INDUSTRIAL TRAINING SEMINAR

on

Kotlin

Submitted in the partial fulfilment for the award of Diploma in

Computer Science & Engineering

MEHR CHAND POLYTECHNIC COLLEGE

JALANDHAR

Submitted to

Mr. Navam Sir

Submitted by:

Sahil

551/22

ACKNOWLEDGEMENT

Through this acknowledgement, I express my sincere gratitude to all those people who are associated with this project and are helping me with it to make it a worthwhile experience. First and fore most I would like to thank almighty for giving me courage to make this project. At the outset, I would like to propose a word of thanks for the people who gave me unending support and help in numerous ways.

Firstly, I express our thanks to **Mr. Prince Sir** (HOD CSE DEPTT) who gave me this opportunity to learn the subject in a practical approach who guided me and gave me valuable suggestions regarding the project report.

Secondly, I would like to thanks **Mr. Navam Sir** (Incharge of Final year CSE deptt) who gave me this opportunity to learn the subject in a practical approach who guided me and gave me valuable suggestions regarding the project report

Thirdly, I would like to thank **Ms. Neha Basra Mam** (Project guide) and FACULTY MEMBERS of O7 SERVICES who are giving me their immense support in completing the project. The atmosphere provided is full of gaining more and more knowledge and to keep enthusiastic nature. The teachers provide a lot of help in resolving my doubts and making this project successful in shorter time.

I would like to thank my parents and friends who help me a lot in finishing the project in limited time.

Lastly, I would like to thank God. He has given me strength and encouragement throughout all the challenging moments of completing this dissertation. I am truly grateful for his unconditional and endless love, mercy, and grace.

I am making this project not only for marks but also to increase my knowledge. Thanks again to all who are helping me in completion of this project.

Ref. No. 07S/6W24/25.

07 SERVICES
A Step Towards Success
An IT Company With Learning Hub

Certificate

OF COMPLETION

THIS CERTIFICATE IS PROUDLY PRESENTED TO

SAHIL

FOR SUCCESSFULLY COMPLETING TRAINING COURSE IN

KOTLIN ANDROID

FROM JULY 2024 TO AUGUST 2024

ISO-9001:2015

Instructor's Signature

Director's Signature

INDEX

1. ABOUT THE COMPANY

About the Company

O7 Services is an **ISO 9001:2015** and **MSME PB10D0011152** Certified Company and is blessed with a Strong and Talented Team. Basically, they deal in Web Development, Mobile Development, Custom Software Development, Designing, Hosting services, Digital Marketing, Registration of Domain Names with the various latest extensions, Internet and Social Media Marketing, Search Engine Optimization (SEO), Pay Per Click (PPC) Management and provide the most advanced IT solutions, supporting full business cycle: preliminary consulting, system development & deployment, quality assurance and 24×7 supports. With a rich experience of over 6 successful years, O7 Services tends to build long-lasting strategic partnerships with their clients to ensure affordable prices, timely delivery and measurable business results. Their Head Office is in Jalandhar and the Branch Office is in Hoshiarpur.

Some of the products developed by O7 Services are- Tracking System, Invoice Software, School Management System, Hospital Management system, Parents- Teacher Communication App, Fee Management system, Task Management System, Online Food Ordering App, Security App, Admission system, Inventory Software, Car Servicing App etc.

Apart from this, O7 Services also provides 6 Weeks/ 6 Months Industrial Training, Project Based Training, Corporate Training and Job Oriented Courses Training to the students on all major IT Trends like-

Flutter, Kotlin, Android, IOS, FIREBASE, Python, Angular, React JS, Node JS, ASP.NET, .NET CORE, PHP, Laravel, CodeIgniter, OOPs PHP, MEAN Stack, MERN Stack, Digital Marketing, WordPress, Red Hat Linux, CCNA, CCNP, CCNA Security, MCSE, MCITP, Java, Java MVC, C/C++, Photoshop, Adobe Illustrator, CorelDraw etc

Mission-Vision-Goals

Vision:

"Vision is the Art of seeing what is Invisible to Others"

Worldwide reputation is the dream of every company and O7 Services want to achieve it through their work.

Mission:

"Your Dream is our Mission"

O7 Services deliver Innovative Web and Mobile Solutions, ensure complete transparency in the procedure, provide regular updates on the project and give equal priority to all the customers regardless what is the size or type of their project.\

Goals:

"A Dream becomes a Goal when Action is taken towards its Achievement"

Their Goals are SMART- Specific, Measurable, Attainable, Relevant and Timely goals. They tend to provide best Web Solutions, Desktop Applications & Mobile Applications, build the Brand of our Customers, build Trust and Long-Lasting Relationships, help our customers with Internet Marketing and be the Best in the field of Industrial Training.

2. BASICS OF KOTLIN

Kotlin is a modern, trending programming language that was released in 2016 by JetBrains. It has become very popular since it is compatible with Java (one of the most popular programming languages out there), which means that Java code (and libraries) can be used in Kotlin programs.

Kotlin is used for:

- Mobile applications (especially Android apps)
- Server-side applications
- Data science
- And much, much more!

Kotlin is a cross-platform, statically typed, general-purpose programming language with type inference. Kotlin is designed to interoperate fully with Java, and the JVM version of Kotlin's standard library depends on the Java Class Library, but type inference allows its syntax to be more concise. Kotlin mainly targets the JVM.

2.1 OOPS concepts

OOP stands for **Object-Oriented Programming**. Procedural programming is about writing procedures or methods that perform operations on the data, while object-oriented programming is about creating objects that contain both data and methods.

Object-oriented programming has several advantages over procedural programming:

- OOP is faster and easier to execute
- OOP provides a clear structure for the programs

Concept of OOPS:

1. **Class:** Classes in Kotlin are declared using the keyword class:

The class declaration consists of the class name, the class header (specifying its type parameters, the primary constructor, and some other things), and the class body surrounded by curly braces.

Both the header and the body are optional; if the class has no body, the curly braces can be omitted. For example, class Empty {….}

2. **Objects:** It is a basic unit of Object-Oriented Programming and represents real life entities. A typical Kotlin program creates many objects, which as you know, interact by invoking methods. An object consists of:

- **State**: It is represented by attributes of an object. It also reflects the properties of an object.
- **Behaviour**: It is represented by methods of an object. It also reflects the response of an object with other objects.
- **Identity**: It gives a unique name to an object and enables one object to interact with other objects.

3. **Inheritance:** Inheritance is a mechanism wherein a new class is derived from an existing class. In Java, classes may inherit or acquire the properties and methods of other classes. A class derived from another class is called a subclass, whereas the class from which a subclass is derived is called a superclass. A subclass can have only one superclass, whereas a superclass may have one or more subclasses.

   For example,

```
open class Person(age: Int) {

// code for eating, talking, walking

}

class MathTeacher(age: Int): Person(age) {

// other features of math teacher

}
```

4. **Polymorphism:** The word polymorphism means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form.
   Types of polymorphism
   Polymorphism is mainly divided into two types:

- **Compile-time Polymorphism:**
  In compile-time polymorphism, the name functions, that is, the signature remains the same but parameters or return type is different. At compile time, the compiler then resolves which functions we are trying to call based on the type of parameters and more.
- **Runtime Polymorphism**:
  In run-time polymorphism, the compiler resolves a call to overridden/overloaded methods at runtime. We can achieve run-time polymorphism using method overriding.

5. **Abstraction**: Abstraction is one of the core concepts of Objected Oriented Programming. When there is a scenario that you are aware of what functionalities a class should have, but not aware of how the functionality is implemented or if the functionality could be implemented in several ways, it is advised to use abstraction. So, when some class extending or implementing this abstraction, they may implement the abstract methods.
   Abstraction in Kotlin could be achieved by following ways:

- Kotlin Interfaces
- Kotlin Abstract Classes

6. **Encapsulation:** OOPS encapsulation in Kotlin unlike Python is enforced and has some fine grained levels (scope modifiers/keywords) which are the following:

- **Private**: With a top-level element it is not accessible outside the Kotlin file it is defined in, elements defined in a class/interface (eg, properties, functions) are only accessible in the same place where they are defined
- **public**: Elements can be accessed anywhere
- **protected**: Same as **private** except sub classes can access class/interface elements (includes member properties and functions unless they are marked **private**), this encapsulation level isn't supported on top level elements
- **internal**: Anything in the module can access elements defined in the same module

2.2 Basics of Kotlin

Kotlin is a modern, trending programming language. Kotlin is easy to learn, especially if you already know Java (it is 100% compatible with Java). Kotlin is used to develop Android apps, server-side apps, and much more.

