26-04-2020

# Data structure in java:

* Enumeration
* BitSet
* Vector
* Stack
* Dictionary
* Hashtable
* Properties

# Enumeration:

It defines the methods by which I can enumerate the elements in a collection of objects.

Boolean hasMoreElements(): it returns true if there still more element to extract.

Object nextElement(): it returns the next object in the enumeration.

[Live Demo](http://tpcg.io/os1Caq)

import java.util.Vector;

import java.util.Enumeration;

public class EnumerationTester {

public static void main(String args[]) {

Enumeration days;

Vector dayNames = new Vector();

dayNames.add("Sunday");

dayNames.add("Monday");

dayNames.add("Tuesday");

dayNames.add("Wednesday");

dayNames.add("Thursday");

dayNames.add("Friday");

dayNames.add("Saturday");

days = dayNames.elements();

while (days.hasMoreElements()) {

System.out.println(days.nextElement());

}

}

}

#### Output

Sunday

Monday

Tuesday

Wednesday

Thursday

Friday

Saturday

# Vector:

**Enumeration elements()**

Returns an enumeration of the components of this vector.

**void ensureCapacity(int minCapacity)**

Increases the capacity of this vector, if necessary, to ensure that it can hold at least the number of components specified by the minimum capacity argument.

**void ensureCapacity(int minCapacity)**

Increases the capacity of this vector, if necessary, to ensure that it can hold at least the number of components specified by the minimum capacity argument.

# Stack:

**Object peek( )**

Returns the element on the top of the stack, but does not remove it.

# Hashtable:

**Enumeration keys( )**

Returns an enumeration of the keys contained in the hash table.

[Live Demo](http://tpcg.io/5eD0B4)

import java.util.\*;

public class HashTableDemo {

public static void main(String args[]) {

// Create a hash map

Hashtable balance = new Hashtable();

Enumeration names;

String str;

double bal;

balance.put("Zara", new Double(3434.34));

balance.put("Mahnaz", new Double(123.22));

balance.put("Ayan", new Double(1378.00));

balance.put("Daisy", new Double(99.22));

balance.put("Qadir", new Double(-19.08));

// Show all balances in hash table.

names = balance.keys();

while(names.hasMoreElements()) {

str = (String) names.nextElement();

System.out.println(str + ": " + balance.get(str));

}

System.out.println();

// Deposit 1,000 into Zara's account

bal = ((Double)balance.get("Zara")).doubleValue();

balance.put("Zara", new Double(bal + 1000));

System.out.println("Zara's new balance: " + balance.get("Zara"));

}

}

#### Output

Qadir: -19.08

Zara: 3434.34

Mahnaz: 123.22

Daisy: 99<https://www.guru99.com/images/1/2.png>.22

Ayan: 1378.0

<https://www.guru99.com/images/1/2.png>

Zara's new balance: 4434.34

# JVM:



# ArrayList:

ArrayList is a data structure that can be stretched to accommodate additional elements within itself and shrink back to a smaller size when elements are removed. It is a very important data structure useful in handling the dynamic behavior of elements.

# HashMap:

A HashMap basically designates **unique keys** to corresponding **values** that can be retrieved.

HashMap stores only **object** **references**. That is why, it is impossible to use **primitive data types** like double or int. Use wrapper class (like Integer or Double) instead.

If **no element** exists in the Map, it will throw a ‘**NoSuchElementException’**.

import java.util.HashMap;

import java.util.Map;

public class Sample\_TestMaps{

public static void main(String[] args){

Map<String, String> objMap = new HashMap<String, String>();

objMap.put("Name", "Suzuki");

objMap.put("Power", "220");

objMap.put("Type", "2-wheeler");

objMap.put("Price", "85000");

System.out.println("Elements of the Map:");

System.out.println(objMap);

}

}

Output:

Elements of the Map:

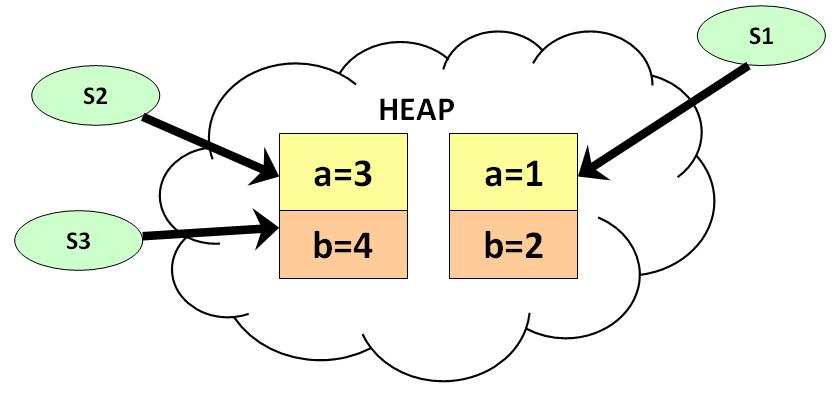
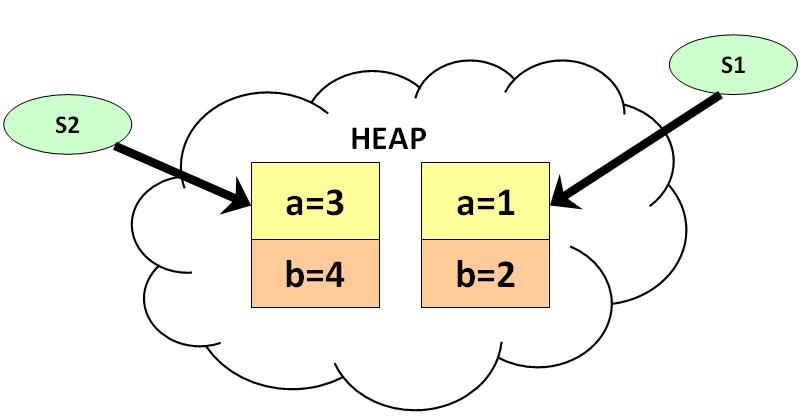
{Type=2-wheeler, Price=85000, Power=220, Name=Suzuki}

