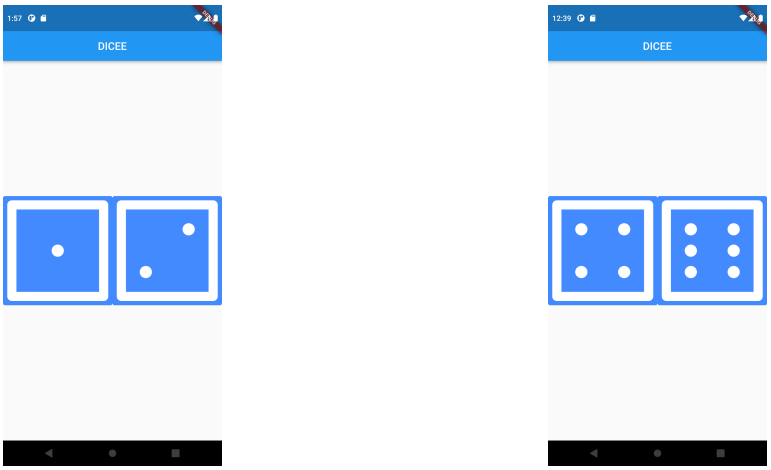
```
  });
}
@override
Widget build(BuildContext context) {
..
}
}
```

8. Modify the onPressed() function in the Two text Button as follows,

```
Expanded(
          flex: 1,
          child: TextButton(
            style: TextButton.styleFrom(
              backgroundColor: Colors.blueAccent,
            ),
          child:Image.asset('images/dice$left.png'), // dynamically changes the
              image number
            onPressed: (){
              changeface(); // calls the UI change on button pressed
            },
        ),
        ),
  Expanded(
          flex: 1,
          child: TextButton(
            style: TextButton.styleFrom(
              backgroundColor: Colors.blueAccent,
            ),
            child:Image.asset('images/dice$right.png'), // dynamically changes
                the image number
            onPressed: (){
              changeface(); // calls the UI change on button pressed
            },
          ),
        ),
```

14.2 Results

The output of the App is shown in figure 14.3a & 14.3b.



(a) Left Button displaying dice1.png, Right Button displaying dice2.png

(b) Upon Clicking left/right button: UI updated with random dicee image(1-6)

Figure 14.3: Output

Chapter 15

XyloPhone App

The Xylophone is a simple app that demonstrates how to play the audio files that is present in the flutter asset folder. To play audio files, we need to make use of third party library. A list of proven good third party packages can be found in the link https://pub.dev/. In order to play the audio files we use a third party library called as "audioplayers" which is available in https://pub.dev/packages/audioplayers. To use this package we need to add the dependencies in pubspec.yaml file as shown below, and click on pub get to sync the files into user's flutter project.

```
dependencies:
  audioplayers: ^0.19.1
```

A sample code snippet of its usage is shown below. The code snippet demonstrates how we can play the audio using audioplayer package. The steps are as follows,

```
void PlayAudio(int i){
    final player = AudioCache();
    player.play("note$i.wav");
  }
```

Add the import explicitly at the top of the program as shown below

```
import 'package:audioplayers/src/audio_cache.dart';
```

15.1 Exercise

Build a Flutter App called as Xylophone which imitates the behaviour of a physical Xylophone, where in it has bunch of buttons, with different colors stretched to the screen. Upon click of each button, different audio notes need to be played.
The following are the steps to be followed,

1. Open the browser and download the stub project from this URL https://github.com/londonappbrewery/xylophone-flutter.

2. Copy the asset folder which consists of 7 different audio files into the user's project folder. The project folder upon copying the asset folder into our project directory is as shown in figure 15.1,

3. Sync the assets folder and add the audioplayers dependencies into the pubspec.yaml file as shown below, and click on pub get and return to main.dart and click on get-dependencies to sync the project.