1. Kotlin Syntax

The fun keyword is used to declare a function. A function is a block of code designed to perform a particular task. In the example above, it declares the main() function. The main() function is something you will see in every Kotlin program. This function is used to execute code. Any code inside the main() function's curly brackets {} will be executed.

```
fun main(){

println("Hello World")

}
```

2. Kotlin Variables

Variables are containers for storing data values.To create a variable, use var or val, and assign a value to it with the equal sign (=):

- **val:** When you create a variable with the val keyword, the value cannot be changed/reassigned.
- **var:** When you create a variable with the var keyword, the value can be changed/reassigned.

```
var variableName=value

val variableName=value
```

3. Kotlin Data Types

In Kotlin, the *type* of a variable is decided by its value:

```
val myNum=5                    // Int

val myDoubleNum=5.99           // Double

val myLetter='D'               // Char

val myBoolean =true            // Boolean

val myText="Hello"             // String
```

4. Kotlin Operators

- **Arithmetic Operators**

  * Multiplication
  / Division
  + Addition
  - Subtraction
  % Remainder/Modulo

- **Relational Operators**

  x == y Returns true if x is equal to y
  x > y Returns true if x is greater than y
  x >= y Returns true if x is greater than or equal to y
  x < y Returns true if x is less than y
  x <= y Returns true if x is less than or equal to y
  x != y Returns true if x is not equal to y

- **Logical Operators**
  && And Operator
  || Or Operator
  ! Not Operator

- **Assignment Operators**
  x += y Add x to y and place result in x
  x -= y Subtract y from x and place result in x
  x *= y Multiply x by y and place result in x

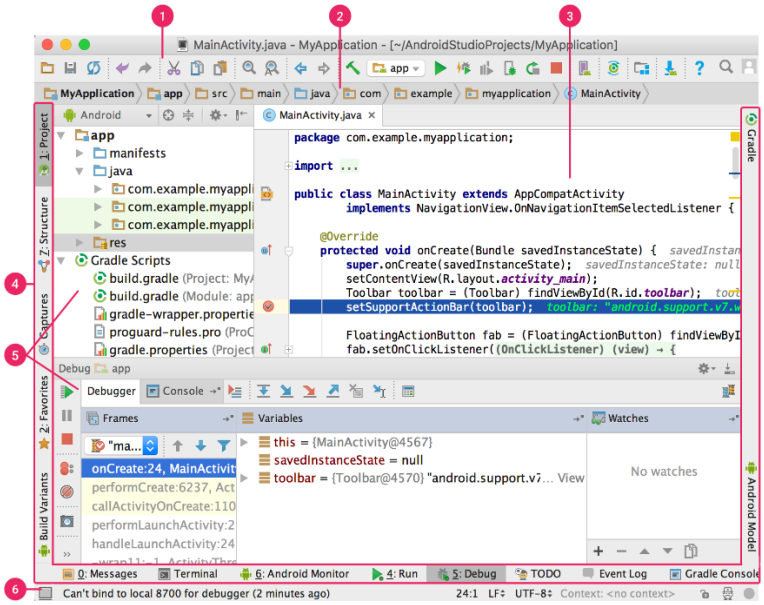
x /= y Divide x by y and place result in x
x %= y Perform Modulo on x and y and place result in x

2.3 Installation

To install Android Studio on Windows, proceed as follows:

1. If you downloaded an .exe file (recommended), double-click to launch it.If you downloaded a .zip file, unpack the ZIP, copy the **android-studio** folder into your **Program Files** folder, and then open the **android-studio > bin** folder and launch studio64.exe (for 64-bit machines) or studio.exe (for 32-bit machines).
2. Follow the setup wizard in Android Studio and install any SDK packages that t recommends.

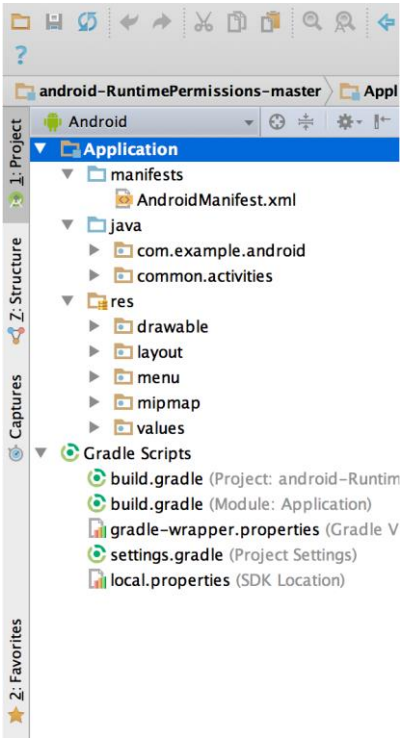
Android Studio is the official Integrated Development Environment (IDE) for Android app development, based on IntelliJ Idea. On top of IntelliJ's powerful code editor and developer tools, Android Studio offers even more features that enhance your productivity when building Android apps, such as:



- A flexible Gradle-based build system
- A fast and feature-rich emulator
- A unified environment where you can develop for all Android devices
- Apply Changes to push code and resource changes to your running app without restarting your app
- Code templates and GitHub integration to help you build common app features and import sample code
- Extensive testing tools and frameworks

- Built-in support for Google Cloud Platform, making it easy to integrate Google Cloud Messaging and App Engine

2.4 Android Studio Structure



Each project in Android Studio contains one or more modules with source code files and resource files. Types of modules include:

- Android app modules
- Library modules
- Google App Engine modules

By default, Android Studio displays your project files in the Android project view, as shown in figure 1. This view is organized by modules to provide quick access to your project's key source files.

All the build files are visible at the top level under **Gradle Scripts** and each app module contains the following folders:

- **manifests**: Contains the AndroidManifest.xml file.
- **java**: Contains the Java source code files, including JUnit test code.
- **res**: Contains all non-code resources, such as XML layouts, UI strings, and bitmap images.

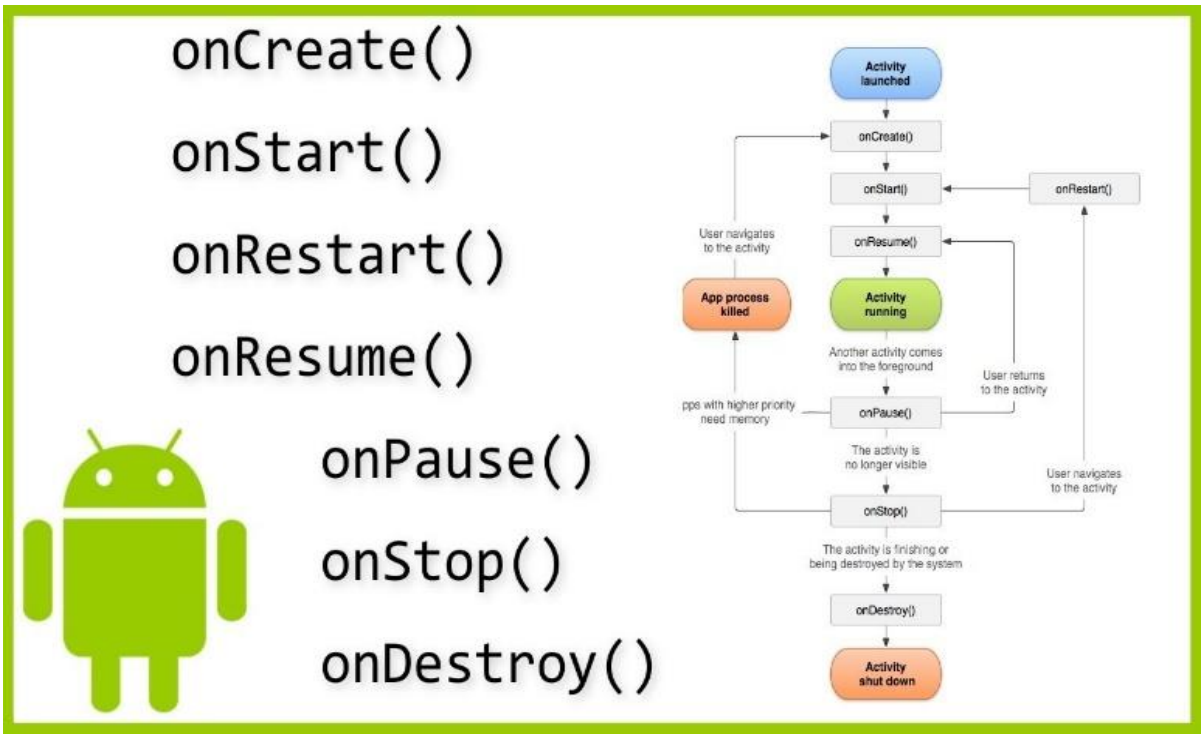
The Android project structure on disk differs from this flattened representation. To see the actual file structure of the project, select **Project** from the **Project** dropdown.

