**Bundle in Android**

* + Android Bundles are generally used for passing data from one activity to another. The bundle is always used with Intent in Android.
  + It can hold all types of values (int, String, boolean, char) and pass them to the new activity. In Bundle key-value pair concept is used.

e.g. putInt (String key, int value), getInt (String key, int value)

putString (String key, String value), getString (String key, String value)

**Pass data between activities** by using Bundle and Intent objects.

Step 1: On First activity

Your first create a Bundle object

Bundle b = new Bundle ();

Now, create an Intent object

Intent in = new Intent (getApplicationContext (), secondActivity.class);

Then, associate the string data stored in "GFG :- Main Activity" with bundle key "key1"

b.putString ("key1", "GFG :- Main Activity");

Pass bundle object b to the intent

in.putExtras(b);

and start second activity

startActivity(in);

Step 2: On Second Activity

to access the data passed from the first activity

Intent in = getIntent();

Now, you need to get the data from the bundle

Bundle b = in.getExtras(); Or Bundle b = getIntent().getExtras();

Finally, get the value of the string data associated with key named "key1"

String s = b.getString ("key1" , “Default”);

If there exists no mapping corresponding to the key, it may lead to NullPointerException. it’s recommended to add default values for the Bundle.

# Intent in Android

* In android, **Intent** is a messaging object which is used to request an action from another [app component](https://www.tutlane.com/tutorial/android/android-application-components-activities-intents-views-layouts-services) such as [activities](https://www.tutlane.com/tutorial/android/android-activity-lifecycle), [services](https://www.tutlane.com/tutorial/android/android-services-with-examples), [broadcast receivers](https://www.tutlane.com/tutorial/android/android-broadcastreceivers-with-example), and [content providers](https://www.tutlane.com/tutorial/android/android-content-providers-with-examples).
* It will help us to maintain the communication between [app components](https://www.tutlane.com/tutorial/android/android-application-components-activities-intents-views-layouts-services) from the same application as well as with the components of other applications.
* It is generally used with startActivity() method to invoke activity, broadcast receivers etc.
* These are the objects of **android.content.Intent** types.

Android intents are mainly used to:

Start the service

Launch an activity

Display a web page

Display a list of contacts

Broadcast a message

Dial a phone call etc.

There are three fundamental use cases :

| **Component** | **Description** |
| --- | --- |
| Starting an Activity | By sending an **Intent** object to startActivity() method we can start a new Activity or existing [Activity](https://www.tutlane.com/tutorial/android/android-activity-lifecycle) to perform required things. |
| Starting a Service | By sending an **Intent** object to startService() method we can start a new [Service](https://www.tutlane.com/tutorial/android/android-services-with-examples) or send required instructions to an existing Service. |
| Delivering a Broadcast | By sending an **Intent** object to sendBroadcast() method we can deliver our messages to other app broadcast receivers. |

Component Name

In android, the component name can be specified for intent by using a fully qualified class name and package name, e.g.  **com.tutlane.sampleActivity**. We can set the component name by setComponent(), setClass(), setClassName() or by using the **Intent** constructor.

Action

We can specify the action name of intent by using setAction() or with an **Intent** constructor.

ACTION\_VIEW -- having a information that activity can show to the user.

ACTION\_SEND -- some data that the user can share through another app such as an email app, social sharing app.

There are two types of intents:

# 1. Implicit Intents

[Implicit Intent](https://www.geeksforgeeks.org/android-implicit-and-explicit-intents-with-examples/) doesn’t specify the component. In such a case, intent provides information on available components provided by the system that is to be invoked.

e.g.

1. When we enter the website URL and click on the button it will open a website in a new browser window in the same application.

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

he simple code snippet of **implicit intent** in the android application.

Intent intent=new Intent(Intent.ACTION\_VIEW);  
intent.setData(Uri.parse("http://www.tutlane.com"));  
startActivity(intent);

# **2.** Explicit Intents

* **Explicit** intents specify the name of the targeted component to be invoked by [activity](https://www.tutlane.com/tutorial/android/android-activity-lifecycle). In such a case, intent provides the external class to be invoked. So only the specified target component will be invoked.
* By using explicit intents we can send or share data/content from one activity to another activity.

e.g. Performing an addition operation in one activity and sending those details to another [activity](https://www.tutlane.com/tutorial/android/android-activity-lifecycle).

the simple code snippet of **explicit intent** in the android application.

