**OOPS CONCEPT JAVA**

**Introduction to java :**

java is object oriented programming language , so everything in java is object.

**Object :**

Something got state and behavoiur.

example : chair,house

java is platform independent language , you can compile on one platform and can run on other platform.

Java takes care on memory allocation , and takes care of garbage collection.

**JDK:**

java runtime environment = java virtual machine + set of libraries.

Javac complier gives a byte code (.class )file which can be runned on any platform using jvm.

**Jdk = jvm + Development tools for example javac,java etc..**

To run a .class file we need a jre but for development we need jdk.

**Modifiers:**

To control the visibility of class or method.

Example:

Public, private ,protected.

Access modifiers:

Public:

It is visible to world

Default:

It is visible within the package.

Private:

Only visible to class .

**Protected:**

Visible to packages and all the sub classes.

Non access Modifiers:

Static:

The static modifers foe calling methods and variables without an object to which it belongs.

final:

we can’t change once implemented.

**Abstract:**

When there is no implementation for all the methods.

Example :

Shape class have area method with no definition., so if rectangle class inherits the shape class so it should define what area of rectangle is within the class.

So here area method is abstract, if abstract method is present in class the class should be abstract.

**Syncronised:**

When we are using multiple threads so if all the threads trying to call the same method then we get an issue so to avoid this we use syncronised ( parallel execution)

Only one thread can access at one point.

**Variable:**

Variable is memory location which are reserved to store data.

Local:

It is local to our method.

Visibility is within method.

Instance:

Variable at class level, it means all methods can access it.

Static:

It is there one for class

All the objects created are stored in heap memory the only references of that class present in ram.

Methods:

A method is group of statements

We can reusability of block of code(method) that minimises redundancy.

Class may have multiple methods.

**Method overloading :**

When same method id defined again with different parameters

, different return types etc…

**Arrays:**

It is nothing but grouping of same data types, referred by command name.

Specific element is accessed using index.

Declare :

Int []a = {2,3,4,5,6};

Int[]a = new int [5];//we are just defining but no giving any values

We are just allocating memory for that.

**List:**

**List** in Java provides the facility to maintain the ordered collection. It contains the index-based methods to insert, update, delete and search the elements. It can have the duplicate elements also. We can also store the null elements in the list.

There are four types of lists they are :

ArrayList,LinkedList,Stack,Vector.

ArrayList:

In arrays when we initialise the size then we can’t change the size of array but if we need to change the size of array in runtime we should use arraylist.

Declare :

ArrayList<String> list = new ArrayList<>();

To add elements in arraylist we use add method as follows

List.add(“abc”);

List.add(“efg”);

Output:

The output will be as follows

abc efg

**Set:**

The set interface present in the [java.util](https://www.geeksforgeeks.org/java-util-package-java/) package and extends the [Collection interface](https://www.geeksforgeeks.org/collections-in-java-2/) is an unordered collection of objects in which duplicate values cannot be stored.

For example:

// Java program to demonstrate a Set

import java.util.\*;

public class SetExample{

    public static void main(String[] args)

    {

        // Set demonstration using HashSet

        Set<String> hash\_Set

            = new HashSet<String>();

        hash\_Set.add("Geeks");

        hash\_Set.add("For");

        hash\_Set.add("Geeks");

        hash\_Set.add("Example");

        hash\_Set.add("Set");

        System.out.println(hash\_Set);

    }

}

Output:

[Set, Example, Geeks, For]

**String:**

It is a sequence of characters.

Once the string is initialised we cannot change ,

String is immutable.

If multiple threads using same string then we no need to use synchronized java itself protects. multiple threads using same strings without tampering it.

String operations:

1. Compare To():

Example:

String s1 = “abcd”;

String s2 = “ efgh”;

S1.compareTo(s2); returns 0;

1. isEmpty():

it gives true if it is empty and false if not.

3.toUppercase();, toLowercase();

4.valueof():

Converts int to string;

Ex:

Int i=100

String.val(100).

5. replace():

Used to replace string with other.

6.contains();

7.endswith();

There are three different variants of strings

**Stringbuffers :**

It is used to create mutable string , it is used for multiple threads.

If string is to be modified like appending inserting etc.. we can use string buffer for that.

Replace(): we can replace characters form position of string buffer.

Delete():we can delete characters of string.

Reverse(), capacity() etc..

A string buffer is like a String, but can be modified

Declare:

[StringBuffer](https://www.tutorialspoint.com/java/lang/stringbuffer_append_boolean.htm) str = new [StringBuffer](https://www.tutorialspoint.com/java/lang/stringbuffer_append_boolean.htm)();

We can append using str.append(‘c’);

To delete

[str.delete(int start, int end)](https://www.tutorialspoint.com/java/lang/stringbuffer_delete.htm)

to know the capacity

str.capacity(); it returns the total capacity .

to know the length

str.length() it returns total no of character count.

To replace

[str.replace(int start, int end, String str)](https://www.tutorialspoint.com/java/lang/stringbuffer_replace.htm)

**String builder:**

If there is no need of syncronisation then it is better to take stringbuilder ,because string buffer is thread safe and also it is not faster than stringbuilder so , it is better to use string builder if no syncronisation is needed.

Insert();

Reverse();

Etc…

For example to explain differences between strings ,stringbuilder, string buffer

// Java program to demonstrate difference between String,

// StringBuilder and StringBuffer

class StringTypes

{

    // Concatenates to String

    public static void concat1(String s1)

    {

        s1 = s1 + "forgeeks";// here s1 is internal variable od concat1 function. It won’t change the actual string in main method this is called immutable.

    }

    // Concatenates to StringBuilder

    public static void concat2(StringBuilder s2)

    {

        s2.append("forgeeks");

    }

    // Concatenates to StringBuffer

    public static void concat3(StringBuffer s3)

    {

        s3.append("forgeeks");

    }

    public static void main(String[] args)

    {

        String s1 = "Geeks";

        concat1(s1);  // s1 is not changed

        System.out.println("String: " + s1);

        StringBuilder s2 = new StringBuilder("Geeks");

        concat2(s2); // s2 is changed

        System.out.println("StringBuilder: " + s2);

        StringBuffer s3 = new StringBuffer("Geeks");

        concat3(s3); // s3 is changed

        System.out.println("StringBuffer: " + s3);

    }

}

Output:

String: Geeks

StringBuilder: Geeksforgeeks

StringBuffer: Geeksforgeeks

OOPs:

Classes : it got state and behavoiur.

Behavoiur is nothing but methods of class and state is variables.

Class student;

We can create object using

Student s = new Student();

**Naming convention:**

Class name it should start with upper case(camel casing)

Interface starts with uppercase.

Methods should start with lowercase.

Variable name starts with lowercase.

These are followed by developers .

**Constructor():**

They are used basically initialisation , preprocessing ,

It is block of code initialise object,

It should have same name as class name with no return type.

**Parametrised constructor:**

Constructor with parameters called as parametrised constructor.

Constructor overloading:

Constructors with different parameter list.

Constructor chaining:

When one constructor is calling another constructor is called construction chaining.

**Static:**

It is variable which is one for class.

Objects with the class an get the static variable .

**This keyword:**

It is used to invoke the present class methods, and to represent current class variables

**Oops concept:**

We have four oops concepts in java they are :-

1.**inheritance**

**2.polymorphism**

**3.abstraction**

**4.encapsulation**

1. **Inheritance:-**

Inheritance is a process where one class acquires the properties of another class

Inheritance is an important pillar of OOP(Object Oriented Programming). It is the mechanism in java by which one class is allow to inherit the features(fields and methods) of another class.

## The class which inherits the properties of other is known as subclass (derived class, child class) and the class whose properties are inherited is known as superclass (base class, parent class).

## extends Keyword

**extends** is the keyword used to inherit the properties of a class. Following is the syntax of extends keyword.

**Syntax**

class Super {

.....

.....

}

class Sub extends Super {

.....

.....

}

class abc{

public int a;

}

Clas efg extends abc{

Int b;

}

public class Main

{

public static void main(String[] args) {

efg s = new efg();

s.a = 20; // a variable is inherited from abc class by efg class.

System.out.println("Hello World " + s.a);

}

}

Output:

Hello world 20.

The super keyword

The super keyword is similar to this keyword. Following are the scenarios where the super keyword is used.

* It is used to differentiate the members of superclass from the members of subclass, if they have same names.
* It is used to invoke the superclass constructor from subclass.

We can call super class constructor by using super();

To know wheather the class is inheried or not we use **instanceof**  keyword which returns true if it is a sub class of the super class or false if class is not inherited by super class.

**2.Polymorphism:-**

Polymorphism is the ability of an object to take on many forms.

The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object

When a method behaves differently based on object calling it is polymorphism (method overriding).

In a class when when a same methods is created with different parameters or different return types etc.. are comes under method overloading.

So specific method is called when based on no of parameters provided or type of parameters provided.

Example:

Public methodDemo{

Public void area(int base, int height){

System.out.println(“area of rectangle “+ base \* height);

}

Public void area(int radius){

System.out.println(“area of circle “+ 3.14\*radius\*radius);

}

Public static void main(string [] args){

MethodDemo abc = new MethodDemo();

abc.area(5,4);// this invokes the area of rectangle

abc.area(6);// this invokes the area if circle method.

}

### Example

Let us look at an example.

public interface Vegetarian{}

public class Animal{}

public class Deer extends Animal implements Vegetarian{}

Now, the Deer class is considered to be polymorphic since this has multiple inheritance.

* A Deer IS-A Animal
* A Deer IS-A Vegetarian
* A Deer IS-A Deer
* A Deer IS-A Object

Same Deer class represents animal and vegetarian this method is called polymorphism .

Super keyword:

Is used for parent class objects.

**3.Abstrcaction :-**

Abstraction is process where the method is declared but not defined is called abstraction .

If a class contains one abstract method then the class should be declared as abstract.

If a class is abstract then it is not necessary to have methods of abstract . but when atleast one method is abstract in a class then it should be declared as abstract.

**Example:-**

abstract class Shape

{

    String color;

    // these are abstract methods

    abstract double area();

    public abstract String toString();

    // abstract class can have constructor

    public Shape(String color) {

        System.out.println("Shape constructor called");

        this.color = color;

    }

    // this is a concrete method

    public String getColor() {

        return color;

    }

}

class Circle extends Shape

{

    double radius;

    public Circle(String color,double radius) {

        // calling Shape constructor

        super(color);

        System.out.println("Circle constructor called");

        this.radius = radius;

    }

    @Override

    double area() {

        return Math.PI \* Math.pow(radius, 2);

    }

    @Override

    public String toString() {

        return "Circle color is " + super.color +

                       "and area is : " + area();

    }

}

class Rectangle extends Shape{

    double length;

    double width;

    public Rectangle(String color,double length,double width) {

        // calling Shape constructor

        super(color);

        System.out.println("Rectangle constructor called");

        this.length = length;

        this.width = width;

    }

    @Override

    double area() {

        return length\*width;

    }

    @Override

    public String toString() {

        return "Rectangle color is " + super.color +

                           "and area is : " + area();

    }

}

public class Test

{

    public static void main(String[] args)

    {

        Shape s1 = new Circle("Red", 2.2);

        Shape s2 = new Rectangle("Yellow", 2, 4);

        System.out.println(s1.toString());

        System.out.println(s2.toString());

    }

}

Abstract classes cannot be instantiated it means we cannot create object of abstract class.

If you inherit an abstract class, you have to provide implementations to all the abstract methods in it.

**Interface:**-

Interface is a class with all abstract classes.

Interfaces cannot be inherited it should be implemented.

The class which implements interfaces should define the abstract methods.

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

**Example:-**

interface Animal {

public void eat();

public void travel();

}

public class MammalInt implements Animal {

public void eat() {

System.out.println("Mammal eats");

}

public void travel() {

System.out.println("Mammal travels");

}

public int noOfLegs() {

return 0;

}

public static void main(String args[]) {

MammalInt m = new MammalInt();

m.eat();

m.travel();

}

}

You can extend interface with another interface.

Default implementation In interface:

\*\*\* We can implement a default method in interface ,so all the classes implement this interface inherits this method but they can’t override this method , they can use this method.

Example:

Interface welcome{

Default void say(){

System.out.println(“hello”);

hi();

}

Void hello();

Private void hi(){

System.out.println(“hi”);

}

}

Public class abc implements welcome{

Public void hello(){

System.out.println(“this is abc class…”);

}

Public static void main(String [] args){

Abc efg = new abc();

Efg.hello();

Efg.say();

}

Output:

This is abc clas…

Hello

hi

If the method is private they can’t be abstract in interface class.

Private methods in interface are used only inside the interface.

You can see above example, we used hi function in default method of interface.

**4.Encapsulation:-**

Encapsulation in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit. In encapsulation, the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class. Therefore, it is also known as **data hiding**.

To achieve encapsulation in Java −

* Declare the variables of a class as private.
* Provide public setter and getter methods to modify and view the variables values

**Package:**

Java package is a mechanism for organising java classes into namespace.

Classes in the same package can access each other’s members.

Java pre-defined packages:

Java.lang:

String,objects,thread,exception,system etc..

Java.util:

Also called as utility classes and very frequently used in code.

Java.io:

fileinputStream, fileoutputStream, fileReader, filewriter, RandomAccessfile etc…

java.net:

url,serversocket,socket etc…

javaApplet:

application which is used to load in browser.

Java.awt:

Event driven , user applications and event triggered when button presses and handles etc..

Java.awt.event:

Mouselisteners, eventlisteners etc..

Java.sql:

Database related stuff.

**Exception:**

Exception is a an event that occurs during the execution of the program, that disrupts the normal flow of the program.

It is referred as runtime error.

Checked exception:

Also called as compile time exception ex:

Try catch method.

Unchecked exception:

Also called as runtime exception.

Exception handling is done to execute program after getting a exception.

Try{

}catch{

}

**File:**

We can create file using

File abc = new File(path to file);

To write contents into file we can do using printwriter class

Try{

PrintWriter write = new PrintWriter(file name);

Write.println(“Akshay”);// it writes Akshay in to file

}catch(FileFoundException e){

//handle exception…

}

To get contents of file we can use scanner class

Try{

Scanner scan = new Scanner(file name);// if it is unable to open file it should handle the exception..

While(scan.hasnext()){

System.out.println(“content:”+scan.nextline());

}

}catch(FileFoundException e ){

// you have to handle exception….

}finally{

// here you can close the file

// this is called however.

Scan.close();

}

}

File output Streams:

We can use FileOutputStream filestream = new FileOutputtream(file name);

Filestream.write(“Akshay”.getBytes());

FileinputStream:

It is used to read contents from file.

FileInputStream input = new FileInputStream(file name);

While(char c = Input.read()!=-1)

{

System.out.println( c );

}

Input.close();

It reads in the form of characters.

**Android studio**

**UI interface**

**Layouts:**

A layout defines the structure for a user interface in your app.

All elements in the layout are built using a hierarchy of [View](https://developer.android.com/reference/android/view/View) and [ViewGroup](https://developer.android.com/reference/android/view/ViewGroup) objects.

A [View](https://developer.android.com/reference/android/view/View) usually draws something the user can see and interact with. Whereas a [ViewGroup](https://developer.android.com/reference/android/view/ViewGroup) is an invisible container that defines the layout structure for [View](https://developer.android.com/reference/android/view/View) and other [ViewGroup](https://developer.android.com/reference/android/view/ViewGroup) objects.

You can declare a layout in two ways:

* **Declare UI elements in XML**. Android provides a straightforward XML vocabulary that corresponds to the View classes and subclasses, such as those for widgets and layouts.

You can also use Android Studio's [Layout Editor](https://developer.android.com/studio/write/layout-editor) to build your XML layout using a drag-and-drop interface.

* **Instantiate layout elements at runtime**. Your app can create View and ViewGroup objects (and manipulate their properties) programmatically.

Using XML files also makes it easy to provide different layouts for different screen sizes and orientations

 you can declare your app's default layouts in XML, and then modify the layout at runtime.

* ***wrap\_content*** tells your view to size itself to the dimensions required by its content.
* ***match\_parent*** tells your view to become as big as its parent view group will allow.