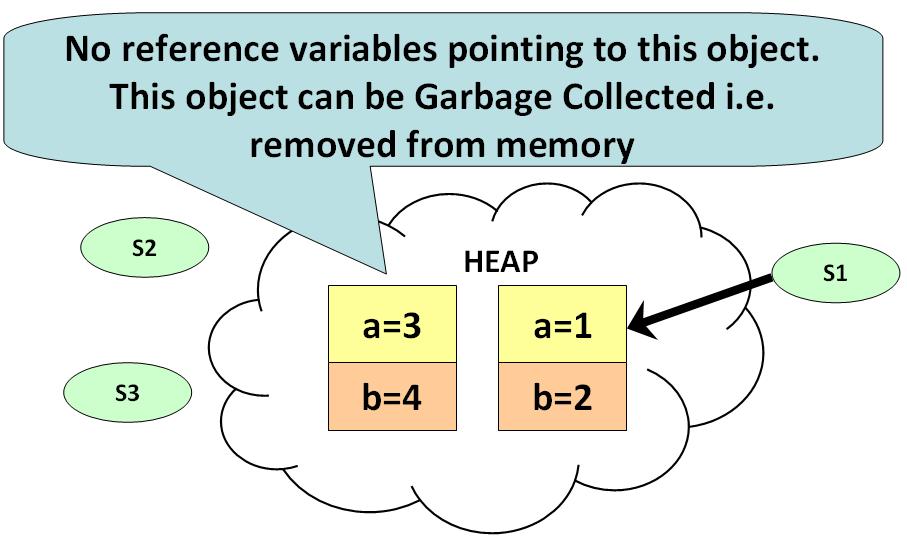
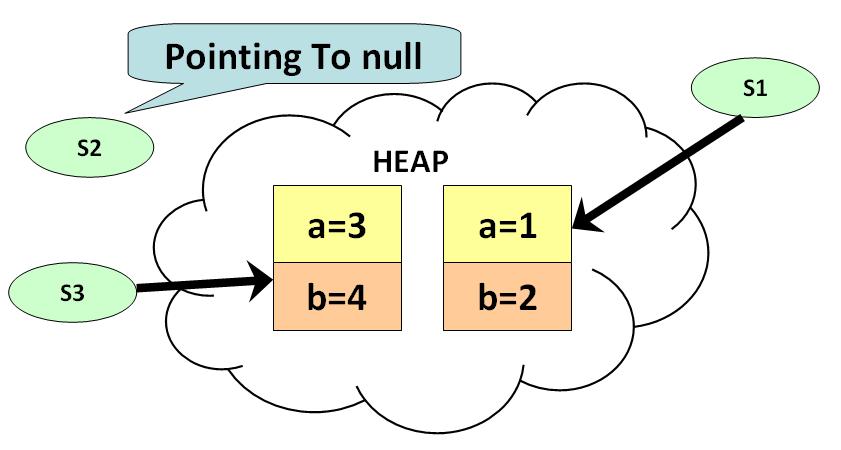
To remove a key value.

map.remove(5);

# Garbage collection:

An object once created uses some memory and the memory remains allocated till there are references for the use of the object. When there are no references to an object, it is assumed to be no longer needed, and the memory, occupied by the object can be reclaimed





# Static Method:

Static method in Java is a method which belongs to the class and not to the object. A static method can access only static data. It is a method which belongs to the class and not to the object(instance). A static method can access only static data. It cannot access non-static data (instance variables).

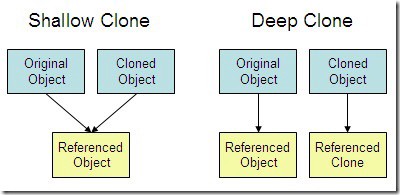
* A static method can call only other static methods and can not call a non-static method from it.
* A static method can be accessed directly by the class name and doesn’t need any object
* A static method cannot refer to "this" or "super" keywords in anyway

# Deep copy and Shallow copy:

Link: <https://www.youtube.com/watch?v=QaCYMgyprtc>

Shallow copies duplicate as little as possible. A shallow copy of a collection is a copy of the collection structure, not the elements. With a shallow copy, two collections now share the individual elements.

Deep copies duplicate everything. A deep copy of a collection is two collections with all of the elements in the original collection duplicated.



# Private Constructor:

In Java, private constructors are used in singleton class. In private constructor, only one object can be created and the object is created within the class and also all the methods are static. An object can not be created if a private constructor is present inside a constructor. A class which have a private constructor and all the methods are static then it is class Utility class.

final class abc

{

private abc()

{}

public static void add(int a, int b)

{

int z = a+b;

System.out.println("Addition: "+z);

}

public static void sub(int x, int y)

{

int z = x-y;

System.out.println("Subtraction: "+z);

}

}

class PrivateConDemo

{

public static void main(String as[])

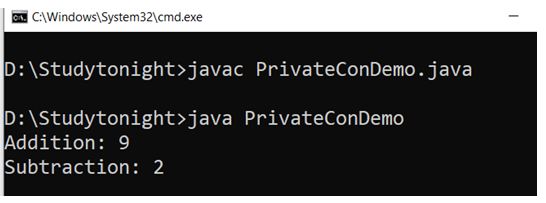
{

abc.add(4, 5);

abc.sub(5, 3);

}

}



# Singleton Class:

In Java, a Singleton class is used when we want to create only one object at a time. To create a singleton class we have to create a private constructor and methods of the class should be static. The concept of Lazy initialization is used for creating a static method. The main difference between a normal class and a singleton class is that in normal class we use constructor and in singleton class, we use **getInstance()** method. The **getInstance()** method creates an object with the name of the class.

class Demo1{

private static Demo1 a = null;

public String s;

private Demo1()

{

s = "Hello Welcome to studytonight.com. You are Reading about Singleton class";

}

public static Demo1 getInstance()

{

if (a== null)

a = new Demo1();

return a;

}

}

class SingletonDemo1

{

public static void main(String args[])

{

Demo1 p = Demo1.getInstance();

Demo1 q = Demo1.getInstance();

Demo1 r = Demo1.getInstance();

p.s = (p.s).toUpperCase();

System.out.println("String from p is " + p.s);

System.out.println("String from q is " + q.s);

System.out.println("String from r is " + r.s);

System.out.println("\n");

r.s = (r.s).toLowerCase();