You can also customize the view of the project files to focus on specific aspects of your app development. For example, selecting the **Problems** view of your project displays links to the source files containing any recognized coding and syntax errors, such as a missing XML element closing tag in a layout file.

2.5 Activity

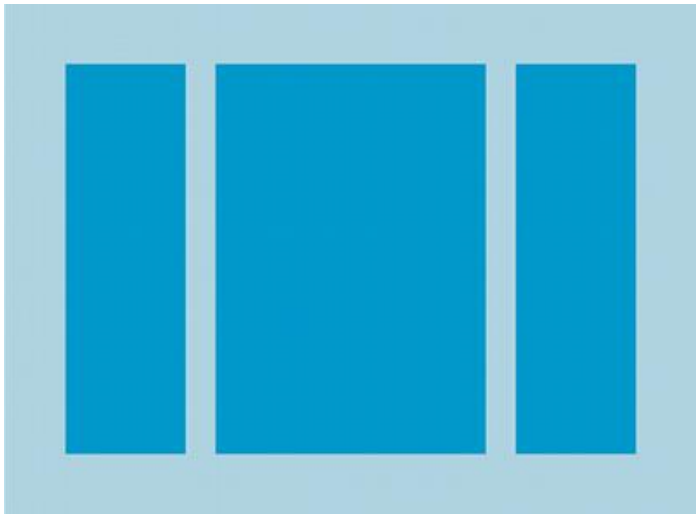
An activity is a single, focused thing that the user can do. Almost all activities interact with the user, so the Activity class takes care of creating a window for you in which you can place your UI with setContentView(View). While activities are often presented to the user as full-screen windows, they can also be used in other ways: as floating windows (via a theme with R.attr.windowIsFloating set), Multi-Window mode or embedded into other windows. There are two methods almost all subclasses of Activity will implement:

- onCreate(Bundle) is where you initialize your activity. Most importantly, here you will usually call setContentView(int) with a layout resource defining your UI, and using findViewById(int) to retrieve the widgets in that UI that you need to interact with programmatically.
- onPause() is where you deal with the user pausing active interaction with the activity. Any changes made by the user should at this point be committed (usually to the ContentProvider holding the data). In this state the activity is still visible on screen.



2.6 Linear Layout

LinearLayout is a view group that aligns all children in a single direction, vertically or horizontally. You can specify the layout direction with the android:orientation attribute.



- All children of a LinearLayout are stacked one after the other, so a vertical list will only have one child per row, no matter how wide they are, and a horizontal list will only be one row high (the height of the tallest child, plus padding). A LinearLayout respects margins between children and the gravity (right, center, or left alignment) of each child.
- Layout_Weight: LinearLayout also supports assigning a weight to individual children with the android:layout_weight attribute. This attribute assigns an "importance" value to a view in terms of how much space it should occupy on the screen. A larger weight value allows it to expand to fill any remaining space in the parent view. Child views can specify a weight value, and then any remaining space in the view group is assigned to children in the proportion of their declared weight. Default weight is zero.
- Unequal distribution:You can also create linear layouts where the child elements use different amounts of space on the screen:
    - ❖ If there are three text fields and two of them declare a weight of 1, while the other is given no weight, the third text field without weight doesn't grow. Instead, this third text field occupies only the area required by its content. The other two text fields, on the other hand, expand equally to fill the space remaining after all three fields are measured.

  The following code snippet shows how layout weights might work in a "send message" activity. The **To** field, **Subject** line, and **Send** button each take up only the height they need. This configuration allows the message itself to take up the rest of the activity's height.

3. PLAYING WITH LAYOUTS

3.1 Relative Layout

RelativeLayout is a view group that displays child views in relative positions. The position of each view can be specified as relative to sibling elements (such as to the left-of or below another view) or in positions relative to the parent RelativeLayout area (such as aligned to the bottom, left or center).

- A RelativeLayout is a very powerful utility for designing a user interface because it can eliminate nested view groups and keep your layout hierarchy flat, which improves performance. If you find yourself using several nested LinearLayout groups, you may be able to replace them with a single RelativeLayout.

- RelativeLayout lets child views specify their position relative to the parent view or to each other (specified by ID). So you can align two elements by right border, or make one below another, centered in the screen, centered left, and so on. By default, all child views are drawn at the top-left of the layout, so you must define the position of each view using the various layout properties available from RelativeLayout.LayoutParams.

Some of the many layout properties available to views in a RelativeLayout include:

- android:layout_alignParentTop**:** If "true", makes the top edge of this view match the top edge of the parent.

- android:layout_centerVertical**:** If "true", centers this child vertically within its parent.
- android:layout_below**:** Positions the top edge of this view below the view specified with a resource ID.

- android:layout_toRightOf**:** Positions the left edge of this view to the right of the view specified with a resource ID..
- The value for each layout property is either a boolean to enable a layout position relative to the parent RelativeLayout or an ID that references another view in the layout against which the view should be positioned.

- In your XML layout, dependencies against other views in the layout can be declared in any order. For example, you can declare that "view1" be positioned below "view2" even if "view2" is the last view declared in the hierarchy. The example below demonstrates such a scenario.

3.2 Constraint Layout

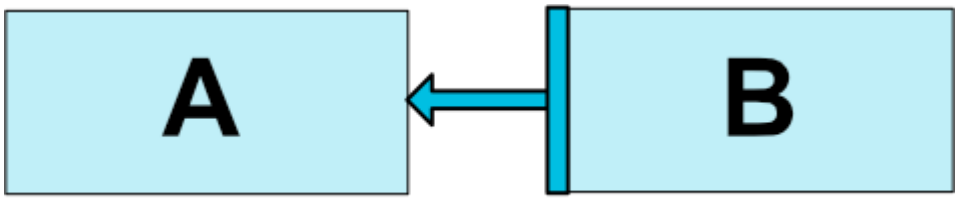
A Constraint Layout is a ViewGroup which allows you to position and size widgets in a flexible way. ConstraintLayout is available as a support library that you can use on Android systems starting with API level 9 (Gingerbread). As such, we are planning on enriching its API and capabilities over time. This documentation will reflect those changes.

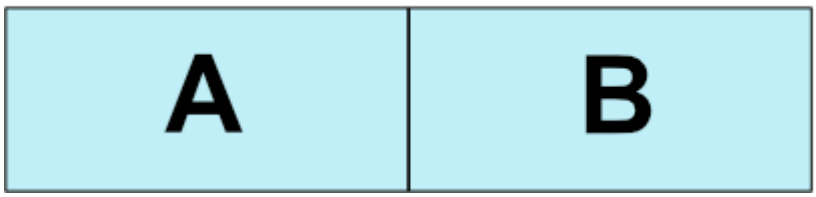
Relative positioning is one of the basic building blocks of creating layouts in ConstraintLayout. Those constraints allow you to position a given widget relative to another one. You can constrain a widget on the horizontal and vertical axis:

- Horizontal Axis: left, right, start and end sides
- Vertical Axis: top, bottom sides and text baseline

The general concept is to constrain a given side of a widget to another side of any other widget.