**Linear Layout:**

A layout that organizes its children into a single horizontal or vertical row. It creates a scrollbar if the length of the window exceeds the length of the screen.

**Relative Layout:**

Enables you to specify the location of child objects relative to each other (child A to the left of child B) or to the parent (aligned to the top of the parent).

**Constrain Layout:**

It's similar to [RelativeLayout](https://developer.android.com/reference/android/widget/RelativeLayout) in that all views are laid out according to relationships between sibling views and the parent layout, but it's more flexible than RelativeLayout and easier to use with Android Studio's Layout Editor.

All the power of ConstraintLayout is available directly from the Layout Editor's visual tools, because the layout API and the Layout Editor were specially built for each other. So you can build your layout with ConstraintLayout entirely by drag-and-dropping instead of editing the XML.

Building Layouts with an Adapter

When the content for your layout is dynamic or not pre-determined, you can use a layout that subclasses [AdapterView](https://developer.android.com/reference/android/widget/AdapterView) to populate the layout with views at runtime. A subclass of the [AdapterView](https://developer.android.com/reference/android/widget/AdapterView) class uses an [Adapter](https://developer.android.com/reference/android/widget/Adapter) to bind data to its layout. The [Adapter](https://developer.android.com/reference/android/widget/Adapter) behaves as a middleman between the data source and the [AdapterView](https://developer.android.com/reference/android/widget/AdapterView) layout—the [Adapter](https://developer.android.com/reference/android/widget/Adapter) retrieves the data (from a source such as an array or a database query) and converts each entry into a view that can be added into the [AdapterView](https://developer.android.com/reference/android/widget/AdapterView) layout.

Common adapterviews are

**List View**

**Grid View**

You can populate an [AdapterView](https://developer.android.com/reference/android/widget/AdapterView) such as [ListView](https://developer.android.com/reference/android/widget/ListView) or [GridView](https://developer.android.com/reference/android/widget/GridView) by binding the [AdapterView](https://developer.android.com/reference/android/widget/AdapterView) instance to an [Adapter](https://developer.android.com/reference/android/widget/Adapter)

Array Adapter ,Use this adapter when your data source is an array.

ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,  
        android.R.layout.simple\_list\_item\_1);

By this we can add strings in list view as follows

adapter.add(“akshay kumar”);

Now we are adding the string data to the list view by using setAdapter().

ListView listView = (ListView) findViewById(R.id.listview);  
listView.setAdapter(adapter);

**Now if we want to do something if they click akshay kumar which shows on screen ,we have to implement onItemClickListener to our activity.**

**Then we have to setItem click listener as below**

**listView.setOnItemClickListener(this);**

**and also we will be having a method called**

**public void onItemClick(AdapterView<?> adapterView, View view, int i, long l){**

**String item = adapter.getItem(i); //it gets the item on position i**

**Toast.maketext(this,”you selected: “+item,toast.LENGTH\_LONG).show();**

**} //where i is the position**

**Add a floating action button:-**

floating action button (FAB) is a circular button that triggers the primary action in your app's UI. This page shows you how to add the FAB to your layout, customize some of its appearance, and respond to button taps.

Floating action buttons provide quick access to important or common actions within an app. They have a variety of uses, including:

* Performing a common action, such as starting a new email in a mail app.
* Displaying additional related actions.
* Update or transforming into other UI elements on the screen.

## Add the floating action button to your layout

The following code shows how the [FloatingActionButton](https://developer.android.com/reference/com/google/android/material/floatingactionbutton/FloatingActionButton) should appear in your layout file:

<android.support.design.widget.FloatingActionButton  
        android:id="@+id/fab"  
        android:layout\_width="wrap\_content"  
        android:layout\_height="wrap\_content"  
        android:layout\_gravity="end|bottom"  
        android:src="@drawable/ic\_my\_icon"  
        android:contentDescription="@string/submit"  
        android:layout\_margin="16dp" />

* The size of the FAB, using the app:fabSize attribute or the [setSize()](https://developer.android.com/reference/com/google/android/material/floatingactionbutton/FloatingActionButton#setSize(int)) method.
* The ripple color of the FAB, using the app:rippleColor attribute or the [setRippleColor()](https://developer.android.com/reference/com/google/android/material/floatingactionbutton/FloatingActionButton#setRippleColor(int)) method.
* The FAB icon, using the android:src attribute or the [setImageDrawable()](https://developer.android.com/reference/android/widget/ImageView#setImageDrawable(android.graphics.drawable.Drawable)) method.
* You can then apply an [View.OnClickListener](https://developer.android.com/reference/android/view/View.OnClickListener) to handle FAB taps. For example, the following code displays a [Snackbar](https://developer.android.com/reference/com/google/android/material/snackbar/Snackbar) when the user taps the FAB:

FloatingActionButton fab = findViewById(R.id.fab);  
fab.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View view) {  
        Snackbar.make(view, "Here's a Snackbar", Snackbar.LENGTH\_LONG)  
                .setAction("Action", null).show();  
    }  
});

# Create Shadows

Material design introduces elevation for UI elements. Elevation helps users understand the relative importance of each element and focus their attention to the task at hand.

The elevation of a view, represented by the Z property, determines the visual appearance of its shadow: views with higher Z values cast larger, softer shadows. Views with higher Z values occlude views with lower Z values; however, the Z value of a view does not affect the view's size.

**We can elevate the view in layouts by adding android:elevation , and mention elevation in dp .**

**For example we creating a view with elevation and background , the background represents the shadow of view after elevating.**

<TextView  
    android:id="@+id/myview"  
    ...  
    android:elevation="2dp"  
    android:background="@drawable/myrect" />

<!-- res/drawable/myrect.xml -->  
<shape xmlns:android="http://schemas.android.com/apk/res/android"  
       android:shape="rectangle">  
    <solid android:color="#42000000" />  
    <corners android:radius="5dp" />  
</shape>

we can set elevation of view in activity by using view.setElevation() method.

**Add application Widgets to home screen:-**

App Widgets are miniature application views that can be embedded in other applications (such as the Home screen) and receive periodic updates. These views are referred to as Widgets in the user interface, and you can publish one with an App Widget provider.

A widget is a small gadget or control of your android application placed on the home screen. Widgets can be very handy as they allow you to put your favourite applications on your home screen in order to quickly access them. You have probably seen some common widgets, such as music widget, weather widget, clock widget e.t.c

**Steps followed to create app widget:-**

In android studio goto java files and right click on it.

Select new option then it shows all options , select Widget option then select app widget option.

Then it asks for name of activity, fill appropriate details and select configure button.

Then android studio adds files in java files as well as xml file in layouts directory.

**package** com.example.myapplication;  
  
**import** android.app.PendingIntent;  
**import** android.appwidget.AppWidgetManager;  
**import** android.appwidget.AppWidgetProvider;  
**import** android.content.Context;  
**import** android.content.Intent;  
**import** android.widget.RemoteViews;  
  
*/\*\*  
 \* Implementation of App Widget functionality.  
 \* App Widget Configuration implemented in {****@link*** *NewAppWidgetConfigureActivity NewAppWidgetConfigureActivity}  
 \*/***public class** NewAppWidget **extends** AppWidgetProvider {  
  
  
  
 @Override  
 **public void** onUpdate(Context context, AppWidgetManager appWidgetManager, **int**[] appWidgetIds) {  
 *// There may be multiple widgets active, so update all of them* **for** (**int** appWidgetId : appWidgetIds) {  
 *// updateAppWidget(context, appWidgetManager, appWidgetId);* Intent intent = **new** Intent(context, MainActivity.**class**);  
 PendingIntent pendingIntent = PendingIntent.*getActivity*(context,0, intent,0);  
 RemoteViews views =**new** RemoteViews(context.getPackageName(),R.layout.***new\_app\_widget***);  
 views.setOnClickPendingIntent(R.id.***appwidget\_text***,pendingIntent);  
 appWidgetManager.updateAppWidget(appWidgetId,views);  
 }  
 }  
  
 @Override  
 **public void** onDeleted(Context context, **int**[] appWidgetIds) {  
 *// When the user deletes the widget, delete the preference associated with it.* **for** (**int** appWidgetId : appWidgetIds) {  
 NewAppWidgetConfigureActivity.*deleteTitlePref*(context, appWidgetId);  
 }  
 }  
  
 @Override  
 **public void** onEnabled(Context context) {  
 *// Enter relevant functionality for when the first widget is created* }  
  
 @Override  
 **public void** onDisabled(Context context) {  
 *// Enter relevant functionality for when the last widget is disabled* }  
}

above code is added in java directory to launch our main activity when our application widget os pressed on home screen.

Now new\_app\_widget xml file:

<**RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:background="#09C"  
 android:padding="@dimen/widget\_margin"**>  
  
 <**TextView  
 android:id="@+id/appwidget\_text"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:layout\_centerHorizontal="true"  
 android:layout\_centerVertical="true"  
 android:layout\_margin="8dp"  
 android:background="#09C"  
 android:contentDescription="@string/appwidget\_text"  
 android:text="@string/appwidget\_text"  
 android:textColor="#ffffff"  
 android:textSize="24sp"  
 android:textStyle="bold|italic"** />  
  
</**RelativeLayout**>

And in manifest file we have to add code as follows:

*<?***xml version="1.0" encoding="utf-8"***?>*<**manifest xmlns:android="http://schemas.android.com/apk/res/android"  
 package="com.example.myapplication"**>  
  
 <**application  
 android:allowBackup="true"  
 android:icon="@mipmap/ic\_launcher"  
 android:label="@string/app\_name"  
 android:roundIcon="@mipmap/ic\_launcher\_round"  
 android:supportsRtl="true"  
 android:theme="@style/AppTheme"**>  
 <**receiver android:name=".NewAppWidget"**>  
 <**intent-filter**>  
 <**action android:name="android.appwidget.action.APPWIDGET\_UPDATE"** />  
 </**intent-filter**>  
  
 <**meta-data  
 android:name="android.appwidget.provider"  
 android:resource="@xml/new\_app\_widget\_info"** />  
 </**receiver**>  
  
 <**activity android:name=".NewAppWidgetConfigureActivity"**>  
 <**intent-filter**>  
 <**action android:name="android.appwidget.action.APPWIDGET\_CONFIGURE"** />  
 </**intent-filter**>  
 </**activity**>  
 <**activity android:name=".MainActivity"**>  
 <**intent-filter**>  
 <**action android:name="android.intent.action.MAIN"** />  
  
 <**category android:name="android.intent.category.LAUNCHER"** />  
 </**intent-filter**>  
 </**activity**>  
 </**application**>  
  
</**manifest**>

**App – shortcuts:-**

Shortcuts deliver specific types of content to your users by helping them quickly access parts of your app.

App Shortcuts allow the user to access primary actions in your app straight from the launcher, taking the user deep into your application, by long pressing on your app icon.

Users can also pin these shortcuts to the home screen for even quicker access to your app’s primary actions.

Now we see how we have to create shortcuts for our application.

First create a xml file for shortcuts a shown below

Shortcuts.xml:-

*<?***xml version="1.0" encoding="utf-8"***?>*<**shortcuts xmlns:android="http://schemas.android.com/apk/res/android"**>  
 <**shortcut  
 android:shortcutId="akshay"  
 android:enabled="true"  
 android:icon="@mipmap/ic\_akshay"  
 android:shortcutShortLabel="@string/akshay"  
 android:shortcutLongLabel="@string/akshay"**>  
 <**intent  
 android:action="action.intent.action.VIEW"  
 android:targetPackage="com.example.myapplication3"  
 android:targetClass="com.example.myapplication3.MainActivity"**>  
 </**intent**>  
 </**shortcut**>  
</**shortcuts**>

In this when e press shortcut then we launching main activity, we also can launch any activity as of our requirement.

Secondly we have to add our xml file in manifest file as shown below

*<?***xml version="1.0" encoding="utf-8"***?>*<**manifest xmlns:android="http://schemas.android.com/apk/res/android"  
 package="com.example.myapplication3"**>  
  
 <**application  
 android:allowBackup="true"  
 android:icon="@mipmap/ic\_launcher"  
 android:label="@string/app\_name"  
 android:roundIcon="@mipmap/ic\_launcher\_round"  
 android:supportsRtl="true"  
 android:theme="@style/AppTheme"**>  
 <**activity android:name=".MainActivity"**>  
 <**intent-filter**>  
 <**action android:name="android.intent.action.MAIN"** />  
  
 <**category android:name="android.intent.category.LAUNCHER"** />  
 </**intent-filter**>  
 <**meta-data android:name="android.app.shortcuts"  
 android:resource="@xml/shortcuts"** />  
 </**activity**>  
 </**application**>  
  
</**manifest**>

**Menus:-**

In android, Menu is an important part of UI component which is used to provide some common functionality around the application. With the help of menu user can experience smooth and consistent experience throughout the application.

Menus are a common user interface component in many types of applications. To provide a familiar and consistent user experience, you should use the [Menu](https://developer.android.com/reference/android/view/Menu) APIs to present user actions and other options in your activities.

There are two types of menu they are:

Options menu

Popup menu

**1.options Menu:-**

Options menu is used in many applications for settings and other methods .

Options menu is used for user convinent to select the operation user needed from options.

**Creating options Menu:-**

Firstly to create options Menu we need Xml file of Menu so create a resource file by right clicking on resources directory, then select new then new resource file .

Now select resource type as menu. And create a xml fle

Add some items in options menu as shown below.

*<?***xml version="1.0" encoding="utf-8"***?>*<**menu xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:app="http://schemas.android.com/apk/res-auto"**>  
 <**item android:id="@+id/settings"  
 android:title="settings"  
 android:icon="@mipmap/ic\_akshay"  
 app:showAsAction="never"**>  
  
</**item**>  
 <**item android:id="@+id/help"  
 android:title="help"  
 android:icon="@drawable/ic\_akshay1"  
 app:showAsAction="never"**>  
  
 </**item**>  
</**menu**>

In this we added two items settings and help.

Now we have to override the onCreateOptionsMenu() method as shown below.

@Override  
**public boolean** onCreateOptionsMenu(Menu menu) {  
 MenuInflater inflater = getMenuInflater();  
 inflater.inflate(R.menu.***main\_menu***,menu);  
 **return true**;  
}

From above code when ever we select options menu then we can see two options settings and help.

To do something whenever we click settings and help items we have to override onOptionsItemSelected(MenuItem item) method.

As shown below

@Override  
**public boolean** onOptionsItemSelected(@NonNull MenuItem item) {  
 **switch** (item.getItemId())  
 {  
 **case** R.id.***settings***:  
 Toast.*makeText*(**this** ,**"settings selected"**,Toast.***LENGTH\_LONG***).show();  
 **return true**;  
  
 **case** R.id.***help***:  
 Toast.*makeText*(**this** ,**"help icon selected"**,Toast.***LENGTH\_LONG***).show();  
 **return true**;  
 **default**:  
 **return super** .onOptionsItemSelected(item);  
 }

We just displayed toast messages after clicking we can do any thing after clicking.

**Popup options:-**

Whenever we need to display options when we click button or any view on screen then we should use Popup options.

**Creating Popup options:-**

To create Popup options we have to create menu xml file as shown above.

Then we have to create a button in layout file them implement on click method for that options s shown below.

<**Button  
 android:id="@+id/popup\_button"  
 android:layout\_width="wrap\_content"  
 android:layout\_height="wrap\_content"  
 android:gravity="center"  
 android:onClick="popup"  
 android:text="popup"  
 app:layout\_constraintBottom\_toBottomOf="parent"  
  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent"  
 app:layout\_constraintVertical\_bias="0.679"  
 tools:ignore="MissingConstraints"**>

</Button>

Now we have to create popup function as shown below

**public void** popup(View view) {  
 PopupMenu  
popup = **new** PopupMenu(**this**,view);  
 popup.setOnMenuItemClickListener(**this**);  
 popup.inflate(R.menu.***main\_menu***);  
 popup.show();  
 }

by this when you click button then items on menu xml file can be seen.