System.out.println("String from p is " + p.s);

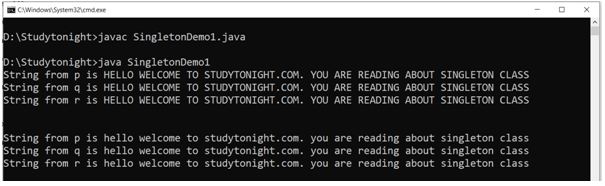
System.out.println("String from q is " + q.s);

System.out.println("String from r is " + r.s);

}

}

Output:

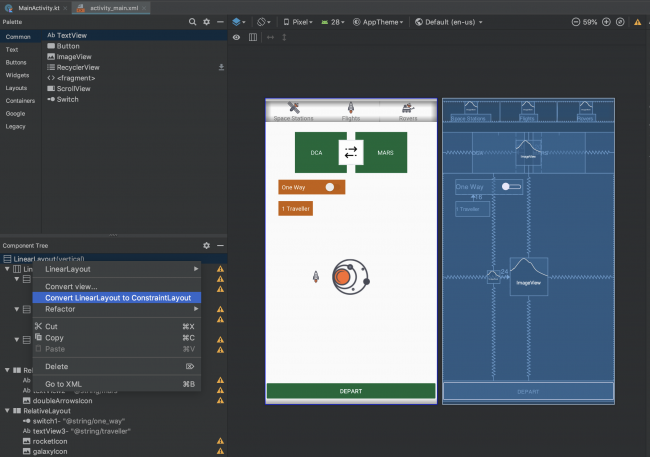


It **is** used where only a single instance of a **class is** required to control the action throughout the execution. A **singleton class** shouldn't have multiple instances in any case and at any cost. **Singleton classes are** used for logging, driver objects, caching and thread pool, database connections.

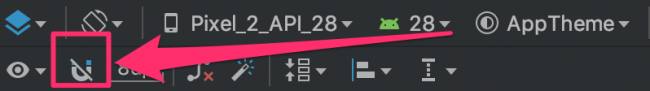
27/04/2020

# Constrained Layout :

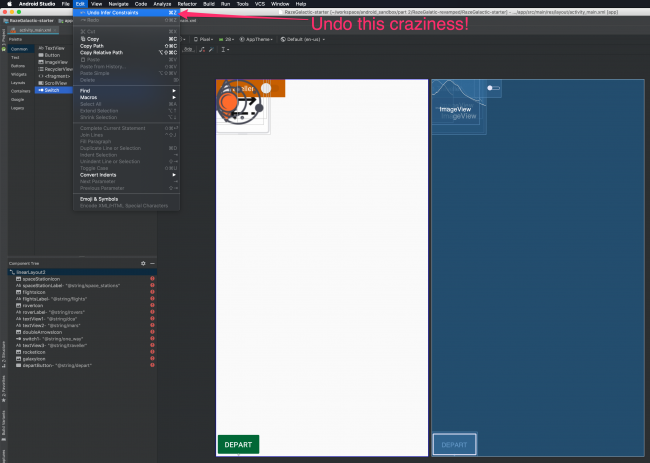
* Convert from other types of layouts to ConstraintLayout.
* Dynamically position UI elements onscreen in relation to other elements.
* Animate your views.

1.We can convert Linear Layout to Constraint Layout 

2. Make sure the auto connect is turned off:



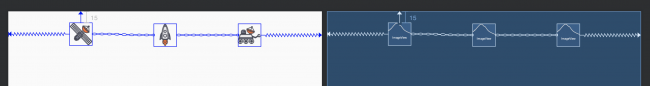
3. During the conversion process, Android Studio performs a number of steps. The last one may have been Infer Constraints, whose results might not quite be what you wanted. ;] If that’s the case, simply go to the Edit menu and choose Undo Infer Constraints or just click ctrl + z



#### 4.Chains:

A chain occurs whenever there is bi-directional constraints. Nothing will specially change in the xml . if there is mutual constraints then we can make a chain.

Shift + select two or more component and with right-click select center > horizontally.



There are three chain modes:

1. Packed

2. Spread

3. Spread inside

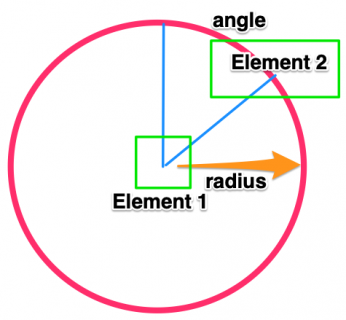
5.Aligning Views

6. Guidelines:

We can add Guidelines from toolbar either horizontal guideline or vertical guideline. The other element can be constrained with respect these guidelines and it helps to create animation using keyframes.

7. Circular Position Constraints:

UI elements can be constrain relative to each other using distance and an angle. This allows us to position them on a circle.



<ImageView

android:id="@+id/rocketIcon"

android:layout\_width="30dp"

android:layout\_height="30dp"

android:src="@drawable/rocket\_icon"

app:layout\_constraintCircle="@id/galaxyIcon"

app:layout\_constraintCircleAngle="270"

app:layout\_constraintCircleRadius="100dp" />

\*\*\* we can animate the constraint layout by copying similar but slightly changed xml file with a different name and it can be animated with toggling the layout with transition or with the constraintSet ( constraint attribute of an element) .

# SharedPreference:

Link: <https://codelabs.developers.google.com/codelabs/android-training-shared-preferences/index.html?index=..%2F..android-training#3>

read that data back in when the app is restarted

private SharedPreferences mPreferences; // member variable to hold a reference of a shared preference  
private String sharedPrefFile = "com.example.android.hellosharedprefs"; // member variable to hold the name of the sharedpref file

The preference file can have different name but it is convenient to keep it same as the package name.

Inside of onCreate() method

mPreferences = getSharedPreferences(sharedPrefFile, *MODE\_PRIVATE*); // initialize the shared pref  
//getSharedPreferences() method opens the pref file named 'sharedPrefFile' and with mode 'MODE\_PRIVATE'

\*\*\*MODE\_PRIVATE is highly recommended.