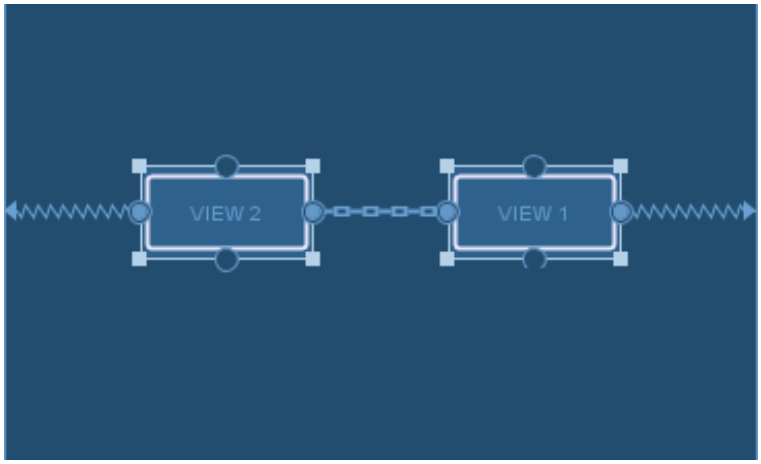
For example, in order to position button B to the right of button A





3.3 Constraint Layout Chains

A chain is a group of views that are linked to each other with bi-directional position constraints. For example, figure shows two views that both have a constraint to each other, thus creating a horizontal chain.



Once chains are set, we can distribute the views horizontally or vertically with the following styles.

1. **Spread:** The views are evenly distributed For eg,
   app:layout_constraintHorizontal_chainStyle=”spread”
   app:layout_constraintVertical_chainStyle=”spread”

2. **Spread inside:** The first and last view are affixed to the constraints on each end of the chain and the rest are evenly distributed. For eg,
   app:layout_constraintHorizontal_chainStyle=”spread_inside”
   app:layout_constraintVertical_chainStyle=”spread_inside”

3. **Packed:** The views are packed together (after margins are accounted for). You can then adjust the whole chain’s bias (left/right or up/down) by changing the chain’s head view bias. E.g.
   app:layout_constraintHorizontal_chainStyle=”packed”
   app:layout_constraintVertical_chainStyle=”packed”

4. **Weighted:** When the chain is set to either **spread** or **spread inside**, you can fill the remaining space by setting one or more views to “match constraints” (0dp). By default, the space is evenly distributed between each view that's set to "match constraints," but you can assign a weight of importance to each view using:

   1. layout_constraintHorizontal_weight
   2. layout_constraintVertical_weight

3.4 UI Views

Views are used to create input and output fields in an Android App. It can be input text field, radio field, image fields etc.
A **ViewGroup** is a special view that can contain other views (called children.) The view group is the base class for layouts and views containers. This class also defines the ViewGroup.LayoutParams class which serves as the base class for layouts parameters.

Various types of views are:

- Text View: This class is used to display text on the android application screen. It also allows user to optionally edit it. Although it contains text editing operations, the basic class does not allow editing, So Edit Text class is included to do so.
  For eg.

```
<TextView
   android:id="@+id/tvWel”
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:text="Welcome!" />
```

- Edit Text: This class makes text to be editable in Android application. It helps in building the data interface taken from any user, also contains certain features through which we can hide the data which are confidential.
  For eg.

```
<EditText
android:id="@+id/myEdittext"
android:layout_width="fill_parent”
android:layout_height="wrap_content"/>
```

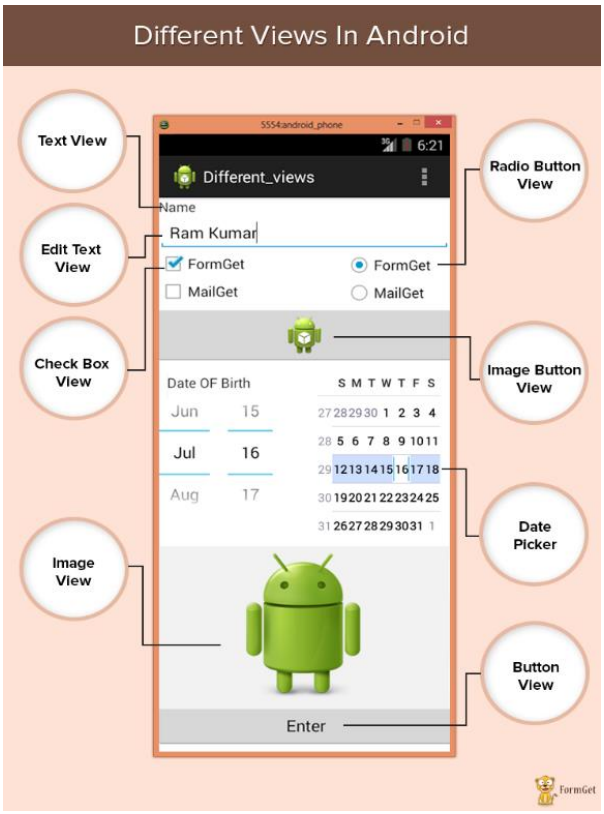
- Image view: Image view helps to display images in an android application. Any image can be selected, we just have to paste our image in a drawable folder from where we can access it. For example:
  For eg,

```
<ImageView
android:id="@+id/myimageview"
android:layout_width="100dp"
android:layout_height="100dp"
android:src="@drawable/ic_launcher" />
```

- Check Box: Checkbox is used in that applications where we have to select one option from multiple provided. Checkbox is mainly used when 2 or more options are present.
  For eg.

```
 <CheckBox
android:id="@+id/checkBox1"
android:layout_width="100dp"
```

android:text="Formget."
android:checked="true" />



- Radio Button: Radio button is like checkbox, but there is slight difference between them. Radio button is a two-states button that can be either checked or unchecked.
  For eg,
  <RadioButton
  android:id="@+id/radioButton1"
  android:layout_width="100dp"
  android:layout_height="wrap_content"
  android:layout_margin="20dp"
  android:text="Formget"
  android:checked="true" />

- Button View: This class is used to create a button on an application screen. Buttons are very helpful in getting into a content. Android button represents a clickable push-button widget.
  For eg,
  <Button
  android:id="@+id/button1"
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:text="Click Here!" />

4. DATA BINDING

4.1 DataBinding

Data Binding allows you to effortlessly communicate across views and data sources. This pattern is important for many Android designs, including model view ViewModel (MVVM), which is currently one of the most common Android architecture patterns.
In Android, the Data Binding Library is a support library that allows you to bind UI components in your layouts to data sources in your app using a declarative format rather than programmatically.
To enable DataBinding:
Navigate to Gradle Scripts > gradle.scripts(module) and add the following code to it.

```
buildFeatures{
   dataBinding = true
 }
```

Below are the advantages of using the Data Binding Library in your Android application:

1. You can reduce findViewById calls and enhance your app's performance
2. Helps get rid of memory leaks or nullPointerExceptions
3. Uses declarative layout, which is more adaptable
4. Supercharges developer productivity by writing error-free, shorter, simpler-to-understand, and more maintainable code
5. Data and Views are separated from each other
6. The compiler verifies types during compile time and displays errors if and when you attempt to assign the incorrect type to a variable, thanks to type-safety

4.2 Intent

Android application components can connect to other Android applications. This connection is based on a task description represented by an Intent object.
Intents are asynchronous messages which allow application components to request functionality from other Android components. Intents allow you to interact with components from the same applications as well as with components contributed by other applications. For example, an activity can start an external activity for taking a picture.

Intents are objects of the android.content.Intent type. Your code can send them to the Android system defining the components you are targeting. For example, via the startActivity() method you can define that the intent should be used to start an activity.An intent can contain data via a Bundle. This data can be used by the receiving component.