Now we need some work when we click items in popup options then we have to override onMenuItemClick(MenuItem item) as shown below

@Override  
**public boolean** onMenuItemClick(MenuItem item) {  
  
 **switch** (item.getItemId())  
 {  
 **case** R.id.***settings***:  
 Toast.*makeText*(**this** ,**"pop up settings selected"**,Toast.***LENGTH\_LONG***).show();  
 **return true**;  
  
 **case** R.id.***help***:  
 Toast.*makeText*(**this** ,**"pop up help icon selected"**,Toast.***LENGTH\_LONG***).show();  
 **return true**;  
 **default**:  
 **return super** .onOptionsItemSelected(item);  
 }

In this when we click we just get toast message .

**Android: Settings:-**

In android we have a layout for settings screen called as Preference Screen.

It mainly used in all applications for changing functionalities of applications.

To add this settings layout in application and comes into use when we press settings option in menu.

**Create preference layout:-**

Firstly right click on the resource directory and select new resource file

Then give any name to the file and select **resource type** as preferences, and directory type as xml then hit ok button.

We get a layout xml file with preferenceScreen layout

Now we need to add some preferences in this layouts as shown below.

*<?***xml version="1.0" encoding="utf-8"***?>*<**PreferenceScreen xmlns:android="http://schemas.android.com/apk/res/android"**>  
 <**CheckBoxPreference  
 android:key="enable"  
 android:title="first option"  
 android:summary="@string/akshay"  
 android:defaultValue="true"  
 android:onClick="akshay"** />  
 <**Preference  
 android:dependency="enable"  
 android:key="option2"  
 android:title="second option"**/>  
  
</**PreferenceScreen**>

There are different types preferences as shown above

checkbox preference is used to enable or disable usages in application.

Every preference id differentiated by using Key.

Now create activity which is created when you pressed settings option in menu as shown below

**package** com.example.myapplication;  
  
**import** androidx.appcompat.app.AppCompatActivity;  
  
**import** android.os.Bundle;  
**import** android.preference.Preference;  
**import** android.preference.PreferenceFragment;  
**import** android.util.Log;  
**import** android.widget.Toast;  
  
**import** java.util.prefs.PreferenceChangeEvent;  
  
**public class** settings **extends** AppCompatActivity  
{  
 fragment **settingsFragment**;  
 @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***activity\_settings***);  
 getSupportActionBar().setTitle(**"Settings"**);  
 **if**(findViewById(R.id.***preference\_frame***)!= **null**)  
 {  
 **settingsFragment** = **new** fragment();  
 getFragmentManager().beginTransaction().add(R.id.***preference\_frame***,**settingsFragment**).commit();  
 }  
 }  
  
 */\* @Override  
 protected void onResume() {  
 super.onResume();  
 Preference preference = settingsFragment.findPreference("enable");  
 preference.setOnPreferenceClickListener(new Preference.OnPreferenceClickListener() {  
 @Override  
 public boolean onPreferenceClick(Preference preference) {  
 Toast.makeText(getApplicationContext(),"enable",Toast.LENGTH\_LONG).show();  
  
 return true;  
 }  
 });  
 }\*/*}

now we need a fragment class which extends with preference fragment in this we use our preference layout and this fragment is added in our activity as shown above.

**package** com.example.myapplication;  
  
**import** android.os.Bundle;  
**import** android.preference.Preference;  
**import** android.preference.PreferenceFragment;  
**import** android.widget.Toast;  
  
**import** androidx.annotation.Nullable;  
  
**public class** fragment **extends** PreferenceFragment {  
 @Override  
 **public void** onCreate(@Nullable Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 addPreferencesFromResource(R.xml.***preferences***);//adding our preference layout in fragment.  
 }  
 @Override  
 **public void** onResume() {  
 **super**.onResume();  
 Preference preference = findPreference(**"enable"**);  
 preference.setOnPreferenceClickListener(**new** Preference.OnPreferenceClickListener() {  
 @Override  
 **public boolean** onPreferenceClick(Preference preference) {  
 Toast.*makeText*(getActivity(),**"selected enabled"**,Toast.***LENGTH\_LONG***).show();  
  
 **return true**;  
 }  
 });  
 }  
}

now we add this fragment in activity layout as follows

**activity\_settings.xml**

*<?***xml version="1.0" encoding="utf-8"***?>*<**androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:app="http://schemas.android.com/apk/res-auto"  
 xmlns:tools="http://schemas.android.com/tools"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 tools:context=".settings"**>  
 <**FrameLayout  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:id="@+id/preference\_frame"**/>  
</**androidx.constraintlayout.widget.ConstraintLayout**>

**Working:-**

Now when we goto menu option and select settings option then oncreate method of

settings class is called in which we use setContentView(R.layout.***activity\_settings***);

then it launches activity\_settings layout

when we added fragment to our layout by using command

**settingsFragment** = **new** fragment();  
 getFragmentManager().beginTransaction().add(R.id.***preference\_frame***,**settingsFragment**).commit();

By this command the on create method of fragment class is called.

In oncreate method we useaddPreferencesFromResource(R.xml.***preferences***);

So the preference layout is launched on our activity.

We can also create on click events of that preferences by getting preferences using their keys .

We can setsetOnPreferenceClickListener for every preferences and do action as required for the application.

**Adding the App Bar:-**

The app bar, also known as the action bar, is one of the most important design elements in your app's activities, because it provides a visual structure and interactive elements that are familiar to users. Using the app bar makes your app consistent with other Android apps, allowing users to quickly understand how to operate your app and have a great experience.

**Setup an App Bar:-**

We have to make sure that our activity should extends with AppcompatActivity as shown below.

public class MyActivity extends AppCompatActivity {  
  // ...  
}

In the app manifest, set the [<application>](https://developer.android.com/guide/topics/manifest/application-element) element to use one of appcompat's [NoActionBar](https://developer.android.com/reference/androidx/appcompat/R.style#Theme_AppCompat_NoActionBar) themes. Using one of these themes prevents the app from using the native [ActionBar](https://developer.android.com/reference/android/app/ActionBar) class to provide the app bar. For example:

<application  
    android:theme="@style/Theme.AppCompat.Light.NoActionBar"  
    />

Add a [Toolbar](https://developer.android.com/reference/androidx/appcompat/widget/Toolbar) to the activity's layout. For example, the following layout code adds a [Toolbar](https://developer.android.com/reference/androidx/appcompat/widget/Toolbar) and gives it the appearance of floating above the activity:

<android.support.v7.widget.Toolbar  
   android:id="@+id/my\_toolbar"  
   android:layout\_width="match\_parent"  
   android:layout\_height="?attr/actionBarSize"  
   android:background="?attr/colorPrimary"  
   android:elevation="4dp"  
   android:theme="@style/ThemeOverlay.AppCompat.ActionBar"  
   app:popupTheme="@style/ThemeOverlay.AppCompat.Light"/>

In the activity's [onCreate()](https://developer.android.com/reference/android/app/Activity#onCreate(android.os.Bundle)) method, call the activity's [setSupportActionBar()](https://developer.android.com/reference/androidx/appcompat/app/AppCompatActivity#setSupportActionBar(android.support.v7.widget.Toolbar)) method, and pass the activity's toolbar. This method sets the toolbar as the app bar for the activity.

For example:

@Override  
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity\_my);  
    **Toolbar myToolbar = (Toolbar) findViewById(R.id.my\_toolbar);  
    setSupportActionBar(myToolbar);**

}

**Adding Action Buttons On Action Bar:-**

The app bar allows you to add buttons for user actions. This feature lets you put the most important actions for the current context right at the top of the app.

Note:- to implement see Menu.

**Add an up Action:-**

Your app should make it easy for users to find their way back to the app's main screen. One simple way to do this is to provide an Up button on the app bar for all activities except the main one. When the user selects the Up button, the app navigates to the parent activity.

To support the up functionality in an activity, you need to declare the activity's parent. You can do this in the app manifest, by setting an android:parentActivityName attribute. As shown below.

application ... >  
    ...  
  
    <!-- The main/home activity (it has no parent activity) -->  
  
    <activity  
        android:name="com.example.myfirstapp.MainActivity" ...>  
        ...  
    </activity>  
  
    <!-- A child of the main activity -->  
    <activity  
        android:name="com.example.myfirstapp.MyChildActivity"  
        android:label="@string/title\_activity\_child"  
        **android:parentActivityName="com.example.myfirstapp.MainActivity"** >  
  
        <!-- Parent activity meta-data to support 4.0 and lower -->  
        <meta-data  
            **android:name="android.support.PARENT\_ACTIVITY"  
            android:value="com.example.myfirstapp.MainActivity" />**

    </activity>  
</application>

**Activity:-**

An activity represents a single screen with a user interface just like window or frame of Java.

Android system initiates its program with in an **Activity** starting with a call on *onCreate()* callback method.

**Activity lifecycle:-**

**onCreate()**

This is the first callback and called when the activity is first created.

**onStart()**

This callback is called when the activity becomes visible to the user.

**onResume()**

This is called when the user starts interacting with the application.

**onPause()**

The paused activity does not receive user input and cannot execute any code and called when the current activity is being paused and the previous activity is being resumed.

When the user taps the Back or Recents button. When the system calls [onPause()](https://developer.android.com/reference/android/app/Activity#onPause()) for your activity, it technically means your activity is still partially visible, but most often is an indication that the user is leaving the activity, and the activity will soon enter the Stopped or Resumed state.

**onStop()**

This callback is called when the activity is no longer visible.

**onDestroy()**

This callback is called before the activity is destroyed by the system.

**onRestart()**

This callback is called when the activity restarts after stopping it.

**Fragment:-**

A [Fragment](https://developer.android.com/reference/androidx/fragment/app/Fragment) represents a behavior or a portion of user interface.

We can think of a fragment as a modular section of an activity, which has its own lifecycle, receives its own input events, and which you can add or remove while the activity is running.

Life Cycle Of Fragment:-

There are three major callback functions used in fragments they are

[onCreate()](https://developer.android.com/reference/androidx/fragment/app/Fragment#onCreate(android.os.Bundle))

The system calls this when creating the fragment. Within your implementation, you should initialize essential components of the fragment that you want to retain when the fragment is paused or stopped, then resumed.

[onCreateView()](https://developer.android.com/reference/androidx/fragment/app/Fragment#onCreateView(android.view.LayoutInflater,%20android.view.ViewGroup,%20android.os.Bundle))

The system calls this when it's time for the fragment to draw its user interface for the first time. To draw a UI for your fragment, you must return a [View](https://developer.android.com/reference/android/view/View) from this method that is the root of your fragment's layout. You can return null if the fragment does not provide a UI.

[onPause()](https://developer.android.com/reference/androidx/fragment/app/Fragment#onPause())

The system calls this method as the first indication that the user is leaving the fragment (though it doesn't always mean the fragment is being destroyed). This is usually where you should commit any changes that should be persisted beyond the current user session (because the user might not come back).

**Creating a Fragment:-**

To create Fragment we need to create a subclass of Fragment class. The subclass looks similar to activity.it contain callbacks similar to Activity.

Suppose we need to load a example\_fragment .xml layout in our activity

We need to write a java class for example ExampleFragment.java

public static class ExampleFragment extends Fragment {  
    @Override  
    public View onCreateView(LayoutInflater inflater, ViewGroup container,  
                             Bundle savedInstanceState) {  
        // Inflate the layout for this fragment  
        return inflater.inflate(R.layout.example\_fragment, container, false);  
    }  
}

when ever instance is created it loads example\_fragment.xml.

we need to declare a fragment inside a layout on which fragment is loaded.

<?xml version="1.0" encoding="utf-8"?>  
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:orientation="horizontal"  
    android:layout\_width="match\_parent"  
    android:layout\_height="match\_parent">  
    <fragment android:name="com.example.news.ArticleListFragment"  
            android:id="@+id/list"  
            android:layout\_weight="1"  
            android:layout\_width="0dp"  
            android:layout\_height="match\_parent" />  
      
</LinearLayout>

To make a fragment trasaction in our activity we must need to use FragmentTransaction API

FragmentManager fragmentManager = getSupportFragmentManager();  
FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();

You can then add a fragment using the [add()](https://developer.android.com/reference/androidx/fragment/app/FragmentTransaction#add(int,%20android.support.v4.app.Fragment)) method, specifying the fragment to add and the view in which to insert it. For example:

ExampleFragment fragment = new ExampleFragment();  
fragmentTransaction.add(R.id.list, fragment);  
fragmentTransaction.commit();

Once you've made your changes with [FragmentTransaction](https://developer.android.com/reference/androidx/fragment/app/FragmentTransaction), you must call [commit()](https://developer.android.com/reference/androidx/fragment/app/FragmentTransaction#commit()) for the changes to take effect.

The lifecycle of the activity in which the fragment lives directly affects the lifecycle of the fragment, such that each lifecycle callback for the activity results in a similar callback for each fragment. For example, when the activity receives [onPause()](https://developer.android.com/reference/android/app/Activity#onPause()), each fragment in the activity receives [onPause()](https://developer.android.com/reference/androidx/fragment/app/Fragment#onPause()).

**Interact with other apps**:

An Android app typically has several [activities](https://developer.android.com/guide/components/activities). Each activity displays a user interface that allows the user to perform a specific task (such as view a map or take a photo).

To take the user from one activity to another, your app must use an [Intent](https://developer.android.com/reference/android/content/Intent) to define your app's "intent" to do something.

When you pass an [Intent](https://developer.android.com/reference/android/content/Intent) to the system with a method such as [startActivity()](https://developer.android.com/reference/android/app/Activity#startActivity(android.content.Intent)), the system uses the [Intent](https://developer.android.com/reference/android/content/Intent) to identify and start the appropriate app component. Using intents

This shows you how to create an implicit intent for a particular action, and how to use it to start an activity that performs the action in another app even allows your app to start an activity that is contained in a separate app.

For example, here's how to create an intent to initiate a phone call using the [Uri](https://developer.android.com/reference/android/net/Uri) data to specify the telephone number:

Uri number = Uri.parse("tel:5551234");  
Intent callIntent = new Intent(Intent.ACTION\_DIAL, number);

The  [startActivityForResult()](https://developer.android.com/reference/android/app/Activity#startActivityForResult(android.content.Intent,%20int)) and [onActivityResult()](https://developer.android.com/reference/android/app/Activity#onActivityResult(int,%20int,%20android.content.Intent)) APIs are available on the Activity class on all API levels, it is strongly recommended to use the Activity Result APIs When your app invokes this intent by calling [startActivity()](https://developer.android.com/reference/android/app/Activity#startActivity(android.content.Intent)), the Phone app initiates a call to the given phone number.

**Intents:-**

An [Intent](https://developer.android.com/reference/android/content/Intent) is a messaging object you can use to request an action from another [app component](https://developer.android.com/guide/components/fundamentals#Components). Although intents facilitate communication between components in several ways, there are three fundamental use cases:

* **Starting an activity**

An [Activity](https://developer.android.com/reference/android/app/Activity) represents a single screen in an app. You can start a new instance of an [Activity](https://developer.android.com/reference/android/app/Activity) by passing an [Intent](https://developer.android.com/reference/android/content/Intent) to [startActivity()](https://developer.android.com/reference/android/content/Context#startActivity(android.content.Intent)). The [Intent](https://developer.android.com/reference/android/content/Intent) describes the activity to start and carries any necessary data.

If you want to receive a result from the activity when it finishes, call [startActivityForResult()](https://developer.android.com/reference/android/app/Activity#startActivityForResult(android.content.Intent,%20int)). Your activity receives the result as a separate [Intent](https://developer.android.com/reference/android/content/Intent) object in your activity's [onActivityResult()](https://developer.android.com/reference/android/app/Activity#onActivityResult(int,%20int,%20android.content.Intent)) callback. For more information, see the [Activities](https://developer.android.com/guide/components/activities) guide.

* **Starting a service**

A [Service](https://developer.android.com/reference/android/app/Service) is a component that performs operations in the background without a user interface. With Android 5.0 (API level 21) and later, you can start a service with [JobScheduler](https://developer.android.com/reference/android/app/job/JobScheduler).

For versions earlier than Android 5.0 (API level 21), you can start a service by using methods of the [Service](https://developer.android.com/reference/android/app/Service) class. You can start a service to perform a one-time operation (such as downloading a file) by passing an [Intent](https://developer.android.com/reference/android/content/Intent) to [startService()](https://developer.android.com/reference/android/content/Context#startService(android.content.Intent)). The [Intent](https://developer.android.com/reference/android/content/Intent) describes the service to start and carries any necessary data.