Intent di = new Intent(this, ActivityView.class);  
di.setData(Uri.parse("http://www.tutlane.com"));  
startService(di);

First activity (**MainActivity.java**)

Intent intent = new Intent(MainActivity.this,ResultActivity.class);  
intent.putExtra("SUM",num1+" + "+num2+" = "+(num1+num2));  
startActivity(intent);

Second activity (**ResultActivity.java**)

Intent intent = getIntent();  
String addition = (String)intent.getSerializableExtra("SUM");  
result.setText(addition);

**AndroidManifest.xml**

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

Intent - Activity2">

</activity>

# Context in Android

* + It's the context of current state of the application/object.
  + We can call context to get information about different parts of our application like Activities, Applications, etc.
  + Context is Instances of the class android.content.Context provide the connection to the Android system which executes the application.

**1. It is the Context of the current/active state of the application.**

**2. It is used to get information about the activity and application.**

**3. It is used to get access to resources, databases, shared preferences, etc.**

**4. Both the Activity and Application classes extend the Context class.**

Different methods by which you can get context

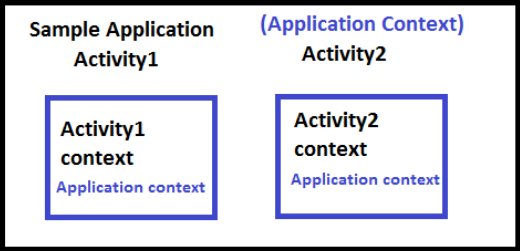
* getApplicationContext() - gives the Context setup (created) during the creation of application.
* getContext() - Context is tied to an Activity and its lifecycle.
* getBaseContext() - gives the context of the Activity that is set (created) by the constructor of activity.
* or this (when in the activity class)

There are mainly two types of Context :

1. **Application Context**

* This Context is tied to the[**Lifecycle of an Application**](https://www.geeksforgeeks.org/activity-lifecycle-in-android-with-demo-app/). Mainly it is an instance that is a singleton and can be accessed via **getApplicationContext()**.
* **getApplicationContext()** method is generally used for the application level and can be used to refer to all the activities.
* the **Application Context**is **not related to UI**.

#### **List of functionalities of Application Context :**

* Load Resource Values
* Start a Service
* Bind to a Service
* Send a Broadcast
* Register Broadcast Receiver

1. **Activity Context**

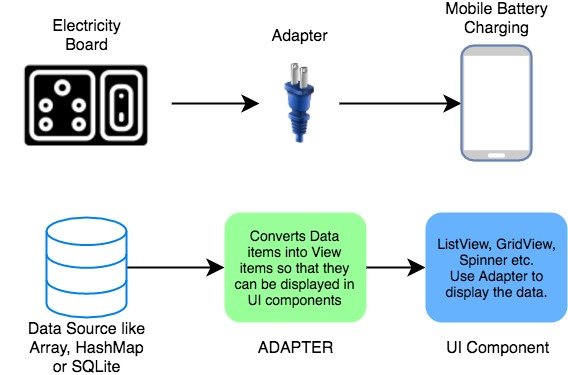
* This Context is tied to the life cycle of activity. It is used for the current Context. The method of invoking the Activity Context is **getContext()**.
* **getContext()** returns the Context which is linked to the Activity from which it is called. This is useful to call the Context from only the current running activity.

#### **List of Functionalities of Activity Context :**

* Load Resource Values
* Start a Service
* Bind to a Service
* Send a Broadcast
* Register Broadcast Receiver
* Start an Activity
* Show a Dialog
* Layout Inflation

**Adapter In Android**

* + The Adapter acts as a bridge between the UI Component and the Data Source. It converts data from the data sources into view items that can be displayed into the UI Component.
  + Data Source can be Arrays, HashMap, Database, etc. and UI Components can be [ListView](https://www.geeksforgeeks.org/android-listview-in-java-with-example/), GridView, [Spinner](https://www.geeksforgeeks.org/spinner-in-android-using-java-with-example/), etc.