Starting activities or services:

To start an activity, use the method startActivity() This method is defined on the Context object which Activity extends. For eg,

```
var intent=Intent(this,NextScreen::class.java)
start Activity(intent)
```

Two types of intents are:

1. **Explicit Intent:** In Explicit intent, we can move from one activity to another, and also we can pass data from one activity to another. Using explicit intent any other component can be specified. For eg,
   var intent=Intent(this,NextScreen::class.java)
   intent,putExtra(“message”,”hello”)
   startActivity(intent)
2. **Implicit Intent:** Android **Implicit Intent** invokes the component of another app to handle the request. It does not specify the component name specifically. For eg,
   intent = Intent(Intent.ACTION_VIEW)
   intent.setData(Uri.parse("https://www.javatpoint.com/"))
   startActivity(intent)
   intent= Intent(Intent.ACTION_VIEW, Uri.parse("https://www.javatpoint.com/"))
   startActivity(intent)

Following table lists down various important Android Intent Standard Actions. You can check Android Official Documentation for a complete list of Actions –

Sr.No	Activity Action Intent & Description
1	**ACTION_ALL_APPS** List all the applications available on the device.
2	**ACTION_ANSWER** Handle an incoming phone call.
3	**ACTION_ATTACH_DATA** Used to indicate that some piece of data should be attached to some other place
4	**ACTION_BATTERY_CHANGED** This is a sticky broadcast containing the charging state, level, and other information about the battery.
5	**ACTION_BATTERY_LOW** This broadcast corresponds to the "Low battery warning" system dialog.

6	**ACTION_BATTERY_OKAY** This will be sent after ACTION_BATTERY_LOW once the battery has gone back up to an okay state.
7	**ACTION_BOOT_COMPLETED** This is broadcast once, after the system has finished booting.
8	**ACTION_BUG_REPORT** Show activity for reporting a bug.
9	**ACTION_CALL** Perform a call to someone specified by the data.
10	**ACTION_CALL_BUTTON** The user pressed the "call" button to go to the dialer or other appropriate UI for placing a call.
11	**ACTION_CAMERA_BUTTON** The "Camera Button" was pressed.
12	**ACTION_CHOOSER** Display an activity chooser, allowing the user to pick what they want to before proceeding.
13	**ACTION_CONFIGURATION_CHANGED** The current device Configuration (orientation, locale, etc) has changed.
14	**ACTION_DATE_CHANGED** The date has changed.

5. NAVGRAPHS & FRAGMENTS

5.1 Fragments

- **Android Fragment** is the part of activity, it is also known as sub-activity. There can be more than one fragment in an activity. Fragments represent multiple screen inside one activity.

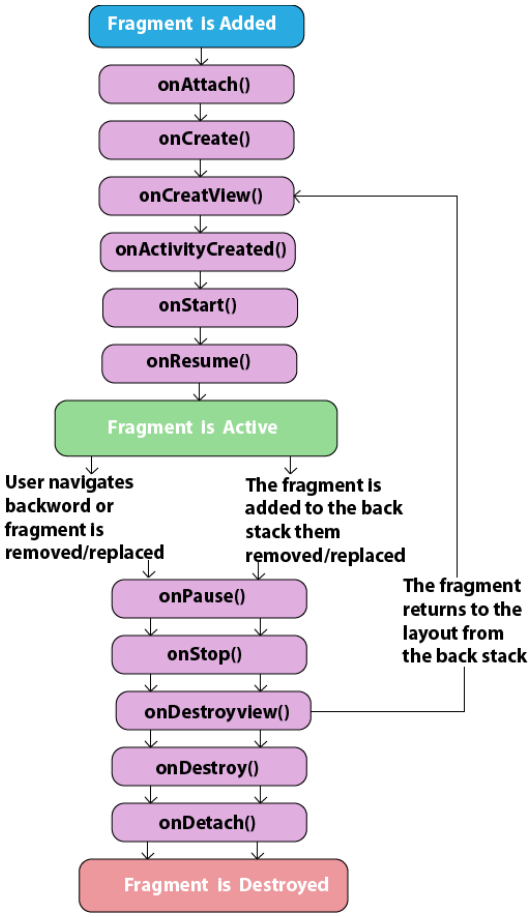
- Android fragment lifecycle is affected by activity lifecycle because fragments are included in activity.

- Each fragment has its own life cycle methods that is affected by activity life cycle because fragments are embedded in activity.

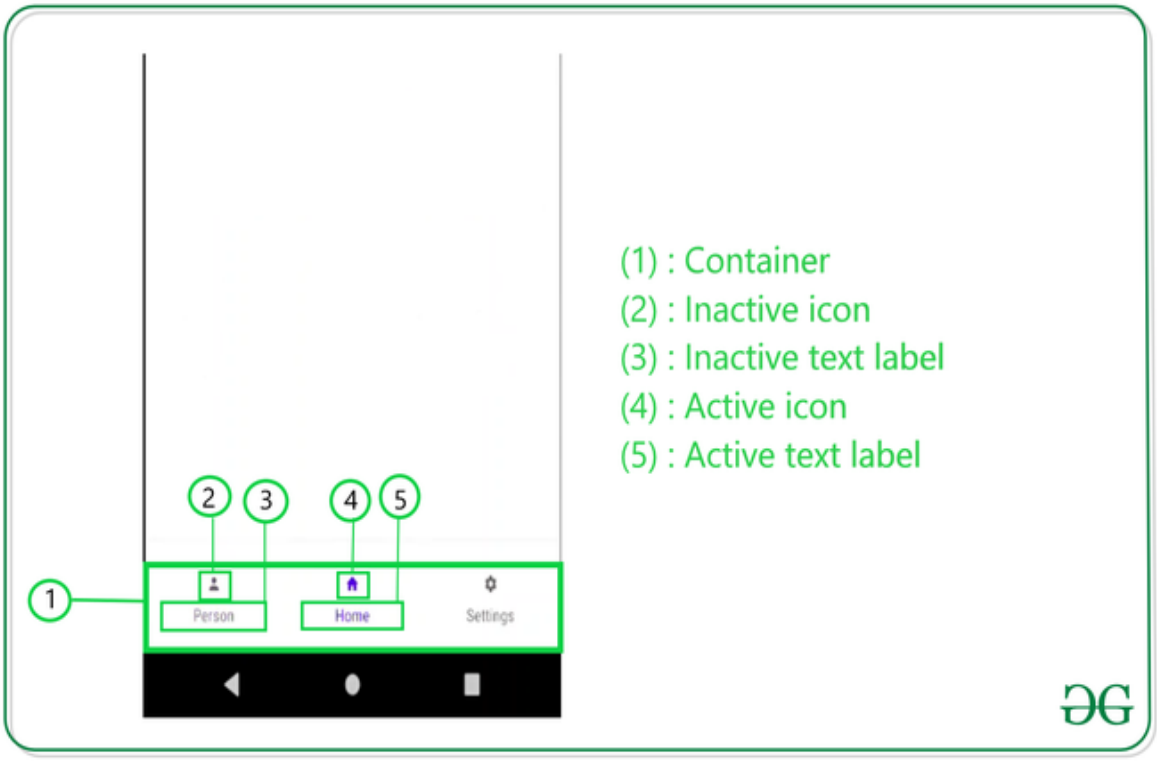
- The **FragmentManager** class is responsible to make interaction between fragment objects.

Android Fragment Lifecycle

The lifecycle of android fragment is like the activity lifecycle. There are 12 lifecycle methods for fragment.



5.2 Bottom Navigation Bar



Bottom navigation bar allows the user to switch to different activities/fragments easily. It makes the user aware of the different screens available in the app. The user is able to check which screen are they on at the moment.

For eg,
XML file:

```
<com.google.android.material.bottomnavigation.BottomNavigationView
 android:id="@+id/bottom_navigatin_view"
 android:layout_width="match_parent"
 android:layout_height="wrap_content"
 app:layout_constraintBottom_toBottomOf="parent"
 app:layout_constraintLeft_toLeftOf="parent"
 app:layout_constraintRight_toRightOf="parent" />
```

Add Menu Items:

The next step is to add menu items for the bottom navigation view. **Right-click** on the **res** directory, select **New > Android Resource File** and select menu, provide a nameand click **OK**.