If the service is designed with a client-server interface, you can bind to the service from another component by passing an [Intent](https://developer.android.com/reference/android/content/Intent) to [bindService()](https://developer.android.com/reference/android/content/Context#bindService(android.content.Intent,%20android.content.ServiceConnection,%20int)). For more information, see the [Services](https://developer.android.com/guide/components/services) guide.

* **Delivering a broadcast**

A broadcast is a message that any app can receive. The system delivers various broadcasts for system events, such as when the system boots up or the device starts charging. You can deliver a broadcast to other apps by passing an [Intent](https://developer.android.com/reference/android/content/Intent) to [sendBroadcast()](https://developer.android.com/reference/android/content/Context#sendBroadcast(android.content.Intent)) or [sendOrderedBroadcast()](https://developer.android.com/reference/android/content/Context#sendOrderedBroadcast(android.content.Intent,%20java.lang.String)).

**Intent Types:-**

There are two types of intents:

**Explicit intents:**

The intents which are used to switch from one activity to another activity, launch services,broadcasrt receivers are called explicit intents.

In this intents we supply fully qualified component class name.

Intent my\_intent = new intent(this.myservice.class);

Component name is essential because it shows the intent is explicit, it means intent is delivered to the only app component which is specified.

Myservice class should be added in manifest file.

**Implicit intents** :

In this intents we don’t supply class name but declare a general action to perform, which allows component from another class to handle it.

 For example, if you want to show the user a location on a map, you can use an implicit intent to request that another capable app show a specified location on a map.

Example for implicit intent

public void startTimer(String message, int seconds) {  
    Intent intent = new Intent(AlarmClock.ACTION\_SET\_TIMER)  
            .putExtra(AlarmClock.EXTRA\_MESSAGE, message)  
            .putExtra(AlarmClock.EXTRA\_LENGTH, seconds)  
            .putExtra(AlarmClock.EXTRA\_SKIP\_UI, true);  
    if (intent.resolveActivity(getPackageManager()) != null) {  
        startActivity(intent);  
    }  
}

we need to add below code in manifest file

<activity ...>  
    <intent-filter>  
        <action android:name="android.intent.action.SET\_TIMER" />  
        <category android:name="android.intent.category.DEFAULT" />  
    </intent-filter>  
</activity>

**Intent-filters:-**

 An intent filter declares the capabilities of its parent component — what an activity or service can do and what types of broadcasts a receiver can handle.

**BackgroungTasks:**

Every Android app has a main thread which is in charge of handling UI (including measuring and drawing views), coordinating user interactions, and receiving lifecycle events.

If there is too much work happening on this thread, the app appears to hang or slow down, leading to an undesirable user experience.

Any long-running computations and operations such as decoding a bitmap, accessing the disk, or performing network requests should be done on a separate background thread

**Sending operations to threads:-**

A thread is a lightweight sub-process, it going to do background operations without interrupt to ui.

The speed and efficiency of a long-running, data-intensive operation often improves when you split it into smaller operations running on multiple threads. On a device that has a CPU with multiple processors (cores), the system can run the threads in parallel, rather than making each sub-operation wait for a chance to run.

Example:

Thread t = new Thread(new Runnable() {

            @Override

            public void run() {

               runOnUiThread(new Runnable() {

                  @Override

                  public void run() {

                     ///write the code which you want to run on thread here.

                  }

               });

Now we have to start the thread to run the code which is in run method .

t.start();

whenever we use this command the thread we created starts running.

Handler:-

In android Handler is mainly used to update the main thread from background thread or other than main thread. There are two methods are in handler.

* **Post()** − it going to post message from background thread to main thread using looper.
* **sendmessage()** − if you want to organize what you have sent to ui (message from background thread) or ui functions. you should use sendMessage().

Example:

ProgressDialog mProgressBar;

Handler mHandler=new Handler();

new Thread(new Runnable() {

         @Override

         public void run() {

            for (int i = 0; i <= 100; i++) {

               final int currentProgressCount = i;

               try {

                  Thread.sleep(50);

               } catch (InterruptedException e) {

                  e.printStackTrace();

               }

               //Update the value background thread to UI thread

               mHandler.post(new Runnable() {

                  @Override

                  public void run() {

                      mProgressBar.setProgress(currentProgressCount);

                  }

               });

            }

         }

      }).start();

**Service:-**

A [Service](https://developer.android.com/reference/android/app/Service) is an application component that can perform long-running operations in the background, and it doesn't provide a user interface.

 Another application component can start a service, and it continues to run in the background even if the user switches to another application.

These are the three different types of services:

**Foreground**

A foreground service performs some operation that is noticeable to the user.

 Foreground services must display a [Notification](https://developer.android.com/guide/topics/ui/notifiers/notifications). Foreground services continue running even when the user isn't interacting with the app.

**Background**

A background service performs an operation that isn't directly noticed by the user.

**Bound**

A service is bound when an application component binds to it by calling [bindService()](https://developer.android.com/reference/android/content/Context#bindService(android.content.Intent,%20android.content.ServiceConnection,%20int)). A bound service offers a client-server interface that allows components to interact with the service, send requests, receive results, and even do so across processes with interprocess communication (IPC). A bound service runs only as long as another application component is bound to it. Multiple components can bind to the service at once, but when all of them unbind, the service is destroyed.

**Choosing between Service and Thread:-**

A service is simply a component that can run in the background, even when the user is not interacting with your application.

If you must perform work outside of your main thread, but only while the user is interacting with your application, you should instead create a new thread.

For example, if you want to play some music, but only while your activity is running, you might create a thread in [onCreate()](https://developer.android.com/reference/android/app/Activity#onCreate(android.os.Bundle)), start running it in [onStart()](https://developer.android.com/reference/android/app/Activity#onStart()), and stop it in [onStop()](https://developer.android.com/reference/android/app/Activity#onStop()).

To create a service, you must create a subclass of [Service](https://developer.android.com/reference/android/app/Service).

you must override some callback methods that handle key aspects of the service lifecycle.

[onStartCommand()](https://developer.android.com/reference/android/app/Service#onStartCommand(android.content.Intent,%20int,%20int))

The system invokes this method by calling [startService()](https://developer.android.com/reference/android/content/Context#startService(android.content.Intent)) when another component (such as an activity) requests that the service be started. When this method executes, the service is started and can run in the background indefinitely. If you implement this, it is your responsibility to stop the service when its work is complete by calling [stopSelf()](https://developer.android.com/reference/android/app/Service#stopSelf()) or [stopService()](https://developer.android.com/reference/android/content/Context#stopService(android.content.Intent)).

[onBind()](https://developer.android.com/reference/android/app/Service#onBind(android.content.Intent))

The system invokes this method by calling [bindService()](https://developer.android.com/reference/android/content/Context#bindService(android.content.Intent,%20android.content.ServiceConnection,%20int)) when another component wants to bind with the service

[onCreate()](https://developer.android.com/reference/android/app/Service#onCreate())

The system invokes this method to perform one-time setup procedures when the service is initially created (before it calls either [onStartCommand()](https://developer.android.com/reference/android/app/Service#onStartCommand(android.content.Intent,%20int,%20int)) or [onBind()](https://developer.android.com/reference/android/app/Service#onBind(android.content.Intent))). If the service is already running, this method is not called.

[onDestroy()](https://developer.android.com/reference/android/app/Service#onDestroy())

The system invokes this method when the service is no longer used and is being destroyed.

If a component starts the service by calling [startService()](https://developer.android.com/reference/android/content/Context#startService(android.content.Intent)) (which results in a call to [onStartCommand()](https://developer.android.com/reference/android/app/Service#onStartCommand(android.content.Intent,%20int,%20int))), the service continues to run until it stops itself with [stopSelf()](https://developer.android.com/reference/android/app/Service#stopSelf()) or another component stops it by calling [stopService()](https://developer.android.com/reference/android/content/Context#stopService(android.content.Intent)).

If a component calls [bindService()](https://developer.android.com/reference/android/content/Context#bindService(android.content.Intent,%20android.content.ServiceConnection,%20int)) to create the service and [onStartCommand()](https://developer.android.com/reference/android/app/Service#onStartCommand(android.content.Intent,%20int,%20int)) is not called, the service runs only as long as the component is bound to it. After the service is unbound from all of its clients, the system destroys it.

**Declaring service in manifest:-**

You must declare all services in your application's manifest file, just as you do for activities and other components.

To declare your service, add a [<service>](https://developer.android.com/guide/topics/manifest/service-element) element as a child of the [<application>](https://developer.android.com/guide/topics/manifest/application-element) element. Here is an example:

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

### Starting a service

You can start a service from an activity or other application component by passing an [Intent](https://developer.android.com/reference/android/content/Intent) to [startService()](https://developer.android.com/reference/android/content/Context#startService(android.content.Intent)) or [startForegroundService()](https://developer.android.com/reference/android/content/Context#startForegroundService(android.content.Intent))

For example we have a service class called **helloservice**

public class helloService extends Service {  
    int startMode;       // indicates how to behave if the service is killed  
    IBinder binder;      // interface for clients that bind  
    boolean allowRebind; // indicates whether onRebind should be used  
  
    @Override  
    public void [**onCreate**](https://developer.android.com/reference/android/app/Service#onCreate())() {  
        // The service is being created  
    }  
    @Override  
    public int [**onStartCommand**](https://developer.android.com/reference/android/app/Service#onStartCommand(android.content.Intent,%20int,%20int))(Intent intent, int flags, int startId) {  
        // The service is starting, due to a call to [**startService()**](https://developer.android.com/reference/android/content/Context#startService(android.content.Intent))  
        return *mStartMode*;  
    }  
    @Override  
    public IBinder [**onBind**](https://developer.android.com/reference/android/app/Service#onBind(android.content.Intent))(Intent intent) {  
        // A client is binding to the service with [**bindService()**](https://developer.android.com/reference/android/content/Context#bindService(android.content.Intent,%20android.content.ServiceConnection,%20int))  
        return *mBinder*;  
    }  
    @Override  
    public boolean [**onUnbind**](https://developer.android.com/reference/android/app/Service#onUnbind(android.content.Intent))(Intent intent) {  
        // All clients have unbound with [**unbindService()**](https://developer.android.com/reference/android/content/Context#unbindService(android.content.ServiceConnection))  
        return *mAllowRebind*;  
    }  
    @Override  
    public void [**onRebind**](https://developer.android.com/reference/android/app/Service#onRebind(android.content.Intent))(Intent intent) {  
        // A client is binding to the service with [**bindService()**](https://developer.android.com/reference/android/content/Context#bindService(android.content.Intent,%20android.content.ServiceConnection,%20int)),  
        // after onUnbind() has already been called  
    }  
    @Override  
    public void [**onDestroy**](https://developer.android.com/reference/android/app/Service#onDestroy())() {  
        // The service is no longer used and is being destroyed  
    }  
}

We have to start a service by using intent.

Intent intent = new Intent(this,helloservice.class);

Now we have to start service with this intent by using command **startService(intent);**

The [startService()](https://developer.android.com/reference/android/content/Context#startService(android.content.Intent)) method returns immediately, and the Android system calls the service's [onStartCommand()](https://developer.android.com/reference/android/app/Service#onStartCommand(android.content.Intent,%20int,%20int)) method. If the service isn't already running, the system first calls [onCreate()](https://developer.android.com/reference/android/app/Service#onCreate()), and then it calls [onStartCommand()](https://developer.android.com/reference/android/app/Service#onStartCommand(android.content.Intent,%20int,%20int)).

**Broadcast Receiver:-**

**Broadcast Receivers**simply respond to broadcast messages from other applications or from the system itself. These messages are sometime called events or intents.

Android apps can send or receive broadcast messages from the Android system and other Android apps.

It is used especially to allow application to respond system level broadcast announcements.

For example, the Android system sends broadcasts when various system events occur, such as when the system boots up or the device starts charging. Apps can also send custom broadcasts.

for example, to notify other apps of something that they might be interested in.

To receive messages we need to register broadcast receiver.

 When a broadcast is sent, the system automatically routes broadcasts to apps that have subscribed to receive that particular type of broadcast.

Suppose we are subscribed to receive an notification when battery level is changed then when ever battery level of device is changed the system broadcast that message to all other apps which are subscribed to that notification , then our on receive() function of broadcast receiver class is called .

There are two types of registering broadcast receiver for specific intent.

We need to declare a broadcast receiver in your manifest, the system launches your app.

<receiver android:name=".MyBroadcastReceiver"  android:exported="true">  
    <intent-filter>  
        <action android:name="android.intent.action.BATTERY\_CHANGED" />  
    </intent-filter>  
</receiver>

**Above**  I registered broadcast receiver class MyBroadcastReceiver for intent battery changed , so whenever battery changed onreceive method of this class is called.

A broadcast receiver is implemented as a subclass of **BroadcastReceiver** class and overriding the onReceive() method where each message is received as a **Intent** object parameter.

public class MyBroadcastReceiver extends BroadcastReceiver {

@Override

public void onReceive(Context context, Intent intent) {

Toast.makeText(context, "Intent Detected. Battery changed", Toast.LENGTH\_LONG).show();

}

}

### 2. Context-registered receivers

To register a receiver with a context, perform the following steps:

1. Create an instance of [BroadcastReceiver](https://developer.android.com/reference/android/content/BroadcastReceiver).

BroadcastReceiver br = new MyBroadcastReceiver();

1. Create an [IntentFilter](https://developer.android.com/reference/android/content/IntentFilter) and register the receiver by calling [registerReceiver(BroadcastReceiver, IntentFilter)](https://developer.android.com/reference/android/content/Context#registerReceiver(android.content.BroadcastReceiver,%20android.content.IntentFilter)):

IntentFilter filter = new IntentFilter(ConnectivityManager.CONNECTIVITY\_ACTION);  
    filter.addAction(Intent.ACTION\_AIRPLANE\_MODE\_CHANGED);  
    this.registerReceiver(br, filter);

It receives broadcasts as long as intent is valid.

To stop receiving broadcasts, call [unregisterReceiver(android.content.BroadcastReceiver)](https://developer.android.com/reference/android/content/Context#unregisterReceiver(android.content.BroadcastReceiver)). Be sure to unregister the receiver when you no longer need it or the context is no longer valid.

**Register Broadcast receiver with custom intent:-**

When an application want to broadcast its custom intent to other applications , then it is done as follows

Intent intent = new Intent();

intent.setAction("CUSTOM\_INTENT");

sendBroadcast(intent);

it broadcast the message to other applications , so if we need to react to that intents we need to register our broadcast receiver with that custom intent , then when ever we get broadcast with that intent our broadcast receiver class onReceive() method is called

**on receiver end**

. public class MyReceiver extends BroadcastReceiver {

@Override

public void onReceive(Context context, Intent intent) {

Toast.makeText(context, "Intent Detected.", Toast.LENGTH\_LONG).show();

}

}

This intent *CUSTOM\_INTENT* can also be registered in similar way as we have regsitered system generated intent.

<application

android:icon="@drawable/ic\_launcher"

android:label="@string/app\_name"

android:theme="@style/AppTheme" >

<receiver android:name="MyReceiver">

<intent-filter>

<action android:name="CUSTOM\_INTENT">

</action>

</intent-filter>

</receiver>

</application>

**Keep device awake:-**

Certain apps need to keep the screen turned on, such as games or movie apps. The best way to do this is to use the [FLAG\_KEEP\_SCREEN\_ON](https://developer.android.com/reference/android/view/WindowManager.LayoutParams#FLAG_KEEP_SCREEN_ON) in your activity (and only in an activity, never in a service or other app component). For example:

public class MainActivity extends Activity {  
  @Override  
  protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity\_main);  
    getWindow().addFlags(WindowManager.LayoutParams.FLAG\_KEEP\_SCREEN\_ON);  
  }  
}

**Sensors:-**

Most Android-powered devices have built-in sensors that measure motion, orientation, and various environmental conditions. These sensors are capable of providing raw data with high precision and accuracy.

For example, a game might track readings from a device's gravity sensor to infer complex user gestures and motions, such as tilt, shake, rotation, or swing. Likewise, a weather application might use a device's temperature sensor and humidity sensor to calculate and report the dewpoint, or a travel application might use the geomagnetic field sensor and accelerometer to report a compass bearing.