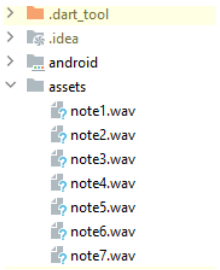


Figure 15.1: Project Structure

```
dependencies:
  flutter:
    sdk: flutter

  cupertino_icons: ^1.0.2
  audioplayers: ^0.19.1 # adding the 3rd party audioplayers package
dev_dependencies:
  flutter_test:
    sdk: flutter
  flutter_lints: ^1.0.0

flutter:
  uses-material-design: true
  assets:   # syncing asset folder
    - assets/
```

4. Build the Widget Tree as shown in figure 16.1.

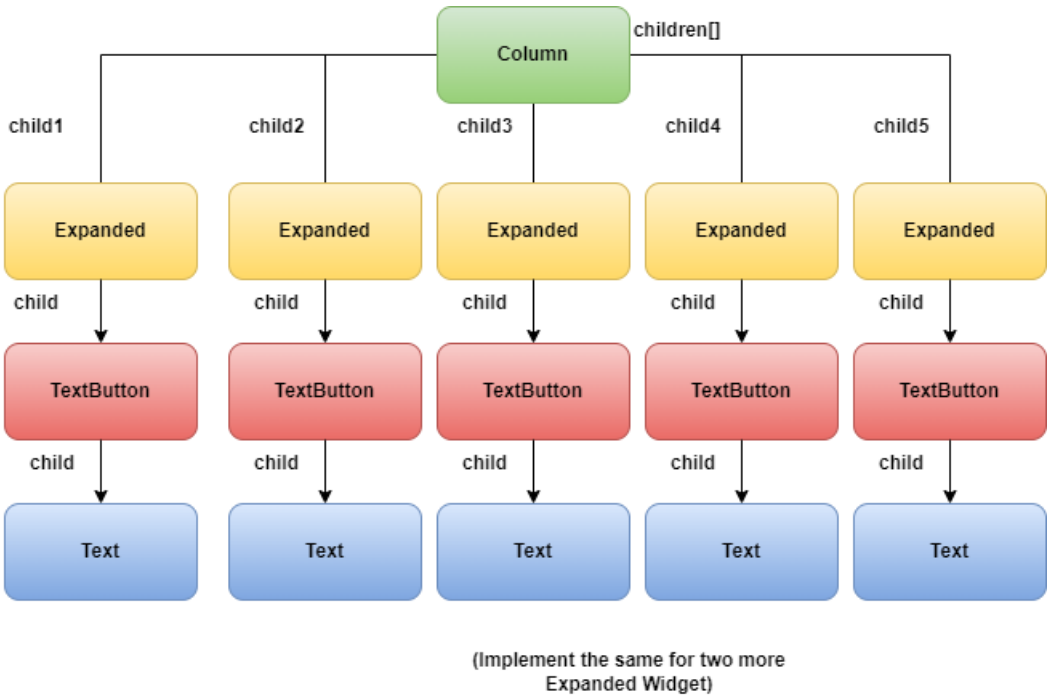


Figure 15.2: Xylophone: Widget Tree

5. The starter code would be a stateful widget and it is as shown below,

```
void main() {
  runApp(MaterialApp(
      home:Scaffold(
        appBar: AppBar(title: Text('XYLOPHONE'),centerTitle: true,),
        body:XyloPage(),
      )
  ));
}
class XyloPage extends StatefulWidget {
  const XyloPage({Key? key}) : super(key: key);

  @override
  _XyloPageState createState() => _XyloPageState();
}

class _XyloPageState extends State<XyloPage> {
 @override
  Widget build(BuildContext context) {
    return Container(); // Build the Widget Tree here
    }
}
```

6. Define the function PlayAudio(int i) where it takes a number and plays the respective audio file from the asset folder as shown below,

```
class _XyloPageState extends State<XyloPage> {
  void playSound(int noteNumber) {
  final player = AudioCache();
  player.play("note$noteNumber.wav");
}
```