For eg,

```
<?xml version="1.0" encoding="utf-8"?>

<menu xmlns:app="http://schemas.android.com/apk/res-auto"

 xmlns:android="http://schemas.android.com/apk/res/android">

    <item android:title="Home" />

    <item android:title="Profile" />

    <item android:title="Setting" />

</menu>
```

5.3 Passing data in fragments

To pass the data from one fragment to another in a better way and less code space ie done by using Bundles in Android. Android Bundles are generally used for passing data from one activity or fragment to another. Basically here concept of key-value pair is used where the data that one wants to pass is the value of the map, which can be later retrieved by using the key. Bundles are used with intent and values are sent and retrieved in the same fashion, as it is done in the case of Intent. It depends on the user what type of values the user wants to pass, but bundles can hold all types of values (int, String, boolean, char) and pass them to the new activity or fragment For eg,

```
// creating the instance of the bundle

val bundle = Bundle()

// storing the string value in the bundle

// which is mapped to key

bundle.putString("key1", "Gfg :- Main Activity")

// creating a intent

intent = Intent(this@MainActivity, SecondActivity::class.java)

// passing a bundle to the intent

intent.putExtras(bundle)

// starting the activity by passing the intent to it.

startActivity(intent).
```

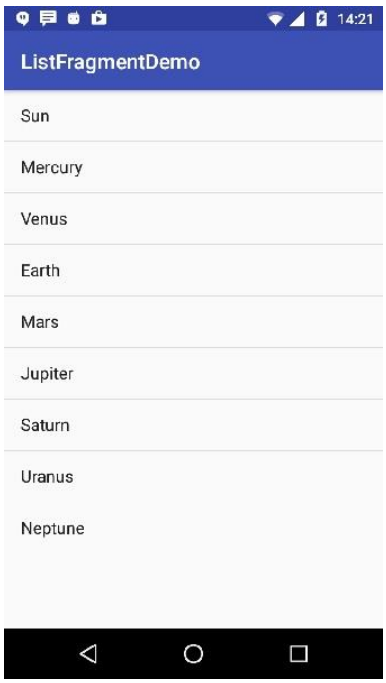
5.4 List Fragment

ListFragment hosts a ListView object that can be bound to different data sources, typically either an array or a Cursor holding query results. Binding, screen layout, and row layout are discussed in the following sections.

Screen Layout
ListFragment has a default layout that consists of a single list view. However, if you desire, you can customize the fragment layout by returning your own view hierarchy from onCreateView(LayoutInflater, ViewGroup, Bundle). The view hierarchy *must* contain a ListView object with the id "@android:id/list" (or R.id.list if it's in code)Optionally, your view hierarchy can contain another view object of any type to display when the list view is empty. This "empty list" notifier must have an id "android:empty". Note that when an empty view is present, the list view will be hidden when there is no data to display. For eg.

```
<ListView android:id="@id/android:list"
 android:layout_width="match_parent"
android:layout_height="match_parent"
 android:background="#00FF00"
 android:layout_weight="1"
 android:drawSelectorOnTop="false"/>
```

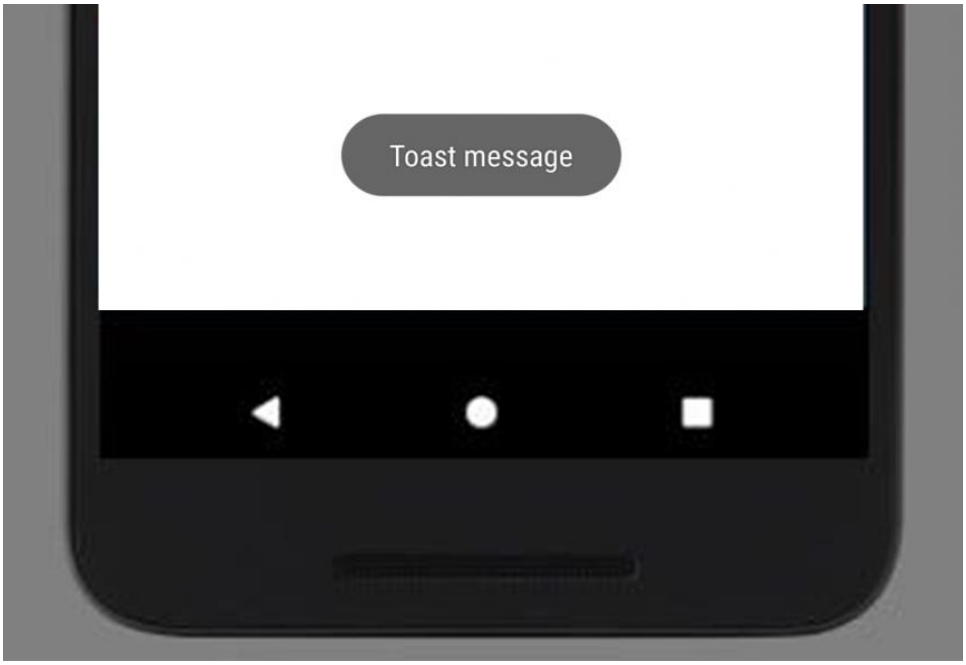
Binding to Data
You bind the ListFragment's ListView object to data using a class that implements the ListAdapter interface. Android provides two standard list adapters: SimpleAdapter for static data (Maps), and SimpleCursorAdapter for Cursor query results.



6. MESSAGES

6.1 Toast

A Toast is a feedback message. It takes a very little space for displaying while overall activity is interactive and visible to the user. It disappears after a few seconds. It disappears automatically. If user wants permanent visible message, Notification can be used.



- The **applicationContest** returns the instance of Context class.
- The message is of String type as **("this is toast message")**.
- The **Toast.LENGTH_SHORT** and **Toast.LENGTH_LONG** are the constant defines the time duration for message display.
- The **show()** method of Toast class used to display the toast message, without this function the Toast will not be displayed.
- The **setGravity()** method of Toast used to customize the position of toast message.

For eg.

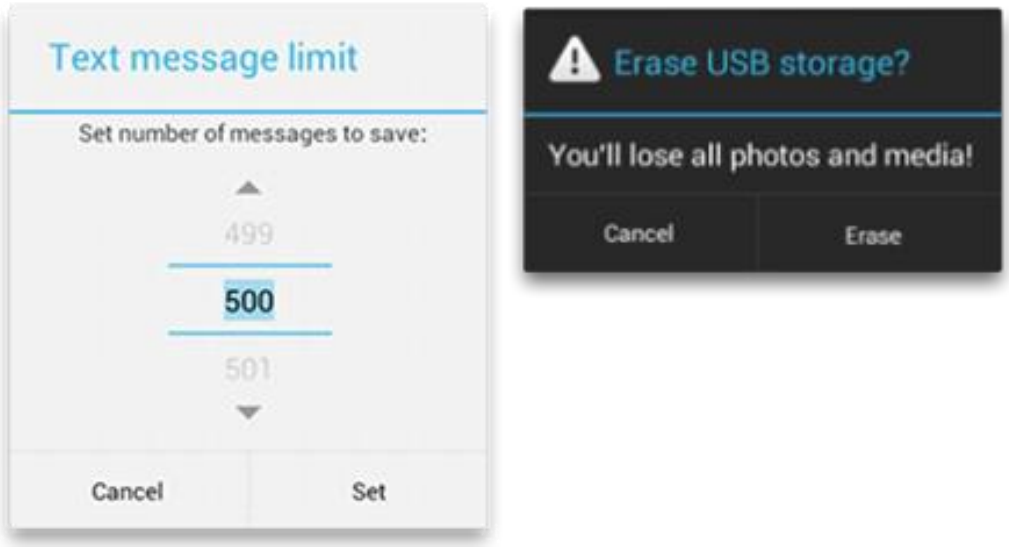
```
Toast.makeText(this,"this is toast message",Toast.LENGTH_SHORT).show()
```

6.2 Dialog

A dialog is a small window that prompts the user to make a decision or enter additional information. A dialog does not fill the screen and is normally used for modal events that require users to take an action before they can proceed.

Dialog Design

For information about how to design your dialogs, including recommendations for language, read the Dialogs design guide.



The Dialog class is the base class for dialogs, but you should avoid instantiating Dialog directly. Instead, use one of the following subclasses:

AlertDialog **:** A dialog that can show a title, up to three buttons, a list of selectable items, or a custom layout.

DatePickerDialog **or** TimePickerDialog**:** A dialog with a pre-defined UI that allows the user to select a date or time.For eg.