* **Motion sensors**

These sensors measure acceleration forces and rotational forces along three axes. This category includes accelerometers, gravity sensors, gyroscopes, and rotational vector sensors.

* **Environmental sensors**

These sensors measure various environmental parameters, such as ambient air temperature and pressure, illumination, and humidity. This category includes barometers, photometers, and thermometers.

* **Position sensors**

These sensors measure the physical position of a device. This category includes orientation sensors and magnetometers.

### Sensor Framework

You can access these sensors and acquire raw sensor data by using the Android sensor framework. The sensor framework is part of the [android.hardware](https://developer.android.com/reference/android/hardware/package-summary) package and includes the following classes and interfaces:

[SensorManager](https://developer.android.com/reference/android/hardware/SensorManager)

You can use this class to create an instance of the sensor service. This class provides various methods for accessing and listing sensors, registering and unregistering sensor event listeners, and acquiring orientation information. This class also provides several sensor constants that are used to report sensor accuracy, set data acquisition rates, and calibrate sensors.

[Sensor](https://developer.android.com/reference/android/hardware/Sensor)

You can use this class to create an instance of a specific sensor. This class provides various methods that let you determine a sensor's capabilities.

[SensorEvent](https://developer.android.com/reference/android/hardware/SensorEvent)

The system uses this class to create a sensor event object, which provides information about a sensor event.

A sensor event object includes the following information: the raw sensor data, the type of sensor that generated the event, the accuracy of the data, and the timestamp for the event.

[SensorEventListener](https://developer.android.com/reference/android/hardware/SensorEventListener)

You can use this interface to create two callback methods that receive notifications (sensor events) when sensor values change or when sensor accuracy changes.

In a typical application you use these sensor-related APIs to perform two basic tasks:

1. **Identifying sensors and sensor capabilities**
2. **Monitor sensor events**

## Identifying Sensors and Sensor Capabilities

The Android sensor framework provides several methods that make it easy for you to determine at runtime which sensors are on a device.

The API also provides methods that let you determine the capabilities of each sensor, such as its maximum range, its resolution, and its power requirements.

To identify the sensors that are on a device you first need to get a reference to the sensor service.

To do this, you create an instance of the [SensorManager](https://developer.android.com/reference/android/hardware/SensorManager) class by calling the [getSystemService()](https://developer.android.com/reference/android/content/Context#getSystemService(java.lang.Class%3CT%3E)) method and passing in the [SENSOR\_SERVICE](https://developer.android.com/reference/android/content/Context#SENSOR_SERVICE) argument.

For example:

private SensorManager sensorManager;  
...  
sensorManager = (SensorManager) getSystemService(Context.SENSOR\_SERVICE);

Next, you can get a listing of every sensor on a device by calling the [getSensorList()](https://developer.android.com/reference/android/hardware/SensorManager#getSensorList(int)) method and using the [TYPE\_ALL](https://developer.android.com/reference/android/hardware/Sensor#TYPE_ALL) constant.

For example:

List<Sensor> deviceSensors = sensorManager.getSensorList(Sensor.TYPE\_ALL);

This will give you all sensors present on device.

**Checking the presence of specific sensor on device:-**

You can also determine whether a specific type of sensor exists on a device by using the [getDefaultSensor()](https://developer.android.com/reference/android/hardware/SensorManager#getDefaultSensor(int)) method and passing in the type constant for a specific sensor.

 If a default sensor does not exist for a given type of sensor, the method call returns null, which means the device does not have that type of sensor.

For example, the following code checks whether there's a magnetometer on a device: private SensorManager sensorManager;  
...  
sensorManager = (SensorManager) getSystemService(Context.SENSOR\_SERVICE);  
if (sensorManager.getDefaultSensor(Sensor.TYPE\_MAGNETIC\_FIELD) != null){  
    // Success! There's a magnetometer.  
} else {  
    // Failure! No magnetometer.  
}

We also can get sensor version details by using sensor\_object.getVersion(); function.

Where sensor\_object is reference of sensor class.

We also can get vendor by using getVendor();

Another useful method is the [getMinDelay()](https://developer.android.com/reference/android/hardware/Sensor#getMinDelay()) method, which returns the minimum time interval (in microseconds) a sensor can use to sense data.

## Monitoring Sensor Events

To monitor raw sensor data you need to implement two callback methods that are exposed through the [SensorEventListener](https://developer.android.com/reference/android/hardware/SensorEventListener) interface: [onAccuracyChanged()](https://developer.android.com/reference/android/hardware/SensorEventListener#onAccuracyChanged(android.hardware.Sensor,%20int)) and [onSensorChanged()](https://developer.android.com/reference/android/hardware/SensorEventListener#onSensorChanged(android.hardware.SensorEvent)). The Android system calls these methods whenever the following occurs:

* **A sensor's accuracy changes.**

In this case the system invokes the [onAccuracyChanged()](https://developer.android.com/reference/android/hardware/SensorEventListener#onAccuracyChanged(android.hardware.Sensor,%20int)) method, providing you with a reference to the [Sensor](https://developer.android.com/reference/android/hardware/Sensor) object that changed and the new accuracy of the sensor. Accuracy is represented by one of four status constants: [SENSOR\_STATUS\_ACCURACY\_LOW](https://developer.android.com/reference/android/hardware/SensorManager#SENSOR_STATUS_ACCURACY_LOW), [SENSOR\_STATUS\_ACCURACY\_MEDIUM](https://developer.android.com/reference/android/hardware/SensorManager#SENSOR_STATUS_ACCURACY_MEDIUM), [SENSOR\_STATUS\_ACCURACY\_HIGH](https://developer.android.com/reference/android/hardware/SensorManager#SENSOR_STATUS_ACCURACY_HIGH), or [SENSOR\_STATUS\_UNRELIABLE](https://developer.android.com/reference/android/hardware/SensorManager#SENSOR_STATUS_UNRELIABLE).

* **A sensor reports a new value.**

In this case the system invokes the [onSensorChanged()](https://developer.android.com/reference/android/hardware/SensorEventListener#onSensorChanged(android.hardware.SensorEvent)) method, providing you with a [SensorEvent](https://developer.android.com/reference/android/hardware/SensorEvent) object. A [SensorEvent](https://developer.android.com/reference/android/hardware/SensorEvent) object contains information about the new sensor data, including: the accuracy of the data, the sensor that generated the data, the timestamp at which the data was generated, and the new data that the sensor recorded.

The following code shows how to use the [onSensorChanged()](https://developer.android.com/reference/android/hardware/SensorEventListener#onSensorChanged(android.hardware.SensorEvent)) method to monitor data from the light sensor. This example displays the raw sensor data in a [TextView](https://developer.android.com/reference/android/widget/TextView) that is defined in the main.xml file as sensor\_data.

public class SensorActivity extends Activity implements SensorEventListener {  
    private SensorManager sensorManager;  
    private Sensor mLight;  
  
    @Override  
    public final void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.main);  
  
        sensorManager = (SensorManager) getSystemService(Context.SENSOR\_SERVICE);  
        mLight = sensorManager.getDefaultSensor(Sensor.TYPE\_LIGHT);  
    }  
  
    @Override  
    public final void onAccuracyChanged(Sensor sensor, int accuracy) {  
        // Do something here if sensor accuracy changes.  
    }  
  
    @Override  
    public final void onSensorChanged(SensorEvent event) {  
        // The light sensor returns a single value.  
        // Many sensors return 3 values, one for each axis.  
        float lux = event.values[0];  
        // Do something with this sensor value.  
    }  
  
    @Override  
    protected void onResume() {  
        super.onResume();  
        sensorManager.registerListener(this, mLight, SensorManager.SENSOR\_DELAY\_NORMAL);  
    }  
  
    @Override  
    protected void onPause() {  
        super.onPause();  
        sensorManager.unregisterListener(this);  
    }  
}

we have to register the sensor so that whenever sensor value changed then onSensorChaanged() method is called.

Whenever we don’t need sensor value we can unregister the sensor.

**Notification:-**

A **notification** is a message you can display to the user outside of your application's normal UI.

When you tell the system to issue a notification, it first appears as an icon in the notification area.

To see the details of the notification, the user opens the notification drawer. Both the notification area and the notification drawer are system-controlled areas that the user can view at any time.

Android **Toast** class provides a handy way to show users alerts but problem is that these alerts are not persistent which means alert flashes on the screen for a few seconds and then disappears.

## Create and Send Notifications

You have simple way to create a notification. Follow the following steps in your application to create a notification −

### Step 1 - Create Notification Builder

As a first step is to create a notification builder using *NotificationCompat.Builder.build()*. You will use Notification Builder to set various Notification properties like its small and large icons, title, priority etc.

NotificationCompat.Builder mBuilder = new NotificationCompat.Builder(this)

### Step 2 - Setting Notification Properties

Once you have **Builder** object, you can set its Notification properties using Builder object as per your requirement. But this is mandatory to set at least following −

* A small icon, set by **setSmallIcon()**
* A title, set by **setContentTitle()**
* Detail text, set by **setContentText()**

mBuilder.setSmallIcon(R.drawable.notification\_icon);

mBuilder.setContentTitle("Notification Alert, Click Me!");

mBuilder.setContentText("Hi, This is Android Notification Detail!");

### Step 3 - Attach Actions

This is an optional part and required if you want to attach an action with the notification. An action allows users to go directly from the notification to an **Activity** in your application, where they can look at one or more events or do further work.

The action is defined by a **PendingIntent** containing an **Intent** that starts an Activity in your application. To associate the PendingIntent with a gesture, call the appropriate method of *NotificationCompat.Builder*. For example, if you want to start Activity when the user clicks the notification text in the notification drawer, you add the PendingIntent by calling **setContentIntent()**.

### Step 4 - Issue the notification

Finally, you pass the Notification object to the system by calling NotificationManager.notify() to send your notification. Make sure you call **NotificationCompat.Builder.build()** method on builder object before notifying it. This method combines all of the options that have been set and return a new **Notification** object.

NotificationManager mNotificationManager = (NotificationManager) getSystemService(Context.NOTIFICATION\_SERVICE);

// notificationID allows you to update the notification later on.

mNotificationManager.notify(notificationID, mBuilder.build());

**example program which includes all four steps of notification are ;-**

package com.example.notificationdemo;

import android.app.Activity;

import android.app.NotificationManager;

import android.app.PendingIntent;

import android.content.Context;

import android.content.Intent;

import android.support.v4.app.NotificationCompat;

import android.os.Bundle;

import android.view.View;

import android.widget.Button;

public class MainActivity extends Activity {

Button b1;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

b1 = (Button)findViewById(R.id.button);

b1.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

addNotification();

}

});

}

private void addNotification() {

NotificationCompat.Builder builder =

new NotificationCompat.Builder(this)

.setSmallIcon(R.drawable.abc)

.setContentTitle("Notifications Example")

.setContentText("This is a test notification");

Intent notificationIntent = new Intent(this, MainActivity.class);

PendingIntent contentIntent = PendingIntent.getActivity(this, 0, notificationIntent,

PendingIntent.FLAG\_UPDATE\_CURRENT);

builder.setContentIntent(contentIntent);

// Add as notification

NotificationManager manager = (NotificationManager) getSystemService(Context.NOTIFICATION\_SERVICE);

manager.notify(0, builder.build());

}

}

**Location Based Services:-**

This becomes possible with the help of **Google Play services**, which facilitates adding location awareness to your app with automated location tracking, geofencing, and activity recognition.

This tutorial shows you how to use Location Services in your APP to get the current location, get periodic location updates, look up addresses etc.

The Location Object

The **Location** object represents a geographic location which can consist of a latitude, longitude, time stamp, and other information such as bearing, altitude and velocity. There are following important methods which you can use with Location object to get location specific information −

|  |  |
| --- | --- |
| **Sr.No.** | **Method & Description** |
| 1 | **float distanceTo(Location dest)**  Returns the approximate distance in meters between this location and the given location. |
| 2 | **float getAccuracy()**  Get the estimated accuracy of this location, in meters. |
| 3 | **double getAltitude()**  Get the altitude if available, in meters above sea level. |
| 4 | **float getBearing()**  Get the bearing, in degrees. |
| 5 | **double getLatitude()**  Get the latitude, in degrees. |
| 6 | **double getLongitude()**  Get the longitude, in degrees. |
| 7 | **float getSpeed()**  Get the speed if it is available, in meters/second over ground. |
| 8 | **boolean hasAccuracy()**  True if this location has an accuracy. |
| 9 | **boolean hasAltitude()**  True if this location has an altitude. |
| 10 | **boolean hasBearing()**  True if this location has a bearing. |
| 11 | **boolean hasSpeed()**  True if this location has a speed. |
| 12 | **void reset()**  Clears the contents of the location. |
| 13 | **void setAccuracy(float accuracy)**  Set the estimated accuracy of this location, meters. |
| 14 | **void setAltitude(double altitude)**  Set the altitude, in meters above sea level. |
| 15 | **void setBearing(float bearing)**  Set the bearing, in degrees. |
| 16 | **void setLatitude(double latitude)**  Set the latitude, in degrees. |
| 17 | **void setLongitude(double longitude)**  Set the longitude, in degrees. |
| 18 | **void setSpeed(float speed)**  Set the speed, in meters/second over ground. |
| 19 | **String toString()**  Returns a string containing a concise, human-readable description of this object. |

Get the Current Location

To get the current location, create a location client which is **LocationClient** object, connect it to Location Services using **connect()** method, and then call its **getLastLocation()** method. This method returns the most recent location in the form of **Location** object that contains latitude and longitude coordinates and other information as explained above.

These interfaces provide following important callback methods, which you need to implement in your activity class −

|  |  |
| --- | --- |
| **Sr.No.** | **Callback Methods & Description** |
| 1 | **abstract void onConnected(Bundle connectionHint)**  This callback method is called when location service is connected to the location client successfully. You will use **connect()** method to connect to the location client. |
| 2 | **abstract void onDisconnected()**  This callback method is called when the client is disconnected. You will use **disconnect()** method to disconnect from the location client. |
| 3 | **abstract void onConnectionFailed(ConnectionResult result)**  This callback method is called when there was an error connecting the client to the service. |

Get the Updated Location

If you are willing to have location updates, then apart from above mentioned interfaces, you will need to implement **LocationListener** interface as well. This interface provide following callback method, which you need to implement in your activity class −

|  |  |
| --- | --- |
| **Sr.No.** | **Callback Method & Description** |
| 1 | **abstract void onLocationChanged(Location location)**  This callback method is used for receiving notifications from the LocationClient when the location has changed. |

package com.example.tutorialspoint7.myapplication;

import android.app.AlertDialog;

import android.app.Service;

import android.content.Context;

import android.content.DialogInterface;

import android.content.Intent;

import android.location.Location;

import android.location.LocationListener;

import android.location.LocationManager;

import android.os.Bundle;

import android.os.IBinder;

import android.provider.Settings;

import android.util.Log;

public class gps exten ds Service implements LocationListener {

boolean isGPSEnabled = false;

boolean isNetworkEnabled = false;

Location location; // location

double latitude; // latitude

double longitude; // longitude

protected LocationManager locationManager;

locationManager = (LocationManager) mContext.getSystemService(LOCATION\_SERVICE);

// getting GPS status

isGPSEnabled = locationManager.isProviderEnabled(LocationManager.GPS\_PROVIDER);

// getting network status

isNetworkEnabled = locationManager

.isProviderEnabled(LocationManager.NETWORK\_PROVIDER);

location = locationManager

.getLastKnownLocation(LocationManager.NETWORK\_PROVIDER);

if (location != null) {

latitude = location.getLatitude();

longitude = location.getLongitude();

@Override

public void onLocationChanged(Location location) {

}

We have to implement Location Listener class and we have to override onLocationChanged() method .

So when ever the device location gets changed this method is called and gives location object from which we can get latitude and longitude.