Now let’s save preferences in onPause()

@Override  
protected void onPause() {  
 super.onPause();  
  
 SharedPreferences.Editor preferencesEditor = mPreferences.edit();  
 //SharedPreferences.Editor is needed to edit the shared pref file  
 //this class includes multiple "put" methods for different data types  
  
 preferencesEditor.putInt(COUNT\_KEY,mCount);// saving the count key-value pair  
 preferencesEditor.putInt(COLOR\_KEY,mColor);// saving the color key-value pair  
  
 preferencesEditor.apply();  
 //this will save the preferences  
 //apply() method saves the preferences asynchronously  
 //commit() method saves the preferences asynchronously  
}

Restore preferences in onCreate()

// Restore the shared preference data  
mCount = mPreferences.getInt(COUNT\_KEY,0);  
mColor = mPreferences.getInt(COLOR\_KEY,mColor);  
//It don't need to get a shared preference editor for reading  
//getInt() take a key and a default value . it returns saved value if found any  
//otherwise it returns default value in this case,0 for count and mColor for color

Now reset the sharedpreference with a button:

SharedPreferences.Editor preferencesEditor = mPreferences.edit();  
//this will needed to edit the mPreferences reference  
preferencesEditor.clear();  
// simple idea is to clear the editor  
preferencesEditor.apply();  
//and apply it to save with empty preference

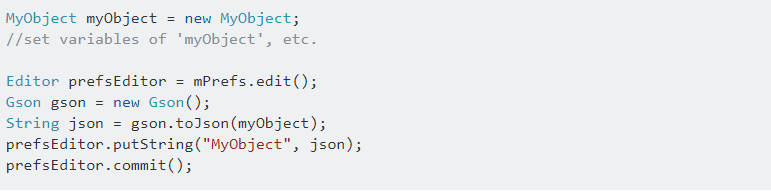
Summery:

* The [SharedPreferences](https://developer.android.com/reference/android/content/SharedPreferences.html) class allows an app to store small amounts of primitive data as key-value pairs.
* Shared preferences persist across different user sessions of the same app.
* To write to the shared preferences, get a [SharedPreferences.Editor](https://developer.android.com/reference/android/content/SharedPreferences.Editor) object.
* Use the various "put" methods in a SharedPreferences.Editor object, such as [putInt()](https://developer.android.com/reference/android/content/SharedPreferences.Editor#putInt(java.lang.String,%20int)) or [putString()](https://developer.android.com/reference/android/content/SharedPreferences.Editor#putString(java.lang.String,%20java.lang.String)), to put data into the shared preferences with a key and a value.
* Use the various "get" methods in a SharedPreferences object, such as [getInt()](https://developer.android.com/reference/android/content/SharedPreferences#getInt(java.lang.String,%20int)) or [getString()](https://developer.android.com/reference/android/content/SharedPreferences#getString(java.lang.String,%20java.lang.String)), to get data out of the shared preferences with a key.
* Use the [clear()](https://developer.android.com/reference/android/content/SharedPreferences.Editor#clear()) method in a SharedPreferences.Editor object to remove all the data stored in the preferences.
* Use the [apply()](https://developer.android.com/reference/android/content/SharedPreferences.Editor#apply()) method in a SharedPreferences.Editor object to save the changes to the preferences file.

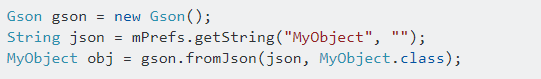
# Store object in sharedreferences:

We can do this by converting the object into a json using (Gson library) and then store the json string in shared preference file.

#### implementation 'com.google.code.gson:gson:2.8.5'



Then we can retrieve the json string of that object and again convert it into Object reference.



28-04-2020

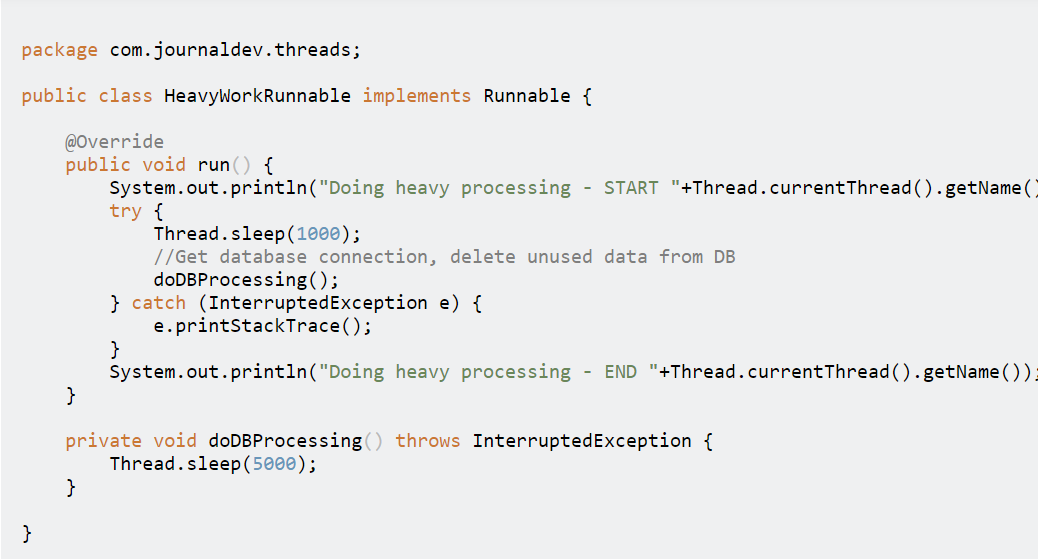
# Multithreading basic:

Every java application has at least one thread – [main thread](https://www.journaldev.com/611/exception-in-thread-main-java).The benefits of java threads:

1. Java Threads are lightweight compared to processes, it takes less time and resource to create a thread.
2. Threads share their parent process data and code
3. Context switching between threads is usually less expensive than between processes.
4. Thread intercommunication is relatively easy than process communication.