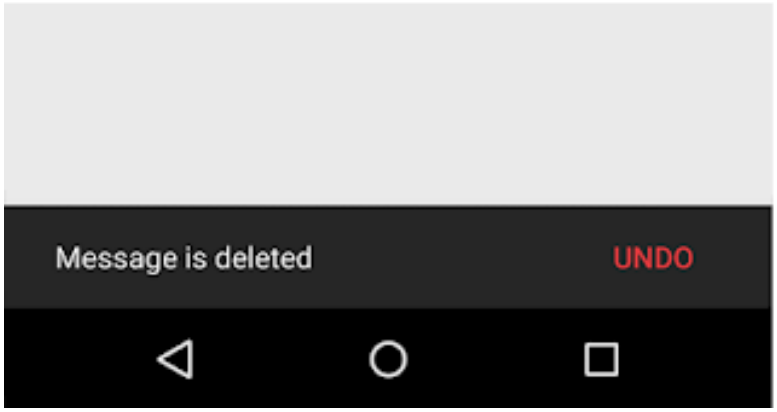
```
val builder = AlertDialog.Builder(it)
    builder.setMessage(R.string.dialog_start_game)
            .setPositiveButton(R.string.start,
                DialogInterface.OnClickListener { dialog, id ->
                    // START THE GAME!
                })
            .setNegativeButton(R.string.cancel,
                DialogInterface.OnClickListener { dialog, id ->
                    // User cancelled the dialog
                })
        // Create the AlertDialog object and return it
        builder.create()
```

6.3 Snackbar

Snackbars provide lightweight feedback about an operation. They show a brief message at the bottom of the screen on mobile and lower left on larger devices. Snackbars appear above all other elements on screen and only one can be displayed at a time.

They automatically disappear after a timeout or after user interaction elsewhere on the screen, particularly after interactions that summon a new surface or activity. Snackbars can be swiped off screen.Snackbars can contain an action which is set via setAction(CharSequence, android.view.View.OnClickListener).

For eg,




```
val snackBar = Snackbar.make(
        it, "Replace with your own action",
        Snackbar.LENGTH_LONG
    ).setAction("Action", null)
    snackBar.setActionTextColor(Color.BLUE)
    val snackBarView = snackBar.view
    snackBarView.setBackgroundColor(Color.CYAN)
    val textView =
snackBarView.findViewById(com.google.android.material.R.id.snackbar_text) as TextView
    textView.setTextColor(Color.BLUE)
    snackBar.show()
```

7. RECYCLER VIEW

7.1 Recycler View

RecyclerView is a ViewGroup added to the android studio as a successor of the GridView and ListView. It is an improvement on both of them and can be found in the latest v-7 support packages. It has been created to make possible construction of any lists with **XML** layouts as an item which can be customized vastly while improving on the efficiency of ListViews and GridViews. This improvement is achieved by recycling the views which are out of the visibility of the user. For example, if a user scrolled down to a position where items 4 and 5 are visible; items 1, 2, and 3 would be cleared from the memory to reduce memory consumption.
Implementation: To implement a basic RecyclerView three sub-parts are needed to be constructed which offer the users the degree of control they require in making varying designs of their choice.

1. **The Card Layout:** The card layout is an XML layout which will be treated as an item for the list created by the RecyclerView.
2. **The ViewHolder:** The ViewHolder is a java class that stores the reference to the card layout views that have to be dynamically modified during the execution of the program by a list of data obtained either by online databases or added in some other way.
3. **The Data Class:** The Data class is a custom java class that acts as a structure for holding the information for every item of the RecyclerView.

The Adapter: The adapter is the main code responsible for RecyclerView. It holds all the important methods dealing with the implementation of RecylcerView. The basic methods for a successful implementation are:

- onCreateViewHolder: which deals with the inflation of the card layout as an item for the RecyclerView.
- onBindViewHolder: which deals with the setting of different data and methods related to clicks on particular items of the RecyclerView.
- getItemCount: which Returns the length of the RecyclerView.
- onAttachedToRecyclerView: which attaches the adapter to the RecyclerView.

For eg,

```
package com.japnoor.anticorruptionhelpline
import android.view.LayoutInflater
import android.view.ViewGroup
import androidx.recyclerview.widget.RecyclerView
import com.japnoor.anticorruptionhelpline.databinding.ItemComlaintBinding
class ComplaintsAdapter(var complaintsEntity: ArrayList<ComplaintsEntity>) :
RecyclerView.Adapter<ComplaintsAdapter.ViewHolder>() {

class ViewHolder(var binding : ItemComlaintBinding) :
RecyclerView.ViewHolder(binding.root){
```

```
}

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ViewHolder {
        return
ViewHolder(ItemComlaintBinding.inflate(LayoutInflater.from(parent.context),parent,false))
    }

    override fun onBindViewHolder(holder: ViewHolder, position: Int) {
         holder.binding.tvAgainst.setText(complaintsEntity[position].complaintAgainst)
        holder.binding.tvSummary.setText(complaintsEntity[position].complaintSummary)
        holder.binding.Date.setText(complaintsEntity[position].date)
    }

    override fun getItemCount(): Int {
return complaintsEntity.size
    }
}
```

7.2 Interfaces

Interfaces in Kotlin can contain declarations of abstract methods, as well as method implementations. What makes them different from abstract classes is that interfaces cannot store state. They can have properties, but these need to be abstract or provide accessor implementations.

An interface is defined using the keyword interface:

```
interface MyInterface {
   fun bar()
   fun foo() {
     // optional body
   }
}
```

Implementing interfaces

A class or object can implement one or more interfaces:

```
class Child : MyInterface
{
   override fun bar()
{
      // body
   }
}
```

8. PERMISSIONS

8.1 Permissions

- Every Android app runs in a limited-access sandbox. If your app needs to use resources or information outside of its own sandbox, you can declare a permission and set up a permission request that provides this access. These steps are part of the workflow for using permissions.

- If you declare any dangerous permissions, and if your app is installed on a device that runs Android 6.0 (API level 23) or higher, you must request the dangerous permissions at runtime by following the steps in this guide.

- If you don't declare any dangerous permissions, or if your app is installed on a device that runs Android 5.1 (API level 22) or lower, the permissions are automatically granted, and you don't need to complete any of the remaining steps on this page.
  Basic principles
  The basic principles for requesting permissions at runtime are as follows:

  - Ask for permissions in context, when the user starts to interact with the feature that requires it.
  - Don't block the user. Always provide the option to cancel an educational UI flow related to permissions.
  - If the user denies or revokes a permission that a feature needs, gracefully degrade your app so that the user can continue using your app, possibly by disabling the feature that requires the permission.
  - Don't assume any system behavior. For example, don't assume that permissions appear in the same *permission group*. A permission group merely helps the system minimize the number of system dialogs that are presented to the user when an app requests closely-related permissions.