**Sending Email:-**

## Email is messages distributed by electronic means from one system user to one or more recipients via a network. To send an email from your application, you don’t have to implement an email client from the beginning, but you can use an existing one like the default Email app provided from Android, Gmail, Outlook, K-9 Mail etc. For this purpose, we need to write an Activity that launches an email client, using an implicit Intent with the right action and data. In this example, we are going to send an email from our app by using an Intent object that launches existing email clients. Intent Object - Action to send Email

You will use **ACTION\_SEND** action to launch an email client installed on your Android device. Following is simple syntax to create an intent with ACTION\_SEND action.

Intent emailIntent = new Intent(Intent.ACTION\_SEND);

Intent Object - Data/Type to send Email

To send an email you need to specify **mailto:** as URI using setData() method and data type will be to **text/plain** using setType() method as follows −

emailIntent.setData(Uri.parse("mailto:"));

emailIntent.setType("text/plain");

for example :

ackage com.example.tutorialspoint;

import android.net.Uri;

import android.os.Bundle;

import android.app.Activity;

import android.content.Intent;

import android.util.Log;

import android.view.Menu;

import android.view.View;

import android.widget.Button;

import android.widget.Toast;

public class MainActivity extends Activity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

Button startBtn = (Button) findViewById(R.id.sendEmail);

startBtn.setOnClickListener(new View.OnClickListener() {

public void onClick(View view) {

sendEmail();

}

});

}

protected void sendEmail() {

Log.i("Send email", "");

String[] TO = {""};///we have to give the receiver email id.

String[] CC = {""};

Intent emailIntent = new Intent(Intent.ACTION\_SEND);

emailIntent.setData(Uri.parse("mailto:"));

emailIntent.setType("text/plain");

emailIntent.putExtra(Intent.EXTRA\_EMAIL, TO);

emailIntent.putExtra(Intent.EXTRA\_CC, CC);

emailIntent.putExtra(Intent.EXTRA\_SUBJECT, "Your subject");

emailIntent.putExtra(Intent.EXTRA\_TEXT, "Email message goes here");

try {

startActivity(Intent.createChooser(emailIntent, "Send mail..."));

finish();

Log.i("Finished sending email...", "");

} catch (android.content.ActivityNotFoundException ex) {

Toast.makeText(MainActivity.this, "There is no email client installed.", Toast.LENGTH\_SHORT).show();

}

}

}

**Sending SMS:-**

In Android, you can use SmsManager API or devices Built-in SMS application to send SMS's. In this tutorial, we shows you two basic examples to send SMS message −

**SmsManager API**

SmsManager smsManager = SmsManager.getDefault();

smsManager.sendTextMessage("phoneNo", null, "sms message", null, null);

and we have to give permission in manifest file.

<uses-permission android:name="android.permission.SEND\_SMS" />

We have some methods in Smsamnager class they are.

**ArrayList<String> divideMessage(String text)**

This method divides a message text into several fragments, none bigger than the maximum SMS message size.

**static SmsManager getDefault()**

This method is used to get the default instance of the SmsManager

For example:-

mport android.widget.Button;

import android.widget.EditText;

import android.widget.Toast;

public class MainActivity extends Activity {

private static final int MY\_PERMISSIONS\_REQUEST\_SEND\_SMS =0 ;

Button sendBtn;

EditText txtphoneNo;

EditText txtMessage;

String phoneNo;

String message;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

sendBtn = (Button) findViewById(R.id.btnSendSMS);

txtphoneNo = (EditText) findViewById(R.id.editText);

txtMessage = (EditText) findViewById(R.id.editText2);

sendBtn.setOnClickListener(new View.OnClickListener() {

public void onClick(View view) {

sendSMSMessage();

}

});

}

protected void sendSMSMessage() {

phoneNo = txtphoneNo.getText().toString();

message = txtMessage.getText().toString();

SmsManager smsManager = SmsManager.getDefault();

smsManager.sendTextMessage(phoneNo, null, message, null, null);

Toast.makeText(getApplicationContext(), "SMS sent.",

Toast.LENGTH\_LONG).show();

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

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

package="com.example.tutorialspoint" >

<uses-permission android:name="android.permission.SEND\_SMS" />

<application

android:allowBackup="true"

android:icon="@drawable/ic\_launcher"

android:label="@string/app\_name"

android:theme="@style/AppTheme" >

<activity

android:name="com.example.tutorialspoint.MainActivity"

android:label="@string/app\_name" >

<intent-filter>

<action android:name="android.intent.action.MAIN" />

<category android:name="android.intent.category.LAUNCHER" />

</intent-filter>

</activity>

</application>

</manifest>

This is manifest file in which we have to give permissions for SMS\_SEND.

**Phone calls :-**

## Android provides Built-in applications for phone calls, in some occasions we may need to make a phone call through our application. This could easily be done by using implicit Intent with appropriate actions. Also, we can use PhoneStateListener and TelephonyManager classes, in order to monitor the changes in some telephony states on the device. Intent Object - Action to make Phone Call

You will use **ACTION\_CALL** action to trigger built-in phone call functionality available in Android device. Following is simple syntax to create an intent with ACTION\_CALL action

Intent phoneIntent = new Intent(Intent.ACTION\_CALL);

You can use **ACTION\_DIAL** action instead of ACTION\_CALL, in that case you will have option to modify hardcoded phone number before making a call instead of making a direct call.

To make a phone call at a given number 91-000-000-0000, you need to specify **tel:** as URI using setData() method as follows −

phoneIntent.setData(Uri.parse("tel:91-000-000-0000"));

for example:-

ackage com.example.saira\_000.myapplication;

import android.Manifest;

import android.content.Intent;

import android.content.pm.PackageManager;

import android.net.Uri;

import android.os.Bundle;

import android.support.v4.app.ActivityCompat;

import android.support.v7.app.AppCompatActivity;

import android.view.View;

import android.widget.Button;

public class MainActivity extends AppCompatActivity {

private Button button;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

button = (Button) findViewById(R.id.buttonCall);

button.setOnClickListener(new View.OnClickListener() {

public void onClick(View arg0) {

Intent callIntent = new Intent(Intent.ACTION\_CALL);

callIntent.setData(Uri.parse("tel:0377778888"));

if (ActivityCompat.checkSelfPermission(MainActivity.this,

Manifest.permission.CALL\_PHONE) != PackageManager.PERMISSION\_GRANTED) {

return;

}

startActivity(callIntent);

}

});

}

}

And also we need to add permission in manifest file as shown below

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

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

package="com.example.saira\_000.myapplication" >

<uses-permission android:name="android.permission.CALL\_PHONE" />

<application

android:allowBackup="true"

android:icon="@drawable/ic\_launcher"

android:label="@string/app\_name"

android:theme="@style/AppTheme" >

<activity

android:name="com.example.saira\_000.myapplication.MainActivity"

android:label="@string/app\_name" >

<intent-filter>

<action android:name="android.intent.action.MAIN" />

<category android:name="android.intent.category.LAUNCHER" />

</intent-filter>

</activity>

</application>

</manifest>

**Alert-Dialog :-**

A Dialog is small window that prompts the user to a decision or enter additional information. Some times in your application, if you wanted to ask the user about taking a decision between yes or no in response of any particular action taken by the user, by remaining in the same activity and without changing the screen, you can use Alert Dialog.

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

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

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

alertDialogBuilder.setPositiveButton(CharSequence text,

DialogInterface.OnClickListener listener)

alertDialogBuilder.setNegativeButton(CharSequence text,

DialogInterface.OnClickListener listener)

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

AlertDialog alertDialog = alertDialogBuilder.create();

alertDialog.show();

for example:-

ackage com.example.sairamkrishna.myapplication;

import android.app.AlertDialog;

import android.content.DialogInterface;

import android.support.v7.app.ActionBarActivity;

import android.os.Bundle;

import android.view.View;

import android.widget.Toast;

public class MainActivity extends ActionBarActivity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

}

public void open(View view){

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

alertDialogBuilder.setMessage("Are you sure,

You wanted to make decision");

alertDialogBuilder.setPositiveButton("yes",

new DialogInterface.OnClickListener() {

@Override

public void onClick(DialogInterface arg0, int arg1) {

Toast.makeText(MainActivity.this,"You clicked yes

button",Toast.LENGTH\_LONG).show();

}

});

alertDialogBuilder.setNegativeButton("No",new DialogInterface.OnClickListener() {

Override

public void onClick(DialogInterface dialog, int which) {

finish();

}

});

AlertDialog alertDialog = alertDialogBuilder.create();

alertDialog.show();

}

}

**Animation:-**

In order to perform animation in android , we are going to call a static function loadAnimation() of the class AnimationUtils. We are going to receive the result in an instance of Animation Object. Its syntax is as follows −

Animation animation = AnimationUtils.loadAnimation(getApplicationContext(),

R.anim.myanimation);

In order to apply this animation to an object , we will just call the startAnimation() method of the object. Its syntax is −

ImageView image1 = (ImageView)findViewById(R.id.imageView1);

image.startAnimation(animation);

here we should create an xml file in resources.anim folder.

Then we can give that animation to the image view as shown above.

For ecample I have created a xml file in animation folder.

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

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

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

android:fromXScale="0.5"

android:toXScale="3.0"

android:fromYScale="0.5"

android:toYScale="3.0"

android:duration="5000"

android:pivotX="50%"

android:pivotY="50%" >

</scale>

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

android:startOffset="5000"

android:fromXScale="3.0"

android:toXScale="0.5"

android:fromYScale="3.0"

android:toYScale="0.5"

android:duration="5000"

android:pivotX="50%"

android:pivotY="50%" >

</scale>

</set>

Now I use this myanimation for image view in main activity.

import android.app.Activity;

import android.os.Bundle;

import android.view.View;

import android.view.animation.Animation;

import android.view.animation.AnimationUtils;

import android.widget.ImageView;

import android.widget.Toast;

public class MainActivity extends Activity {

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

}

public void clockwise(View view){

ImageView image = (ImageView)findViewById(R.id.imageView);

Animation animation = AnimationUtils.loadAnimation(getApplicationContext(),

R.anim.myanimation);

image.startAnimation(animation)

}

}

here we are loading that animation to the imageview.then that image will work according to the myanimation xml file.

**Audio Recording :-**

Android has a built in microphone through which you can capture audio and store it , or play it in your phone. There are many ways to do that but the most common way is through MediaRecorder class.

Android provides MediaRecorder class to record audio or video. In order to use MediaRecorder class ,you will first create an instance of MediaRecorder class. Its syntax is given below.

MediaRecorder myAudioRecorder = new MediaRecorder();

Now we should set source of the audio and where should we store the audio .

myAudioRecorder.setAudioSource(MediaRecorder.AudioSource.MIC)

myAudioRecorder.setOutputFile(outputFile);

myAudioRecorder.setOutputFormat(MediaRecorder.OutputFormat.THREE\_GPP);

After specifying the audio source and format and its output file, we can then call the two basic methods prepare and start to start recording the audio.

myAudioRecorder.prepare();

myAudioRecorder.start();

for example:-

import android.media.MediaPlayer;

import android.media.MediaRecorder;

import android.os.Environment;

import android.support.v7.app.AppCompatActivity;

import android.os.Bundle;

import android.view.View;

import android.widget.Button;

import android.widget.Toast;

import java.io.IOException;

import java.util.Random;

mport android.support.v4.app.ActivityCompat;

import android.content.pm.PackageManager;

import android.support.v4.content.ContextCompat;

public class MainActivity extends AppCompatActivity {

Button buttonStart, buttonStop,

MediaRecorder mediaRecorder ;

MediaPlayer mediaPlayer ;

mediaRecorder.setAudioSource(MediaRecorder.AudioSource.MIC);

mediaRecorder.setOutputFormat(MediaRecorder.OutputFormat.THREE\_GPP);

mediaRecorder.setOutputFile(outputFile);//sets format of audio and output file and source of audio.

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

buttonStart = (Button) findViewById(R.id.button);

buttonStop = (Button) findViewById(R.id.button2);

buttonStart.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View view) {

try {

mediaRecorder.prepare();

mediaRecorder.start();//start recording.

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

buttonStop.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View view) {

mediaRecorder.stop();//to stop recording.

}

}

**Audio Manager :-**

You can easily control your ringer volume and ringer profile i-e:(silent,vibrate,loud e.t.c) in android.

Android provides AudioManager class that provides access to these controls.

In order to use AndroidManager class, you have to first create an object of AudioManager class by calling the **getSystemService()** method.

Its syntax is given below.

private AudioManager myAudioManager;

myAudioManager = (AudioManager)getSystemService(Context.AUDIO\_SERVICE);

Once you instantiate the object of AudioManager class, you can use **setRingerMode** method to set the audio or ringer profile of your device. Its syntax is given below.

myAudioManager.setRingerMode(AudioManager.RINGER\_MODE\_VIBRATE);

other modes are

RINGER\_MODE\_NORMAL

RINGER\_MODE\_SILENT

Once you have set the mode , you can call the **getRingerMode()** method to get the set state of the system. Its syntax is given below.

int mod = myAudioManager.getRingerMode();

**Bluetooth :-**

Among many ways, Bluetooth is a way to send or receive data between two different devices. Android platform includes support for the Bluetooth framework that allows a device to wirelessly exchange data with other Bluetooth devices.

Android provides BluetoothAdapter class to communicate with Bluetooth. Create an object of this calling by calling the static method getDefaultAdapter(). Its syntax is given below.

private BluetoothAdapter BA;

BA = BluetoothAdapter.getDefaultAdapter();

In order to enable the Bluetooth of your device, call the intent with the following Bluetooth constant ACTION\_REQUEST\_ENABLE. Its syntax is.

Intent turnOn = new Intent(BluetoothAdapter.ACTION\_REQUEST\_ENABLE);

startActivityForResult(turnOn, 0);

Once you enable the Bluetooth , you can get a list of paired devices by calling getBondedDevices() method. It returns a set of bluetooth devices. Its syntax is.

private Set<BluetoothDevice>pairedDevices;

pairedDevices = BA.getBondedDevices();

Now we get the list of paired devices , so to send data we have to create bluetoothSocket as shown below

Object[] devices = pairedDevices.toArray();

BluetoothDevice device = (BluetoothDevice) devices[0];

ParcelUuid[] uuid = device.getUuids();

try {

BluetoothSocket socket = device.createInsecureRfcommSocketToServiceRecord(uuid[0].getUuid());

socket.connect();

Toast.makeText(this, "Socket connected", Toast.LENGTH\_LONG).show();

outputStream = socket.getOutputStream();

inStream = socket.getInputStream();

} catch (IOException e) {

Toast.makeText(this, "Exception found", Toast.LENGTH\_LONG).show();

**Camera:-**

These are the following two ways, in which you can use camera in your application

* Using existing android camera application in our application
* Directly using Camera API provided by android in our application
* Using existing android camera application in our application
* You will use MediaStore.ACTION\_IMAGE\_CAPTURE to launch an existing camera application installed on your phone. Its syntax is given below
* Intent intent = new Intent(android.provider.MediaStore.ACTION\_IMAGE\_CAPTURE);
* Now you will use the function *startActivityForResult()* to launch this activity and wait for its result. Its syntax is given below
* startActivityForResult(intent,0)

No matter which function you used to launch the activity , they all return the result. The result can be obtained by overriding the function *onActivityResult*.

@Override  
protected void onActivityResult(int requestCode, int resultCode, Intent data) {  
    if (requestCode == 0 && resultCode == RESULT\_OK) {  
        Bundle extras = data.getExtras();  
        Bitmap imageBitmap = (Bitmap) extras.get("data");  
        imageView.setImageBitmap(imageBitmap);  
    }  
}

**Save image file:-**

To save the image in our respective directory we have to create a function as follows

private File createImageFile() throws IOException {  
    // Create an image file name

String currentPhotoPath;  
    String timeStamp = new SimpleDateFormat("yyyyMMdd\_HHmmss").format(new Date());  
    String imageFileName = "JPEG\_" + timeStamp + "\_";  
    File storageDir = getExternalFilesDir(Environment.DIRECTORY\_PICTURES);  
    File image = File.createTempFile(  
        imageFileName,  /\* prefix \*/  
        ".jpg",         /\* suffix \*/  
        storageDir      /\* directory \*/  
    );  
  
    // Save a file: path for use with ACTION\_VIEW intents  
    currentPhotoPath = image.getAbsolutePath();  
    return image;  
}

This returns the file in which we can store our image.