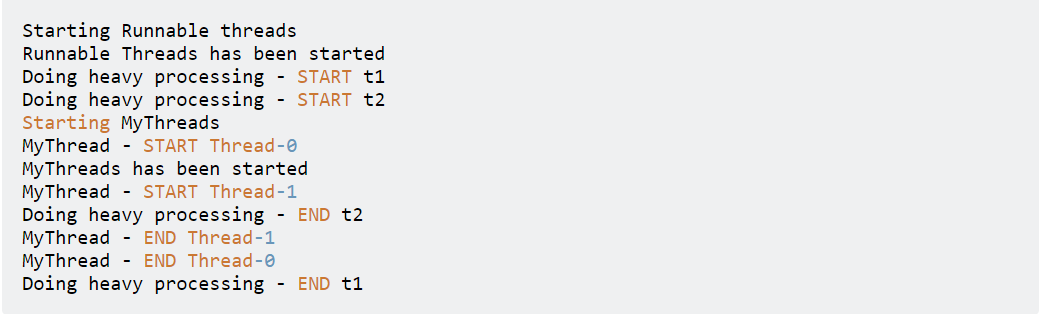
Java provides two ways to create a thread programmatically.

1. Implementing the **java.lang.Runnable** interface.
2. Extending the **java.lang.Thread** class.









Once we start any thread, it’s execution depends on the OS implementation of time slicing and we can’t control their execution. However we can set threads priority but even then it doesn’t guarantee that higher priority thread will be executed first.

If your class provides more functionality rather than just running as Thread, you should implement Runnable interface to provide a way to run it as Thread. If your class only goal is to run as Thread, you can extend Thread class.

Implementing Runnable is preferred because java supports implementing multiple interfaces. If you extend Thread class, you can’t extend any other classes.

## Thread.sleep()

* method can be used to pause the execution of current thread for specified time in milliseconds.
* The actual time thread sleeps before waking up and start execution depends on system timers and schedulers. For a quiet system, the actual time for sleep is near to the specified sleep time but for a busy system it will be little bit more.
* It always pause the current thread execution.
* Any other thread can interrupt the current thread in sleep, in that case InterruptedException is thrown.

## Thread.join()

Using this method we can wait the execution of next thread until the previous thread is finished.

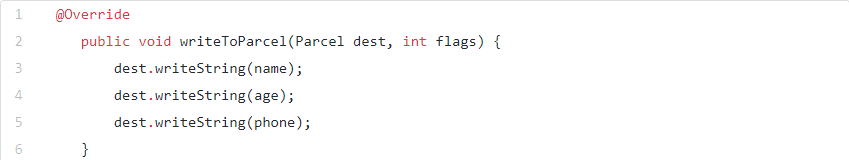
# Parcelable

Link: <https://medium.com/@royanimesh2211/implementing-the-parcelable-interface-in-android-b404819ca441>

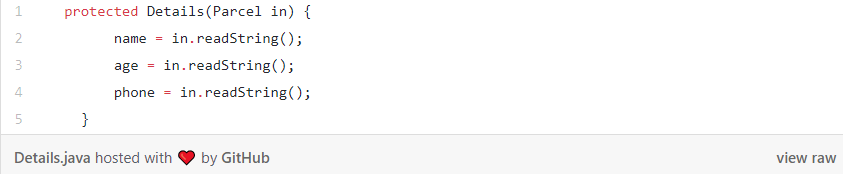
Parcelable is an Android only interface which is used to serialize a class so its properties can be transferred from one activity to another. Parcelable is relatively fast as compared to the java serialization.

## writeToParcel Method:

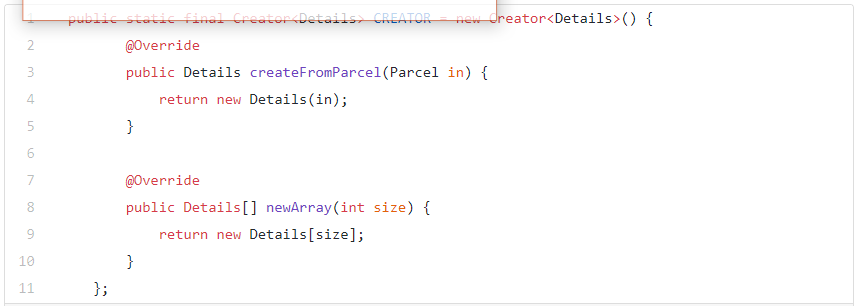
In this method, you need to add all the properties to a parcel which you want to transfer. You use **write** methods to add each of the properties.



Constructor used to reading and saved values from the parcel:



CREATOR used for unparcelling the parcel (creating the object). This method is to bind everything together



describeContents() method:

This method returns the hashcode of the object. This method does not do too much.

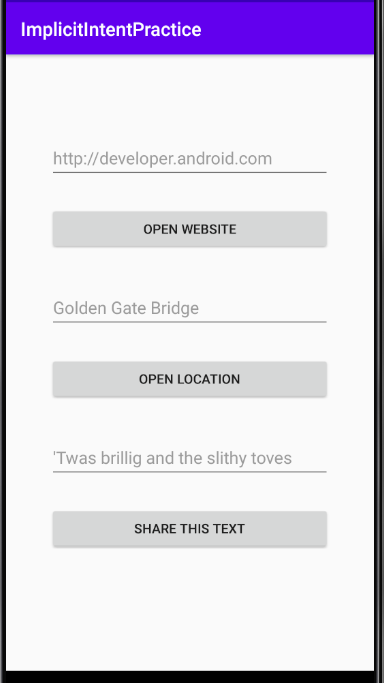
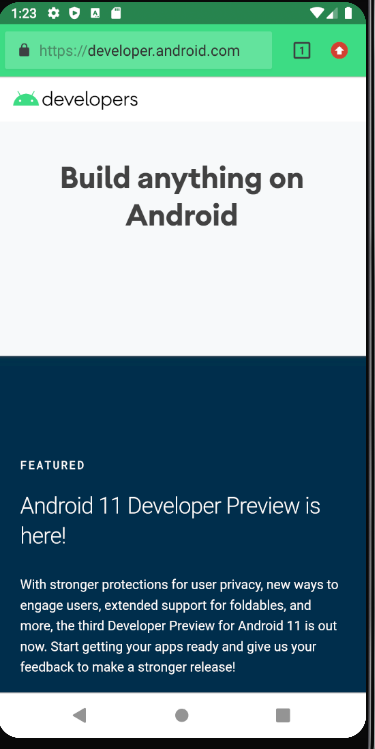
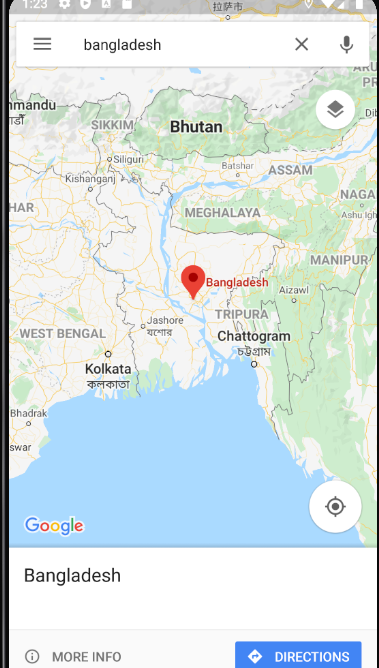
# Implicit intents