#### **Adapters In Android :**

1. **BaseAdapter** – It is parent adapter for all other adapters

public class CustomAdapter extends BaseAdapter {

// Override methods

}

1. **ArrayAdapter** – It is used whenever we have a list of single items which is backed by an array e.g.  list of phone contacts, countries or names.

ArrayAdapter(Context context, int resource, int textViewResourceId,

T[] objects)

e.g. ArrayAdapter arrayAdapter = new ArrayAdapter(this, R.layout.itemListView, R.id.itemTextView, courseList[]);

1. **Custom ArrayAdapter** – It is used whenever we need to display a custom list.
2. **SimpleAdapter** – It is an easy adapter to map static data to views defined in your [XML](https://abhiandroid.com/ui/xml/) file
3. **Custom SimpleAdapter** – It is used whenever we need to display a customized list and needed to access the child items of the list or grid.

**Handler**

* **Handler** Used to communicate between the **UI** and **Background thread**.
* They are associated with message Queue of a Thread and they are used to send messages and runnable to the Message. When you create a new Handler it is bound to a [Looper](https://developer.android.com/reference/android/os/Looper).
* Handlers are used to manage tasks in the background also to handle and manage runnable objects.

There are two methods are in handler.

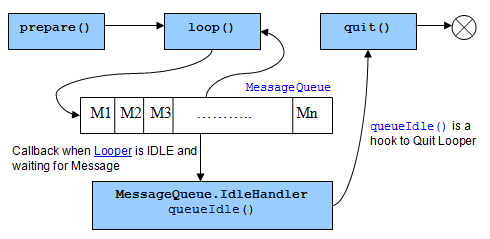
* + **Post()** − it going to post message from background thread to main thread using looper.
  + **sendmessage()** − if you want to organize what you have sent to ui (message from background thread) or ui functions. you should use sendMessage().

**Looper**

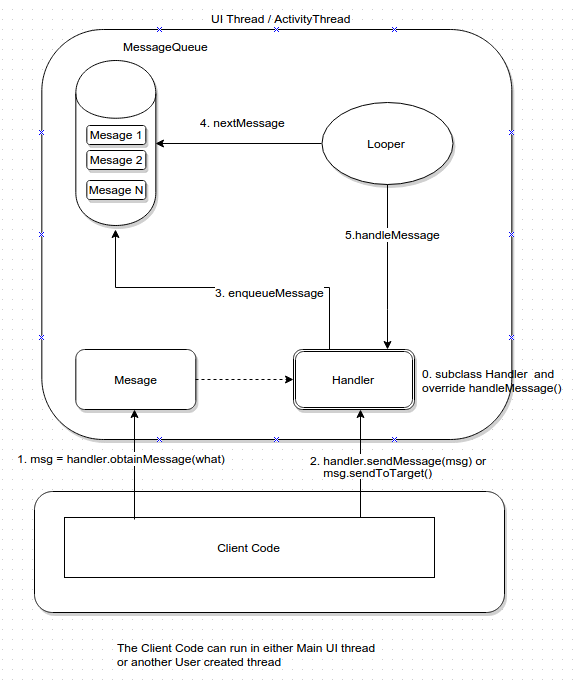
* Class used to run a message loop for a thread. Threads by default do not have a message loop associated with them.
* Looper is a class which is used to execute the Messages (Runnables) in a queue. Normal threads have no such queue, e.g. simple thread does not have any queue.

Looper.prepare(); -- preparing a looper in current thread

Looper.loop(); -- to create a message loop in the current thread



**Looper = Thread + MessageQueue**



**ViewBinding**

* **View Binding** provides the **views** to bind with the **activity**. It replaces **findViewById**
* Using ViewBinding the compilation of the code is a bit faster as compared to the traditional **findViewById()** method.
* Once view binding is enabled in a module, it generates a **binding class** for **each XML layout** file present in that module.
* e.g. the ViewBinding creates **activity\_main.xml** file(snake case) as **ActivityMainBinding** (pascal case), which contains all the property and instances of all the views containing in that layout.
* View binding is enabled on a module by module basis. To enable view binding in a module, set the viewBinding build option to true in the module-level build.gradle file.