Now we can use this function as follows

static final int REQUEST\_TAKE\_PHOTO = 1;  
  
private void dispatchTakePictureIntent() {  
    Intent takePictureIntent = new Intent(MediaStore.ACTION\_IMAGE\_CAPTURE);  
    // Ensure that there's a camera activity to handle the intent  
    if (takePictureIntent.resolveActivity(getPackageManager()) != null) {  
        // Create the File where the photo should go  
        File photoFile = null;  
        try {  
            photoFile = createImageFile();  
        } catch (IOException ex) {  
            // Error occurred while creating the File  
            ...  
        }  
        // Continue only if the File was successfully created  
        if (photoFile != null) {  
            Uri photoURI = FileProvider.getUriForFile(this,  
                                                  "com.example.android.fileprovider",  
                                                  photoFile);  
            takePictureIntent.putExtra(MediaStore.EXTRA\_OUTPUT, photoURI);  
            startActivityForResult(takePictureIntent, REQUEST\_TAKE\_PHOTO);  
        }  
    }  
}

Now, you need to configure the [FileProvider](https://developer.android.com/reference/androidx/core/content/FileProvider). In your app's manifest, add a provider to your application:

<application>  
   ...  
   <provider  
        android:name="android.support.v4.content.FileProvider"  
        android:authorities="com.example.android.fileprovider"  
        android:exported="false"  
        android:grantUriPermissions="true">  
        <meta-data  
            android:name="android.support.FILE\_PROVIDER\_PATHS"  
            android:resource="@xml/file\_paths"></meta-data>  
    </provider>  
    ...  
</application>

**Recording videos:-**

First we have to advertise our application uses camera in manifest file.

<manifest ... >  
    <uses-feature android:name="android.hardware.camera"  
                  android:required="true" />  
    ...  
</manifest>

Now we have to create intent with action as ACTION\_VIDEO\_CAPTURE.

static final int REQUEST\_VIDEO\_CAPTURE = 1;  
  
private void dispatchTakeVideoIntent() {  
    Intent takeVideoIntent = new Intent(MediaStore.ACTION\_VIDEO\_CAPTURE);  
    if (takeVideoIntent.resolveActivity(getPackageManager()) != null) {  
        startActivityForResult(takeVideoIntent, REQUEST\_VIDEO\_CAPTURE);  
    }  
}

the video can be acquired by writing a method onActivityResult();

The Android Camera application returns the video in the [Intent](https://developer.android.com/reference/android/content/Intent) delivered to [onActivityResult()](https://developer.android.com/reference/android/app/Activity#onActivityResult(int,%20int,%20android.content.Intent)) as a [Uri](https://developer.android.com/reference/android/net/Uri) pointing to the video location in storage. The following code retrieves this video and displays it in a [VideoView](https://developer.android.com/reference/android/widget/VideoView).

@Override  
protected void onActivityResult(int requestCode, int resultCode, Intent intent) {  
    if (requestCode == REQUEST\_VIDEO\_CAPTURE && resultCode == RESULT\_OK) {  
        Uri videoUri = intent.getData();  
        videoView.setVideoURI(videoUri);  
    }  
}

**Text to speech :-**

Android allows you convert your text into voice. Not only you can convert it but it also allows you to speak text in variety of different languages.

private EditText write;

ttobj=new TextToSpeech(getApplicationContext(), new TextToSpeech.OnInitListener() {

@Override

public void onInit(int status) {

}

});

In this listener, you have to specify the properties for TextToSpeech object , such as its language ,pitch e.t.c. Language can be set by calling **setLanguage()** method. Its syntax is given below −

ttobj.setLanguage(Locale.UK);

Once you have set the language, you can call **speak** method of the class to speak the text. Its syntax is given below −

ttobj.speak(toSpeak, TextToSpeech.QUEUE\_FLUSH, null);

for example,:-

import android.app.Activity;

import android.os.Bundle;

import android.speech.tts.TextToSpeech;

import android.view.View;

import android.widget.Button;

import android.widget.EditText;

import java.util.Locale;

import android.widget.Toast;

public class MainActivity extends Activity {

TextToSpeech t1;

EditText ed1;

Button b1;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

ed1=(EditText)findViewById(R.id.editText);

b1=(Button)findViewById(R.id.button);

t1=new TextToSpeech(getApplicationContext(), new TextToSpeech.OnInitListener() {

@Override

public void onInit(int status) {

if(status != TextToSpeech.ERROR) {

t1.setLanguage(Locale.UK);

}

}

});

b1.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View v) {

String toSpeak = ed1.getText().toString();

Toast.makeText(getApplicationContext(), toSpeak,Toast.LENGTH\_SHORT).show();

t1.speak(toSpeak, TextToSpeech.QUEUE\_FLUSH, null);

}

});

}

public void onPause(){

if(t1 !=null){

t1.stop();

t1.shutdown();

}

super.onPause();

}

}

**Wifi:-**

You can use the Wi-Fi scanning capabilities provided by the [WifiManager API](https://developer.android.com/reference/android/net/wifi/WifiManager) to get a list of Wi-Fi access points that are visible from the device.

There are three steps for wifi scanning process they are

1. **Register a broadcast listener** for [SCAN\_RESULTS\_AVAILABLE\_ACTION](https://developer.android.com/reference/android/net/wifi/WifiManager#SCAN_RESULTS_AVAILABLE_ACTION).
2. **Request a scan** using [WifiManager.startScan()](https://developer.android.com/reference/android/net/wifi/WifiManager#startScan()).
3. **get scan results** using [WifiManager.getScanResults()](https://developer.android.com/reference/android/net/wifi/WifiManager#getScanResults())

creating broadcast receiver

BroadcastReceiver wifiScanReceiver = new BroadcastReceiver() {  
  @Override  
  public void onReceive(Context c, Intent intent) {  
    boolean success = intent.getBooleanExtra(  
                       WifiManager.EXTRA\_RESULTS\_UPDATED, false);  
    if (success) {  
     List<ScanResult> results = wifiManager.getScanResults();

//use scan results for further operation.  
    } else {  
      // scan failure handling  
       List<ScanResult> results = wifiManager.getScanResults();  
  ... potentially use older scan results ...  
  
    }  
  }  
};

//Now we need wifi manager

WifiManager wifiManager = (WifiManager)  
                   context.getSystemService(Context.WIFI\_SERVICE);

IntentFilter intentFilter = new IntentFilter();  
intentFilter.addAction(WifiManager.SCAN\_RESULTS\_AVAILABLE\_ACTION);

//registering broadcast receiver.  
context.registerReceiver(wifiScanReceiver, intentFilter);

//now we need to start scan by using

boolean success = wifiManager.startScan();

whenever we start scan then on receive of broadcast receiver method is called and we will get the scan results by using wifiManager.getscanresults();.

**Connectivity:-**

**Usb:-**

 When an Android-powered device is in USB accessory mode, the attached Android USB accessory acts as the host, provides power to the USB bus, and enumerates connected devices.

First we have to get usbManager it can be obtained as follows

UsbManager usbmanager = (UsbManager) getSystemService(Context.USB\_SERVICE);

We have to obtain usbAccessory as follows

UsbAccessory accessory = (UsbAccessory) intent.getParcelableExtra(UsbManager.EXTRA\_ACCESSORY);

We can obtain list of accessories attached to device obtained as follows

UsbAccessory[] accessories = usbmanager.getAccessoryList();

It gives all the accessories attached .

After getting accessories to communicate with accessories we have to check wheather we have permissions or not.

mUsbManager.hasPermission(accessory) it returns true if it has permissions to communicate with accessory. Or false if we not have permissions.

To explicitly obtain permission, first create a broadcast receiver. This receiver listens for the intent that gets broadcast when you call [requestPermission()](https://developer.android.com/reference/android/hardware/usb/UsbManager#requestPermission(android.hardware.usb.UsbAccessory,%20android.app.PendingIntent)). The call to [requestPermission()](https://developer.android.com/reference/android/hardware/usb/UsbManager#requestPermission(android.hardware.usb.UsbAccessory,%20android.app.PendingIntent)) displays a dialog to the user asking for permission to connect to the accessory. The following sample code shows how to create the broadcast receiver:

private static final String ACTION\_USB\_PERMISSION =  
    "com.android.example.USB\_PERMISSION";  
private final BroadcastReceiver usbReceiver = new BroadcastReceiver() {  
  
    public void onReceive(Context context, Intent intent) {  
        String action = intent.getAction();  
        if (ACTION\_USB\_PERMISSION.equals(action)) {  
            synchronized (this) {  
                UsbAccessory accessory = (UsbAccessory) intent.getParcelableExtra(UsbManager.EXTRA\_ACCESSORY);  
  
                if (intent.getBooleanExtra(UsbManager.EXTRA\_PERMISSION\_GRANTED, false)) {  
                    if(accessory != null){  
                        //call method to set up accessory communication

openAccessory();

                    }  
                }  
                else {  
                    Log.d(TAG, "permission denied for accessory " + accessory);  
                }  
            }  
        }  
    }  
};

Now we have to register receiver in oncreate method of activity.

UsbManager usbManager = (UsbManager) getSystemService(Context.USB\_SERVICE);  
private static final String ACTION\_USB\_PERMISSION =  
    "com.android.example.USB\_PERMISSION";  
...  
permissionIntent = PendingIntent.getBroadcast(this, 0, new Intent(ACTION\_USB\_PERMISSION), 0);  
IntentFilter filter = new IntentFilter(ACTION\_USB\_PERMISSION);  
registerReceiver(usbReceiver, filter);

To display the dialog that asks users for permission to connect to the accessory, call the [requestPermission()](https://developer.android.com/reference/android/hardware/usb/UsbManager#requestPermission(android.hardware.usb.UsbAccessory,%20android.app.PendingIntent)) method:

UsbAccessory accessory;  
...  
usbManager.requestPermission(accessory, permissionIntent);

### Communicate with an accessory

You can communicate with the accessory by using the [UsbManager](https://developer.android.com/reference/android/hardware/usb/UsbManager) to obtain a file descriptor that you can set up input and output streams to read and write data to descriptor. The streams represent the accessory's input and output bulk endpoints. You should set up the communication between the device and accessory in another thread, so you don't lock the main UI thread. The following example shows how to open an accessory to communicate with:

UsbAccessory accessory;  
ParcelFileDescriptor fileDescriptor;  
FileInputStream inputStream;  
FileOutputStream outputStream;  
...  
  
private void openAccessory() {  
    Log.d(TAG, "openAccessory: " + accessory);  
    fileDescriptor = usbManager.openAccessory(accessory);  
    if (fileDescriptor != null) {  
        FileDescriptor fd = fileDescriptor.getFileDescriptor();  
        inputStream = new FileInputStream(fd);  
        outputStream = new FileOutputStream(fd);  
        Thread thread = new Thread(null, this, "AccessoryThread");  
        thread.start();  
    }  
}

### Terminate communication with an accessory

When you are done communicating with an accessory or if the accessory was detached, close the file descriptor that you opened by calling [close()](https://developer.android.com/reference/android/os/ParcelFileDescriptor#close()). To listen for detached events, create a broadcast receiver like below:

BroadcastReceiver usbReceiver = new BroadcastReceiver() {  
    public void onReceive(Context context, Intent intent) {  
        String action = intent.getAction();  
  
        if (UsbManager.ACTION\_USB\_ACCESSORY\_DETACHED.equals(action)) {  
            UsbAccessory accessory = (UsbAccessory)intent.getParcelableExtra(UsbManager.EXTRA\_ACCESSORY);  
            if (accessory != null) {  
                // call your method that cleans up and closes communication with the accessory  
            }  
        }  
    }  
};

# USB host

When your Android-powered device is in USB host mode, it acts as the USB host, powers the bus, and enumerates connected USB devices.

when communicating with a USB device. In general, you obtain a [UsbManager](https://developer.android.com/reference/android/hardware/usb/UsbManager) to retrieve the desired [UsbDevice](https://developer.android.com/reference/android/hardware/usb/UsbDevice). When you have the device, you need to find the appropriate [UsbInterface](https://developer.android.com/reference/android/hardware/usb/UsbInterface) and the [UsbEndpoint](https://developer.android.com/reference/android/hardware/usb/UsbEndpoint) of that interface to communicate on. Once you obtain the correct endpoint, open a [UsbDeviceConnection](https://developer.android.com/reference/android/hardware/usb/UsbDeviceConnection) to communicate with the USB device.

If you want your application to be notified of an attached USB device, specify an <intent-filter> and <meta-data> element pair for the android.hardware.usb.action.USB\_DEVICE\_ATTACHED intent in your main activity. The <meta-data> element points to an external XML resource file that declares identifying information about the device that you want to detect.

In the XML resource file, declare <usb-device> elements for the USB devices that you want to filter. The following list describes the attributes of <usb-device>. In general, use vendor and product ID if you want to filter for a specific device

Save the resource file in the res/xml/ directory

**Example for manifest file :**

<manifest ...>  
    <uses-feature android:name="android.hardware.usb.host" />  
    <uses-sdk android:minSdkVersion="12" />  
    ...  
    <application>  
        <activity ...>  
            ...  
            <intent-filter>  
                <action android:name="android.hardware.usb.action.USB\_DEVICE\_ATTACHED" />  
            </intent-filter>  
  
            <meta-data android:name="android.hardware.usb.action.USB\_DEVICE\_ATTACHED"  
                android:resource="@xml/device\_filter" />  
        </activity>  
    </application>  
</manifest>

In this case, the following resource file should be saved in res/xml/device\_filter.xml and specifies that any USB device with the specified attributes should be filtered:

<?xml version="1.0" encoding="utf-8"?>  
  
<resources>  
    <usb-device vendor-id="1234" product-id="5678" class="255" subclass="66" protocol="1" />  
</resources>

In your activity, you can obtain the [UsbDevice](https://developer.android.com/reference/android/hardware/usb/UsbDevice) that represents the attached device from the intent like this:

UsbDevice device = (UsbDevice) intent.getParcelableExtra(UsbManager.EXTRA\_DEVICE);

### obtain permission to communicate with a device

Before communicating with the USB device, your application must have permission from your users.

Explicitly asking for permission might be neccessary in some situations such as when your application enumerates USB devices that are already connected and then wants to communicate with one. You must check for permission to access a device before trying to communicate with it. If not, you will receive a runtime error if the user denied permission to access the device.

To explicitly obtain permission, first create a broadcast receiver. This receiver listens for the intent that gets broadcast when you call [requestPermission()](https://developer.android.com/reference/android/hardware/usb/UsbManager#requestPermission(android.hardware.usb.UsbAccessory,%20android.app.PendingIntent)). The call to [requestPermission()](https://developer.android.com/reference/android/hardware/usb/UsbManager#requestPermission(android.hardware.usb.UsbAccessory,%20android.app.PendingIntent)) displays a dialog to the user asking for permission to connect to the device. The following sample code shows how to create the broadcast receiver:

**for example:-**

private static final String ACTION\_USB\_PERMISSION =  
    "com.android.example.USB\_PERMISSION";  
private final BroadcastReceiver usbReceiver = new BroadcastReceiver() {  
  
    public void onReceive(Context context, Intent intent) {  
        String action = intent.getAction();  
        if (ACTION\_USB\_PERMISSION.equals(action)) {  
            synchronized (this) {  
                UsbDevice device = (UsbDevice)intent.getParcelableExtra(UsbManager.EXTRA\_DEVICE);  
  
                if (intent.getBooleanExtra(UsbManager.EXTRA\_PERMISSION\_GRANTED, false)) {  
                    if(device != null){  
                      //call method to set up device communication  
                   }  
                }  
                else {  
                    Log.d(TAG, "permission denied for device " + device);  
                }  
            }  
        }  
    }  
};

To register the broadcast receiver, add this in your onCreate() method in your activity:

UsbManager usbManager = (UsbManager) getSystemService(Context.USB\_SERVICE);  
private static final String ACTION\_USB\_PERMISSION =  
    "com.android.example.USB\_PERMISSION";  
...  
permissionIntent = PendingIntent.getBroadcast(this, 0, new Intent(ACTION\_USB\_PERMISSION), 0);  
IntentFilter filter = new IntentFilter(ACTION\_USB\_PERMISSION);  
registerReceiver(usbReceiver, filter);

To display the dialog that asks users for permission to connect to the device, call the [requestPermission()](https://developer.android.com/reference/android/hardware/usb/UsbManager#requestPermission(android.hardware.usb.UsbAccessory,%20android.app.PendingIntent)) method:

UsbDevice device;  
...  
usbManager.requestPermission(device, permissionIntent);

When users reply to the dialog, your broadcast receiver receives the intent that contains the [EXTRA\_PERMISSION\_GRANTED](https://developer.android.com/reference/android/hardware/usb/UsbManager#EXTRA_PERMISSION_GRANTED) extra, which is a boolean representing the answer. Check this extra for a value of true before connecting to the device.