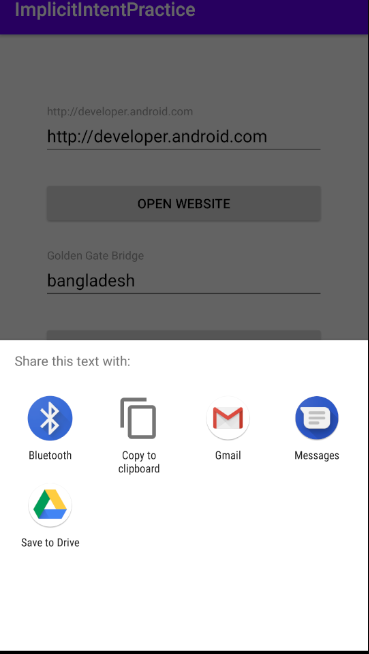
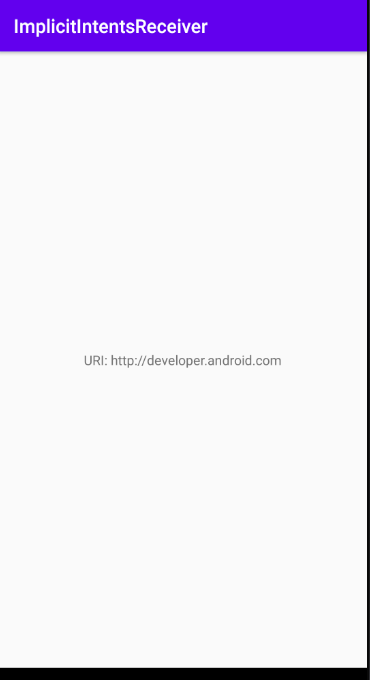
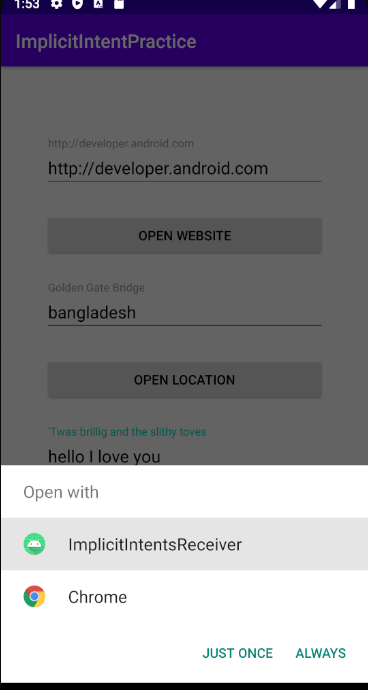
Link: <https://codelabs.developers.google.com/codelabs/android-training-activity-with-implicit-intent/index.html?index=..%2F..%2Fandroid-training&authuser=1#0>

\*\*\*activity can declare one or more intent filters in the AndroidManifest.xml file to advertise that the activity can accept implicit intents, and to define the types of intents that the activity will accept.

To match your request with an app installed on the device, the Android system matches your implicit intent with an activity whose intent filters indicate that they can perform the action. If multiple apps match, the user is presented with an app chooser that lets them select which app they want to use to handle the intent.

App for this:

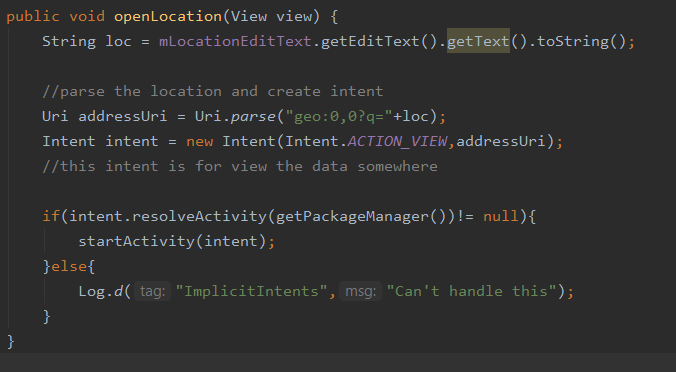
 

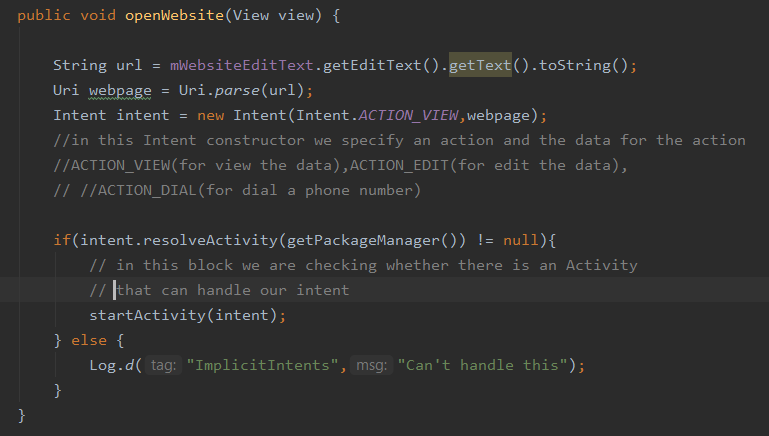
An Activity in your app can always be activated from inside or outside your app with an explicit Intent. To allow an Activity to receive an implicit Intent, you define an Intent filter in your app's AndroidManifest.xml file to indicate which types of implicit Intent your Activity is interested in handling.