  Determine whether your app was already granted the permission

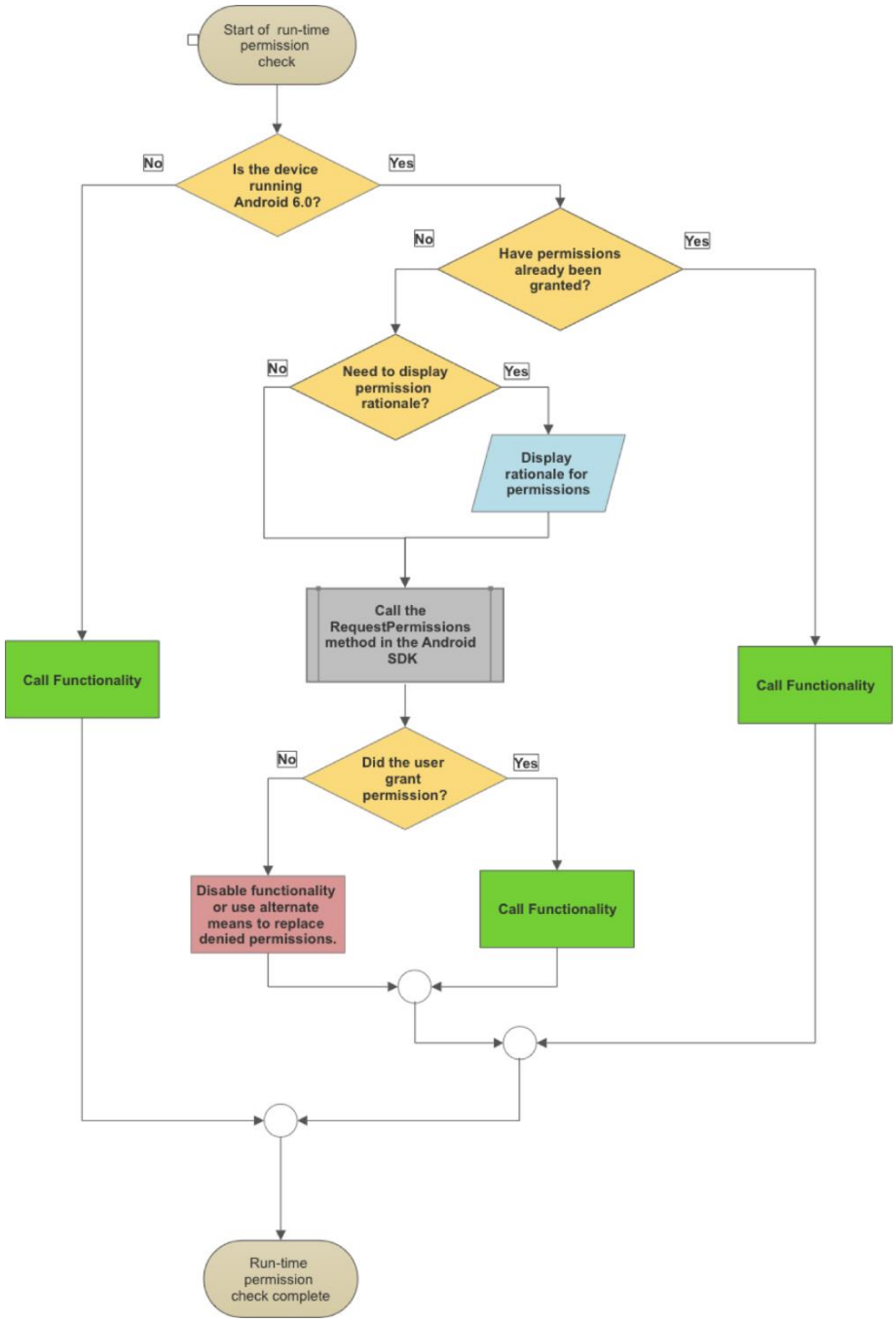
  To check if the user has already granted your app a particular permission, pass that permission into the ContextCompat.checkSelfPermission() method. This method returns either PERMISSION_GRANTED or PERMISSION_DENIED, depending on whether your app has the permission.
  The following steps show how to use the RequestPermission contract. The process is nearly the same for the RequestMultiplePermissions contract.

1. In your activity or fragment's initialization logic, pass in an implementation of ActivityResultCallback into a call to registerForActivityResult().
   The ActivityResultCallback defines how your app handles the user's response to the permission request.

Keep a reference to the return value of registerForActivityResult(), which is of type ActivityResultLauncher.

2. To display the system permissions dialog when necessary, call the launch() method on the instance of ActivityResultLauncher that you saved in the previous step.

   After launch() is called, the system permissions dialog appears. When the user makes a choice, the system asynchronously invokes your implementation of ActivityResultCallback, which you defined in the previous step.



9. SAVING DATA

9.1 Shared Preferences

Shared Preferences is the way in which one can store and retrieve small amounts of primitive data as key/value pairs to a file on the device storage such as String, int, float, Boolean that make up your preferences in an XML file inside the app on the device storage.
Shared Preferences can be thought of as a dictionary or a key/value pair.
For example, you might have a key being "username" and for the value, you might store the user's username. And then you could retrieve that by its key (here username).
You can have a simple shared preference API that you can use to store preferences and pull them back as and when needed. Shared Preferences class provides APIs for reading, writing, and managing this data.

Let's look at some important methods for SharedPreferences.

- getSharedPreferences(String,Int) method is used to retrieve an instance of the SharedPreferences. Here String is the name of the SharedPreferences file and int is the Context passed.
- The SharedPrefernces.Editor() is used to edit values in the SharedPrefernces.
- We can call commit() or apply() to save the values in the SharedPreferences file. The commit() saves the values immediately whereas apply() saves the values asynchronously.

We can set values on our SharedPreference instance using Kotlin in the following way.

```
val sharedPreference =
getSharedPreferences("PREFERENCE_NAME",Context.MODE_PRIVATE)

var editor = sharedPreference.edit()
editor.putString("username","Anupam")
editor.putLong("l",100L)
editor.commit()
```

For retrieving a value:

```
sharedPreference.getString("username","defaultName")
sharedPreference.getLong("l",1L)
```

We can also clear all the values or remove a particular value by calling clear() and remove(String key) methods.

```
editor.clear()
editor.remove("username")
```

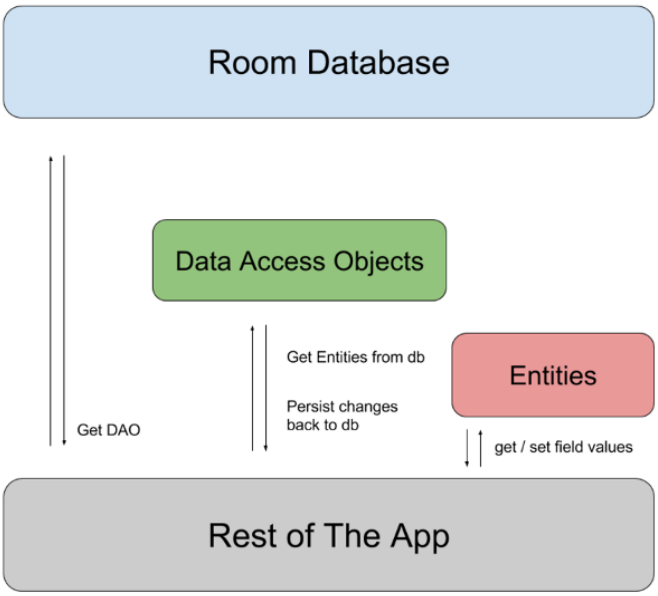
9.2 Room DataBase

Apps that handle non-trivial amounts of structured data can benefit greatly from persisting that data locally. The most common use case is to cache relevant pieces of data so that when the device cannot access the network, the user can still browse that content while they are offline.

The Room persistence library provides an abstraction layer over SQLite to allow fluent database access while harnessing the full power of SQLite. In particular, Room provides the following benefits:

- Compile-time verification of SQL queries.
- Convenience annotations that minimize repetitive and error-prone boilerplate code.
- Streamlined database migration paths.

Because of these considerations, we highly recommend that you use Room instead of using the SQLite APIs directly.



Primary components

There are three major components in Room:

- The database class that holds the database and serves as the main access point for the underlying connection to your app's persisted data.
- Data entities that represent tables in your app's database.
- Data access objects (DAOs) that provide methods that your app can use to query, update, insert, and delete data in the database.

Data entity

The following code defines a User data entity. Each instance of User represents a row in a user table in the app's database.

```
@Entity
data class User(
   @PrimaryKey val uid: Int,
   @ColumnInfo(name = "first_name") val firstName: String?,
   @ColumnInfo(name = "last_name") val lastName: String?
)
```

Data access object (DAO)

The following code defines a DAO called UserDao. UserDao provides the methods that the rest of the app uses to interact with data in the user table.

```
@Dao
interface UserDao {
   @Query("SELECT * FROM user")
   fun getAll(): List<User>

   @Query("SELECT * FROM user WHERE uid IN (:userIds)")
   fun loadAllByIds(userIds: IntArray): List<User>

   @Delete
   fun delete(user: User)
}
```

Database

The following code defines an App Database class to hold the database. App Database defines the database configuration and serves as the app's main access point to the persisted data. The database class must satisfy the following conditions:

- The class must be annotated with a @Database annotation that includes an entities array that lists all of the data entities associated with the database.
- The class must be an abstract class that extends RoomDatabase.
- For each DAO class that is associated with the database, the database class must define an abstract method that has zero arguments and returns an instance of the DAO class.

```
@Database(entities = [User::class], version = 1)
abstract class AppDatabase : RoomDatabase() {
   abstract fun userDao(): UserDao
}
```