**buildFeatures {**

**viewBinding true**

**}**

* + - To set up an instance of the binding class for use with an activity, perform the following steps in the activity's [onCreate ()](https://developer.android.com/reference/kotlin/android/app/Activity#oncreate) method:

// calling binding class for activity\_main.xml which is generated automatically

ActivityMainBinding mainxml;  
  
@Override  
protected void onCreate(Bundle savedInstanceState){  
 super.onCreate(savedInstanceState);

// inflating our xml layout in our activity main binding  
mainxml = ActivityMainBinding.Inflate(getLayoutInflater());

// below line is to set Content view for our layout  
setContentView(mainxml.getRoot());

// calling textView and changing the content  
mainxml.t1.setText("I am Changed Now");

// calling button and setting on click listener for our button.

// we have called our button with its id and set on click listener on it.  
mainxml.btn.setOnClickListener(new View.OnClickListener() {  
 @Override

public void onClick(View view){

startActivity (new Intent

(getApplicationContext(),newPage.class));

}  
});

}

**Data Binding**

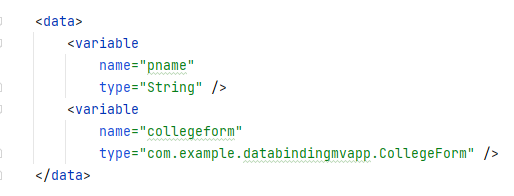
* + - The **Android Data Binding** is a support library that allows you to bind **UI components** in your layouts to **data sources** in your app using a declarative format
    - **It** creates a link between UI layer and the underlying data model that holds the information to display.
    - **It eliminates the need for these method calls “findViewById” and “setText.”**
    - The Data Binding Library generates binding classes that are used to access the layout's variables and views.
    - The expression language allows you to write expressions that connect variables to the views in the layout.
    - To configure your app to use data binding, enable the dataBinding build option in your build.gradle file in the app module

buildFeatures {  
        dataBinding true  
    }

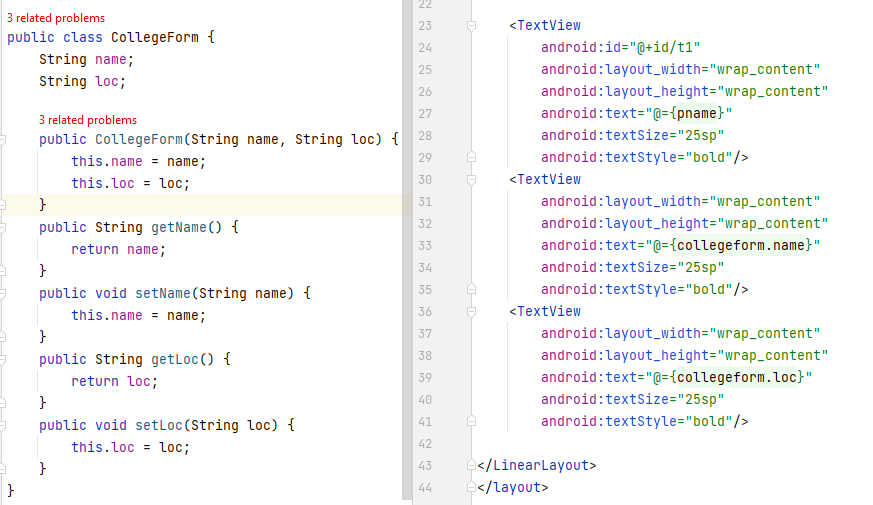
* + - Note the use of @{ } syntax in the assignment expression :

<TextView android:text="@{viewmodel.userName}" />

**Activity\_main.xml**



**CollegeForm.java (Model class) Activity\_main.xml**



**MainActivity.java**