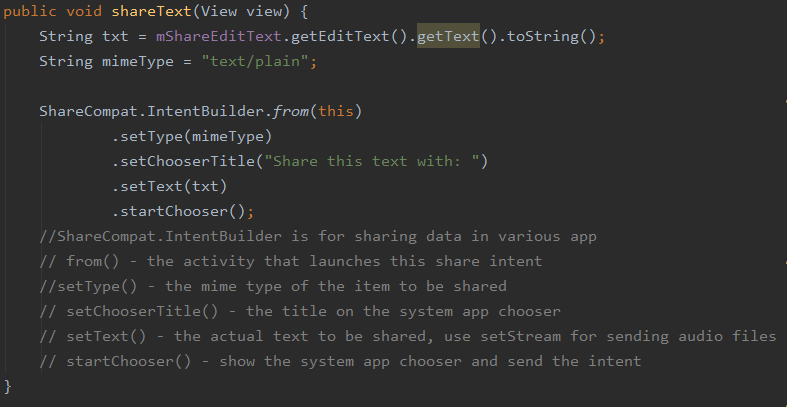
When an app on the device sends an implicit Intent, the Android system matches the action and data of that Intent with any available Activity that includes the right Intent filters. When the Intent filters for an Activity match the Intent:

* If there is only one matching Activity, Android lets the Activity handle the Intent itself.
* If there are multiple matches, Android displays an app chooser to allow the user to pick which app they'd prefer to execute that action.

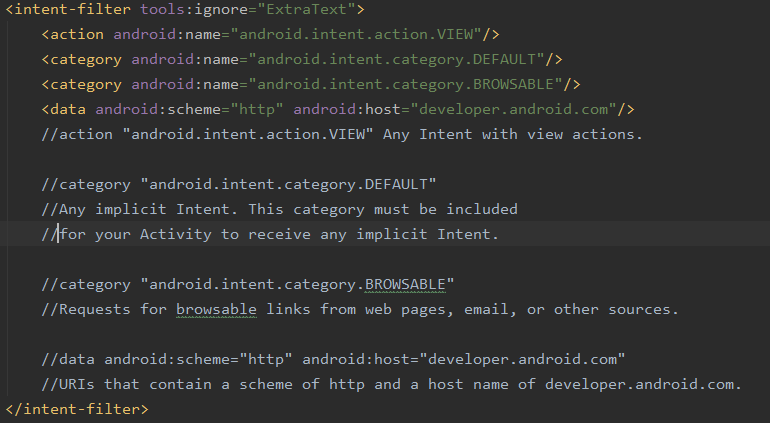
Then for the first app for using implicit intent we can do the code below:



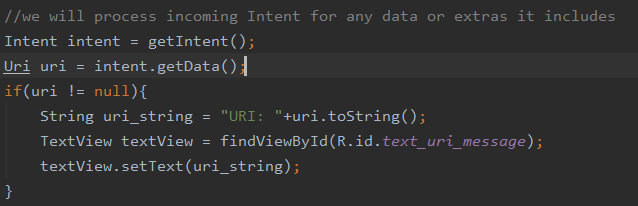




To response to implicit intents like this to app that we create we have to decleare intent filters with their properties in manifest.xml like the below:



To handle the intent from any app and the data we will need to get the intent our mainactivity.java class like the below:



Some common Intents and intent filters:

Link: <https://developer.android.com/guide/components/intents-common>

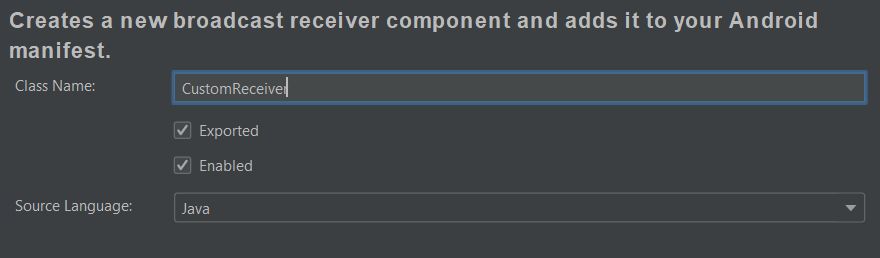
29-04-2020

# Broadcast Receivers:

Broadcasts are messages that the Android system and Android apps send when events occur that might affect the functionality of other apps or app components. For example, the Android system sends a system broadcast when the device boots up, or when headphones are connected or disconnected. If the wired headset is unplugged, you might like your media app to pause the music.

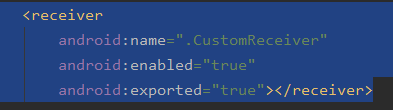
In this practical, you create an app that responds to a change in the charging state of the device. To do this, your app receives and responds to a system broadcast, and it also sends and receives a custom broadcast

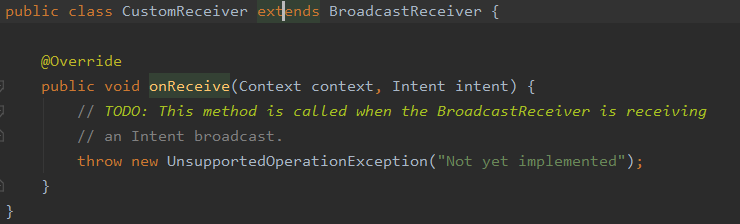
To create a new broadcast receiver, select the package name in the Android Project View and navigate to **File > New > Other > Broadcast Receiver**.



**Exported** allows your broadcast receiver to receive broadcasts from outside your app. **Enabled** allows the system to instantiate the receiver.

This is for static broadcast receiver. But does not support above API level 26 or android 8.0





## **Register receiver for system broadcasts:**

A system broadcast is a message that the Android system sends when a system event occurs. Each system broadcast is wrapped in an Intent object:

-The intent's action field contains event details such as [android.intent.action.HEADSET\_PLUG](https://developer.android.com/reference/android/content/Intent#ACTION_HEADSET_PLUG), which is sent when a wired headset is connected or disconnected.