### Communicate with a device

Communication with a USB device can be either synchronous or asynchronous. In either case, you should create a new thread on which to carry out all data transmissions, so you don't block the UI thread. To properly set up communication with a device, you need to obtain the appropriate [UsbInterface](https://developer.android.com/reference/android/hardware/usb/UsbInterface) and [UsbEndpoint](https://developer.android.com/reference/android/hardware/usb/UsbEndpoint) of the device that you want to communicate on and send requests on this endpoint with a [UsbDeviceConnection](https://developer.android.com/reference/android/hardware/usb/UsbDeviceConnection)

For example:-

private Byte[] bytes;  
private static int TIMEOUT = 0;  
private boolean forceClaim = true;  
  
...  
  
UsbInterface intf = device.getInterface(0);  
UsbEndpoint endpoint = intf.getEndpoint(0);  
UsbDeviceConnection connection = usbManager.openDevice(device);  
connection.claimInterface(intf, forceClaim);  
connection.bulkTransfer(endpoint, bytes, bytes.length, TIMEOUT); //do in another thread

we should do this above process in another thread because it blocks ui thread.

### Terminating communication with a device

When you are done communicating with a device or if the device was detached, close the [UsbInterface](https://developer.android.com/reference/android/hardware/usb/UsbInterface) and [UsbDeviceConnection](https://developer.android.com/reference/android/hardware/usb/UsbDeviceConnection) by calling [releaseInterface()](https://developer.android.com/reference/android/hardware/usb/UsbDeviceConnection#releaseInterface(android.hardware.usb.UsbInterface)) and [close()](https://developer.android.com/reference/android/hardware/usb/UsbDeviceConnection#close()). To listen for detached events, create a broadcast receiver like below:

BroadcastReceiver usbReceiver = new BroadcastReceiver() {  
    public void onReceive(Context context, Intent intent) {  
        String action = intent.getAction();  
  
      if (UsbManager.ACTION\_USB\_DEVICE\_DETACHED.equals(action)) {  
            UsbDevice device = (UsbDevice)intent.getParcelableExtra(UsbManager.EXTRA\_DEVICE);  
            if (device != null) {  
                // call your method that cleans up and closes communication with the device  
            }  
        }  
    }  
};

# Input events overview

On Android, there's more than one way to intercept the events from a user's interaction with your application.

## Event listeners

An event listener is an interface in the [View](https://developer.android.com/reference/android/view/View) class that contains a single callback method. These methods will be called by the Android framework when the View to which the listener has been registered is triggered by user interaction with the item in the UI.

Included in the event listener interfaces are the following callback methods:

onClick()

From [View.OnClickListener](https://developer.android.com/reference/android/view/View.OnClickListener). This is called when the user either touches the item (when in touch mode), or focuses upon the item with the navigation-keys or trackball and presses the suitable "enter" key or presses down on the trackball.

onLongClick()

From [View.OnLongClickListener](https://developer.android.com/reference/android/view/View.OnLongClickListener). This is called when the user either touches and holds the item (when in touch mode), or focuses upon the item with the navigation-keys or trackball and presses and holds the suitable "enter" key or presses and holds down on the trackball (for one second).

onFocusChange()

From [View.OnFocusChangeListener](https://developer.android.com/reference/android/view/View.OnFocusChangeListener). This is called when the user navigates onto or away from the item, using the navigation-keys or trackball.

onKey()

From [View.OnKeyListener](https://developer.android.com/reference/android/view/View.OnKeyListener). This is called when the user is focused on the item and presses or releases a hardware key on the device.

onTouch()

From [View.OnTouchListener](https://developer.android.com/reference/android/view/View.OnTouchListener). This is called when the user performs an action qualified as a touch event, including a press, a release, or any movement gesture on the screen (within the bounds of the item).

# Content providers

Content providers can help an application manage access to data stored by itself, stored by other apps, and provide a way to share data with other apps.

 Most importantly you can configure a content provider to allow other applications to securely access and modify your app data

A content provider component supplies data from one application to others on request. Such requests are handled by the methods of the ContentResolver class. A content provider can use different ways to store its data and the data can be stored in a database, in files, or even over a network.

**sometimes it is required to share data across applications. This is where content providers become very useful.**

 A content provider behaves very much like a database where you can query it, edit its content, as well as add or delete content using insert(), update(), delete(), and query() methods. In most cases this data is stored in an **SQlite** database.

A content provider is implemented as a subclass of **ContentProvider** class and must implement a standard set of APIs that enable other applications to perform transactions.

public class My Application extends ContentProvider {

}

To query a content provider, you specify the query string in the form of a URI which has following format −

<prefix>://<authority>/<data\_type>/<id>

Here is the detail of various parts of the URI −

|  |  |
| --- | --- |
| **Sr.No** | **Part & Description** |
| 1 | **prefix**  This is always set to content:// |
| 2 | **authority**  This specifies the name of the content provider, for example *contacts*, *browser* etc. For third-party content providers, this could be the fully qualified name, such as *com.tutorialspoint.statusprovider* |
| 3 | **data\_type**  This indicates the type of data that this particular provider provides. For example, if you are getting all the contacts from the *Contacts* content provider, then the data path would be *people* and URI would look like this*content://contacts/people* |
| 4 | **id**  This specifies the specific record requested. For example, if you are looking for contact number 5 in the Contacts content provider then URI would look like this *content://contacts/people/5*. |

## Keep the screen on

Certain apps need to keep the screen turned on, such as games or movie apps. The best way to do this is to use the [FLAG\_KEEP\_SCREEN\_ON](https://developer.android.com/reference/android/view/WindowManager.LayoutParams#FLAG_KEEP_SCREEN_ON) in your activity (and only in an activity, never in a service or other app component). For example:

Another way to implement this is in your application's layout XML file, by using the [android:keepScreenOn](https://developer.android.com/reference/android/R.attr#keepScreenOn) attribute:

**android:keepScreenOn="true">**

public class MainActivity extends Activity {  
  @Override  
  protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity\_main);  
    getWindow().addFlags(WindowManager.LayoutParams.FLAG\_KEEP\_SCREEN\_ON);  
  }  
}

Create Content Provider

This involves number of simple steps to create your own content provider.

* First of all you need to create a Content Provider class that extends the *ContentProviderbaseclass.*
* Second, you need to define your content provider URI address which will be used to access the content.
* Next you will need to create your own database to keep the content. Usually, Android uses SQLite database and framework needs to override *onCreate()* method which will use SQLite Open Helper method to create or open the provider's database. When your application is launched, the *onCreate()* handler of each of its Content Providers is called on the main application thread.

Here are the list of methods for accessing data using content providers

They are :-

* **onCreate()** This method is called when the provider is started.
* **query()** This method receives a request from a client. The result is returned as a Cursor object.
* **insert()**This method inserts a new record into the content provider.
* **delete()** This method deletes an existing record from the content provider.
* **update()** This method updates an existing record from the content provider.
* **getType()** This method returns the MIME type of the data at the given URI.

Example of content provider class is

ackage com.example.MyApplication;

import java.util.HashMap;

import android.content.ContentProvider;

import android.content.ContentUris;

import android.content.ContentValues;

import android.content.Context;

import android.content.UriMatcher;

import android.database.Cursor;

import android.database.SQLException;

import android.database.sqlite.SQLiteDatabase;

import android.database.sqlite.SQLiteOpenHelper;

import android.database.sqlite.SQLiteQueryBuilder;

import android.net.Uri;

import android.text.TextUtils;

public class StudentsProvider extends ContentProvider {

static final String PROVIDER\_NAME = "com.example.MyApplication.StudentsProvider";

static final String URL = "content://" + PROVIDER\_NAME + "/students";

static final Uri CONTENT\_URI = Uri.parse(URL);

static final String \_ID = "\_id";

static final String NAME = "name";

static final String GRADE = "grade";

private static HashMap<String, String> STUDENTS\_PROJECTION\_MAP;

static final int STUDENTS = 1;

static final int STUDENT\_ID = 2;

static final UriMatcher uriMatcher;

static{

uriMatcher = new UriMatcher(UriMatcher.NO\_MATCH);

uriMatcher.addURI(PROVIDER\_NAME, "students", STUDENTS);

uriMatcher.addURI(PROVIDER\_NAME, "students/#", STUDENT\_ID);

}

/\*\*

\* Database specific constant declarations

\*/

private SQLiteDatabase db;

static final String DATABASE\_NAME = "College";

static final String STUDENTS\_TABLE\_NAME = "students";

static final int DATABASE\_VERSION = 1;

static final String CREATE\_DB\_TABLE =

" CREATE TABLE " + STUDENTS\_TABLE\_NAME +

" (\_id INTEGER PRIMARY KEY AUTOINCREMENT, " +

" name TEXT NOT NULL, " +

" grade TEXT NOT NULL);";

/\*\*

\* Helper class that actually creates and manages

\* the provider's underlying data repository.

\*/

private static class DatabaseHelper extends SQLiteOpenHelper {

DatabaseHelper(Context context){

super(context, DATABASE\_NAME, null, DATABASE\_VERSION);

}

@Override

public void onCreate(SQLiteDatabase db) {

db.execSQL(CREATE\_DB\_TABLE);

}

@Override

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

db.execSQL("DROP TABLE IF EXISTS " + STUDENTS\_TABLE\_NAME);

onCreate(db);

}

}

@Override

public boolean onCreate() {

Context context = getContext();

DatabaseHelper dbHelper = new DatabaseHelper(context);

/\*\*

\* Create a write able database which will trigger its

\* creation if it doesn't already exist.

\*/

db = dbHelper.getWritableDatabase();

return (db == null)? false:true;

}

@Override

public Uri insert(Uri uri, ContentValues values) {

/\*\*

\* Add a new student record

\*/

long rowID = db.insert( STUDENTS\_TABLE\_NAME, "", values);

/\*\*

\* If record is added successfully

\*/

if (rowID > 0) {

Uri \_uri = ContentUris.withAppendedId(CONTENT\_URI, rowID);

getContext().getContentResolver().notifyChange(\_uri, null);

return \_uri;

}

throw new SQLException("Failed to add a record into " + uri);

}

@Override

public Cursor query(Uri uri, String[] projection,

String selection,String[] selectionArgs, String sortOrder) {

SQLiteQueryBuilder qb = new SQLiteQueryBuilder();

qb.setTables(STUDENTS\_TABLE\_NAME);

switch (uriMatcher.match(uri)) {

case STUDENTS:

qb.setProjectionMap(STUDENTS\_PROJECTION\_MAP);

break;

case STUDENT\_ID:

qb.appendWhere( \_ID + "=" + uri.getPathSegments().get(1));

break;

default:

}

if (sortOrder == null || sortOrder == ""){

/\*\*

\* By default sort on student names

\*/

sortOrder = NAME;

}

Cursor c = qb.query(db, projection, selection,

selectionArgs,null, null, sortOrder);

/\*\*

\* register to watch a content URI for changes

\*/

c.setNotificationUri(getContext().getContentResolver(), uri);

return c;

}

@Override

public int delete(Uri uri, String selection, String[] selectionArgs) {

int count = 0;

switch (uriMatcher.match(uri)){

case STUDENTS:

count = db.delete(STUDENTS\_TABLE\_NAME, selection, selectionArgs);

break;

case STUDENT\_ID:

String id = uri.getPathSegments().get(1);

count = db.delete( STUDENTS\_TABLE\_NAME, \_ID + " = " + id +

(!TextUtils.isEmpty(selection) ? "

AND (" + selection + ')' : ""), selectionArgs);

break;

default:

throw new IllegalArgumentException("Unknown URI " + uri);

}

getContext().getContentResolver().notifyChange(uri, null);

return count;

}

@Override

public int update(Uri uri, ContentValues values,

String selection, String[] selectionArgs) {

int count = 0;

switch (uriMatcher.match(uri)) {

case STUDENTS:

count = db.update(STUDENTS\_TABLE\_NAME, values, selection, selectionArgs);

break;

case STUDENT\_ID:

count = db.update(STUDENTS\_TABLE\_NAME, values,

\_ID + " = " + uri.getPathSegments().get(1) +

(!TextUtils.isEmpty(selection) ? "

AND (" +selection + ')' : ""), selectionArgs);

break;

default:

throw new IllegalArgumentException("Unknown URI " + uri );

}

getContext().getContentResolver().notifyChange(uri, null);

return count;

}

}

Above shown is content provider class so we can insert the content in data base by using the code below in our activity as follows

ContentValues values = new ContentValues();

values.put(StudentsProvider.NAME,

((EditText)findViewById(R.id.editText2)).getText().toString());

values.put(StudentsProvider.GRADE,

((EditText)findViewById(R.id.editText3)).getText().toString());

Uri uri = getContentResolver().insert(

StudentsProvider.CONTENT\_URI, values);

Above insert method invokes the insert method ogf student provider class which inserts data in squilite data base..

Android stack:-

**1.LINUX KERNEL:-**

The foundation of android consist of linux kernel.

## 2.Hardware Abstraction Layer (HAL)

The [hardware abstraction layer (HAL)](https://source.android.com/devices/architecture/hal-types) provides standard interfaces that expose device hardware capabilities to the higher-level [Java API framework](https://developer.android.com/guide/platform#api-framework). The HAL consists of multiple library modules, each of which implements an interface for a specific type of hardware component, such as the [camera](https://source.android.com/devices/camera/index.html) or [bluetooth](https://source.android.com/devices/bluetooth.html) module. When a framework API makes a call to access device hardware, the Android system loads the library module for that hardware component.

3.Native C/C++ Libraries

Many core Android system components and services, such as ART and HAL, are built from native code that require native libraries written in C and C++. The Android platform provides Java framework APIs to expose the functionality of some of these native libraries to apps. For example, you can access [OpenGL ES](https://developer.android.com/guide/topics/graphics/opengl) through the Android framework’s [Java OpenGL API](https://developer.android.com/reference/android/opengl/package-summary) to add support for drawing and manipulating 2D and 3D graphics in your app

## 4.Java API Framework

The entire feature-set of the Android OS is available to you through APIs written in the Java language. These APIs form the building blocks you need to create Android apps by simplifying the reuse of core, modular system components and services, which include the following:

* A rich and extensible [View System](https://developer.android.com/guide/topics/ui/overview) you can use to build an app’s UI, including lists, grids, text boxes, buttons, and even an embeddable web browser
* A [Resource Manager](https://developer.android.com/guide/topics/resources/overview), providing access to non-code resources such as localized strings, graphics, and layout files
* A [Notification Manager](https://developer.android.com/guide/topics/ui/notifiers/notifications) that enables all apps to display custom alerts in the status bar
* An [Activity Manager](https://developer.android.com/guide/components/activities) that manages the lifecycle of apps and provides a common [navigation back stack](https://developer.android.com/guide/components/tasks-and-back-stack)
* [Content Providers](https://developer.android.com/guide/topics/providers/content-providers) that enable apps to access data from other apps, such as the Contacts app, or to share their own data

Developers have full access to the same [framework APIs](https://developer.android.com/reference/packages) that Android system apps use.

5.System Apps

Android comes with a set of core apps for email, SMS messaging, calendars, internet browsing, contacts, and more. Apps included with the platform have no special status among the apps the user chooses to install. So a third-party app can become the user's default web browser, SMS messenger, or even the default keyboard (some exceptions **apply**, such as the system's Settings app).

The system apps function both as apps for users and to provide key capabilities that developers can access from their own app. For example, if your app would like to deliver an SMS message, you don't need to build that functionality yourself—you can instead invoke whichever SMS app is already installed to deliver a message to the recipient you specify.