7. Building the App using Widget Tree, the code snippet would look as shown below,

```
 @override
  Widget build(BuildContext context) {
    return Column(
      crossAxisAlignment: CrossAxisAlignment.stretch, // Setting the column to
          occupy full screen width horizontally
      children: [
        Expanded(child: TextButton(
          style: TextButton.styleFrom(
              backgroundColor: Colors.amber
          ),
          onPressed: (){
            PlayAudio(1);
          }, child: Text('First'),
        ),),
        Expanded(child: TextButton(
          style: TextButton.styleFrom(
              backgroundColor: Colors.cyanAccent
          ),
        onPressed: (){
            PlayAudio(2);
          }, child: Text('Second'),
```

```
      ),),
  // similarly implement the other expanded widgets 5 times and call
      PlayAudio(number)
```

15.2 Results

The output of the App is shown in figure 15.3

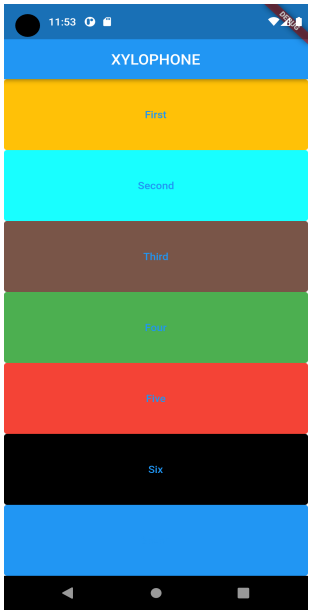


Figure 15.3: Result

Chapter 16

Quiz App

The quiz app makes use of the concepts of classes and objects in dart to build a yes/no type of quiz app, where in the user input is controlled by two buttons for every question. To build this we build a question class which provides a template for the objects and questions class which contains all the questionnaires in the form of a list. The app also makes use of the RaisedButton widget which can be enabled/disabled upon the user choice. The app also is a good illustration of the following,

1. Modularising the code into classes
2. Dart classes and objects
3. Using class constructors
4. private and public modifiers
5. Dart Lists
6. const vs final keywords

16.1 Exercise

Implement a quiz application using flutter that consist of two buttons true/false. Upon answering the question, the result of the answer should be shown and the user will navigate to the next question. Once the questions in the quiz ends, user should be shown the score.

The following are the steps,

1. Create a new class called question and define the member functions and the constructors as shown below,

```
class Question {
  final String questionText; // Question
  final bool answer; // Answer

  Question({required this.questionText, required this.answer});
  // making it as named arguments for the Question Constructor
}
```

2. Use the above class to create the questions and put it as List under the class "questions" as shown below,

```
import 'package:quizapp/question.dart'; // add the question class
class Questions {
  List<Question> questionBank = [ // create a list of Questions using default
      constructor of Question()
    Question(questionText: "Question1", answer: true), // set the Question and Answer
    Question(questionText: "Question2", answer: true),
    Question(questionText: "Question3", answer: false),
    Question(questionText: "Question4", answer: true),
    Question(questionText: "Question5", answer: false),
    Question(questionText: "Question6", answer: false),
    Question(questionText: "Question7", answer: false),
  ];
}
```

3. Build the widget tree as shown in figure 16.1.

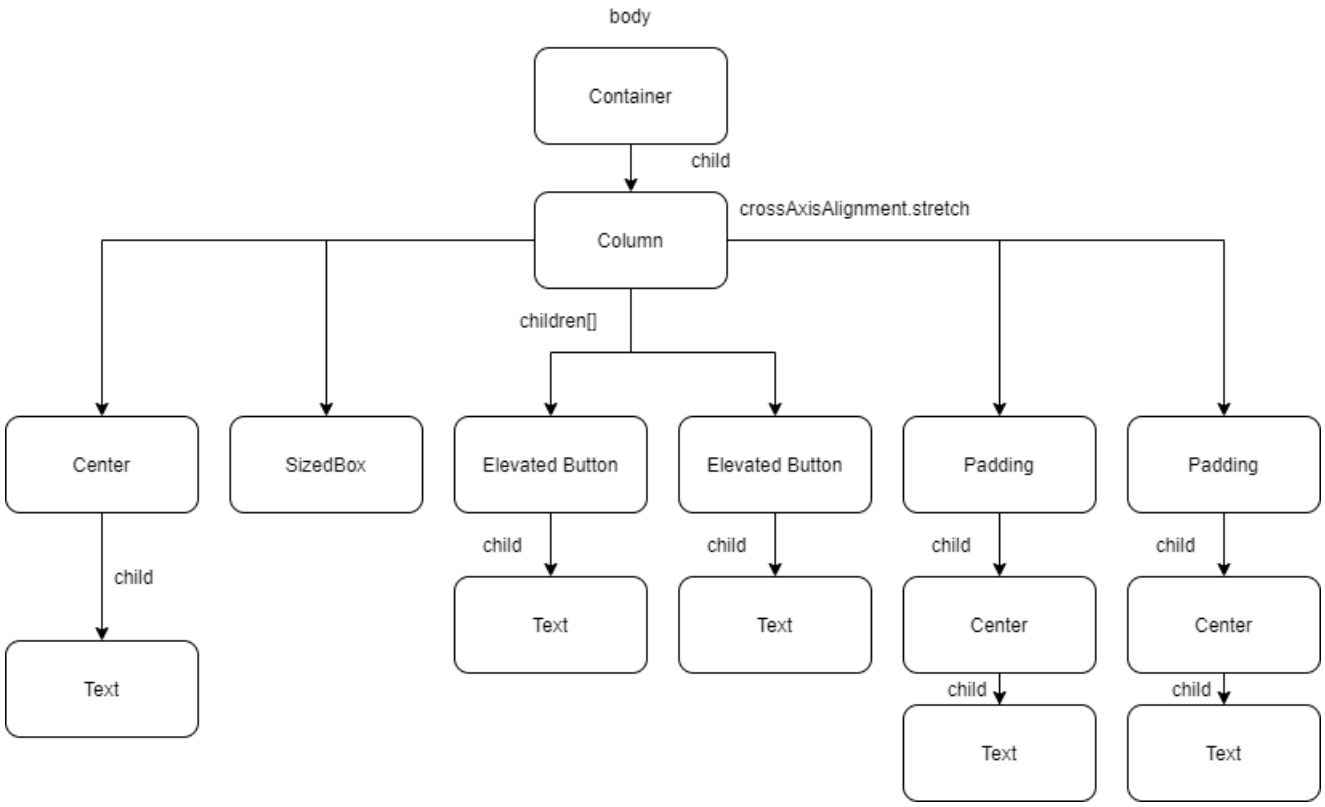


Figure 16.1: Widget Tree

4. Building the initial widget Tree Skeleton would look like below,

```
void main() {
  runApp(MaterialApp(
      home: SafeArea(
        child: Scaffold(
          body: QuizPage(),
        ),
      ),
  ));
}
```

```
class QuizPage extends StatefulWidget {
  const QuizPage({Key? key}) : super(key: key);

  @override
  _QuizPageState createState() => _QuizPageState();
}

class _QuizPageState extends State<QuizPage> {
  @override
  Widget build(BuildContext context) {
    return Container(); // Build the Widget Tree here
  }
```

5. Build the following variables which holds the questionNumber and currentScore value of the quiz as shown below,

```
class _QuizPageState extends State<QuizPage> {
  int questionNumber = 0; // stores the questionNumber, default = first
  int currentScore = 0; // final score initialized to 0
  Questions questions = Questions(); // Create an object of the Question Class
```

6. Build the following helper functions to give functionalities to the button and question change,

```
class _QuizPageState extends State<QuizPage> {
  int questionNumber = 0; // stores the questionNumber, default = first
  int currentScore = 0; // final score initialized to 0
  Questions questions = Questions(); // Create an object of the Question Class
void updateQuestionNumber() {
    setState(() {
      questionNumber = questionNumber + 1; // Increments the Question Number
      print('$questionNumber');
    });
  }

  void udpateCurrentScore(bool choice, int question_number) {
  // based on the choice (T/F button) score will be updated
    if (questions.questionBank.length == question_number) {
      print("End of questions");
    } else {
    // checks the current user input against the list answer, if true increments the
        count
      if (questions.questionBank[question_number].answer == choice) {
        setState(() {
          currentScore++;
        });
      }
    }
  }
// function to ensure that the question number to be not crossing the boundary of
    the list
  bool checkquestionNumber(int questionNumber) {
    return questionNumber < questions.questionBank.length ? true : false;
  }
  ...
```

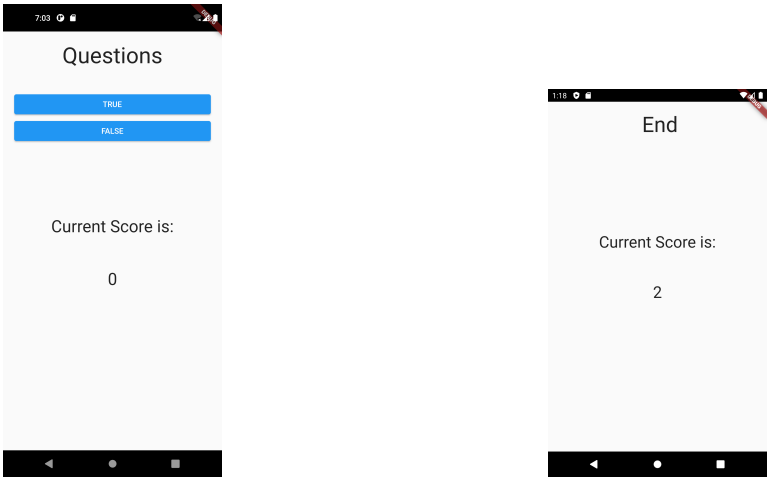
7. After building the widget tree, the column section would look like below,

```
children: [
 Center(
           child: Text("Questions", style: TextStyle(fontSize: 40.0),
           ),
         ),
         SizedBox(height: 20.0),
         ElevatedButton(
           onPressed: () {
             setState(() {
               print('True is pressed');
             });
           },
           child: Text('True'),
         ),
         SizedBox(width: 20.0),
         ElevatedButton(
           onPressed: () {
             setState(() {
               print('false is pressed');
             });
           },
           child: Text('False'),
         ),
         SizedBox(
           height: 100.0,
         ),
         SizedBox(
           height: 50.0,
         ),
         Padding(
           padding: const EdgeInsets.all(30.0),
           child: Center(
             child: Text(
               "Current Score is:",
               style: TextStyle(fontSize: 30),
             ),
           ),
         ),
         Padding(
           padding: const EdgeInsets.all(30.0),
           child: Center(
             child: Text(
               '${currentScore}',
               style: TextStyle(fontSize: 30),
             ),
           ),
         ),

]
```

8. Update the question Text in the first branch of the Widget Tree as follows,

```
Center(
```



(a) Landing Page (b) Showing Scores

Figure 16.2: Output

```
    child: Text(
    checkquestionNumber(questionNumber)?
        questions.questionBank[questionNumber].questionText.toString()
                        : " End",
    style: TextStyle(fontSize: 40.0),
    ),
),
```

9. Control the Elevated button access using the condition check, as shown below,

```
if (checkquestionNumber(questionNumber))
    ElevatedButton(
    onPressed: () {
        setState(() {
           if (questionNumber == questions.questionBank.length) { // check the bound
               of the list
              print("End of questions");
           } else {
           // check the user answer against the answer in the list
              udpateCurrentScore(true, questionNumber);
           // increment the Question Number
              updateQuestionNumber();
           }
    });
    },
    child: Text('True'),
),
// On the similar lines, implement the same for the false button
```

16.2 Results

The output of the App is shown in figure 16.2