A [BroadcastReceiver](https://developer.android.com/reference/android/content/BroadcastReceiver) is either a *static receiver* or a *dynamic receiver*, depending on how you register it:

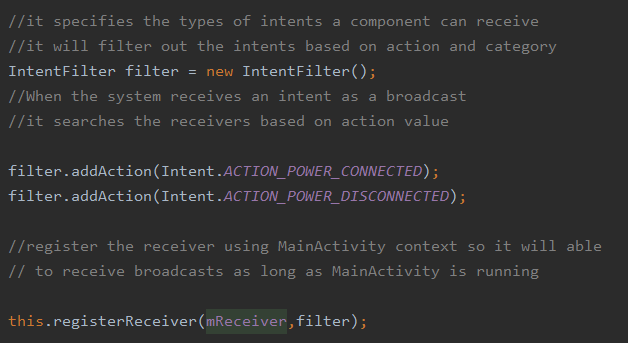
* To register a receiver statically, use the <receiver> element in your AndroidManifest.xml file. Static receivers are also called *manifest-declared receivers*.
* To register a receiver dynamically, use the app context or activity context. The receiver receives broadcasts as long as the registering context is valid, meaning as long as the corresponding app or activity is running. Dynamic receivers are also called *context-registered receivers*.

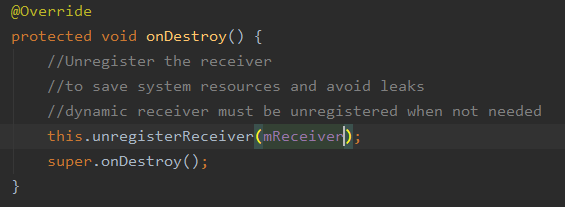
\*\*\*Starting from Android 8.0 (API level 26 and higher), you can't use static receivers to receive most Android system broadcasts, with some [exceptions](https://developer.android.com/guide/components/broadcast-exceptions). So for this task, you use dynamic receivers

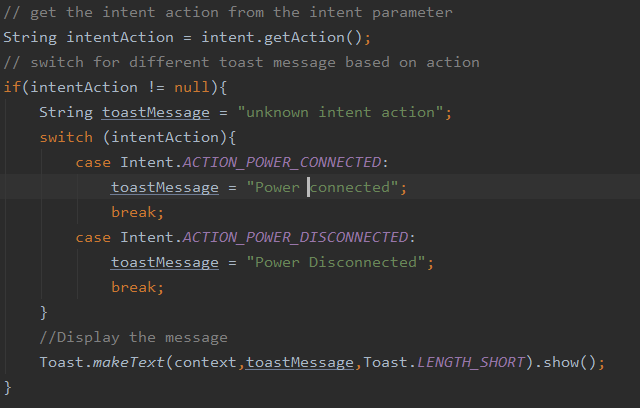
# Implementation of dynamic broadcast receiver for system broad cast:

* Remove the <receiver> element from AndroidManifest.xml
* Create a CustomReceiver object and initialize it
* Create and IntentFilter object and initialize it in onCreate() method
* Add the actions to the intentFilter object
* Register and unregister the receiver
* Implement onReceive() in BroadcastReceiver









# Implement send and receive a custom broadcast:

Use a custom broadcast when you want your app to take an action without launching an activity, for example when you want to let other apps know that data has been downloaded to the device.

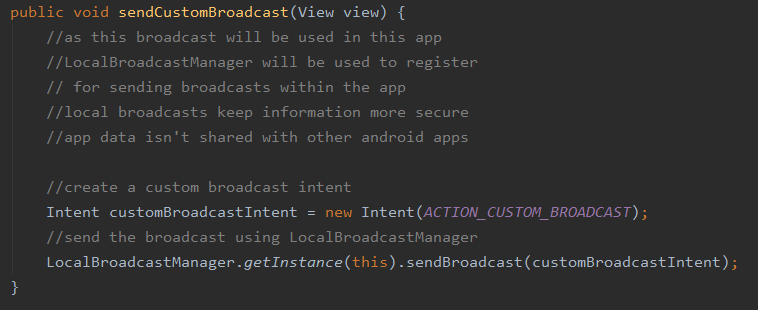
Android provides three ways for your app to send custom broadcasts:

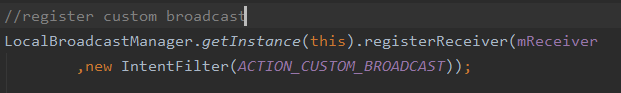
* *Normal broadcasts* are asynchronous. Receivers of normal broadcasts run in an undefined order, often at the same time. To send a normal broadcast, create a broadcast intent and pass it to [sendBroadcast(Intent)](https://developer.android.com/reference/android/content/Context.html#sendBroadcast(android.content.Intent)).
* *Local broadcasts* are sent to receivers that are in the same app as the sender. To send a local broadcast, create a broadcast intent and pass it to [LocalBroadcastManager.sendBroadcast](https://developer.android.com/reference/android/support/v4/content/LocalBroadcastManager.html#sendBroadcast(android.content.Intent)).
* *Ordered broadcasts* are delivered to one receiver at a time. As each receiver executes, it can propagate a result to the next receiver, or it can cancel the broadcast so that the broadcast is not passed to other receivers. To send an ordered broadcast, create a broadcast intent and pass it to [sendOrderedBroadcast(Intent, String)](https://developer.android.com/reference/android/content/Context.html#sendOrderedBroadcast(android.content.Intent,%20java.lang.String)).

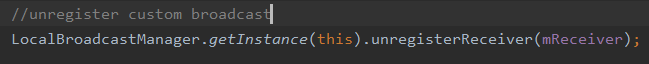
# Procedure:

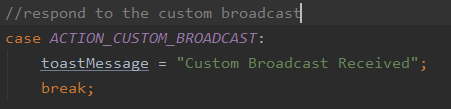
* Define custom broadcast action string.prepending action name with app’s package name.to get package name use Buildconfig.APPLICATION\_ID. So create a constant member variable in both MainActivity and CustomReceiver class.
* **Create ‘sendCustomBroadcast (View)' in 'MainActivity'.** And implement the method
  + **Create a new intent with custom broadcast action string**
  + **Send the broadcast using LocalBroadcastManager**
* Register and unregister the custom broadcast using LocalBroadcastManager
* Respond to the custom broadcast from customBroadcast.java class





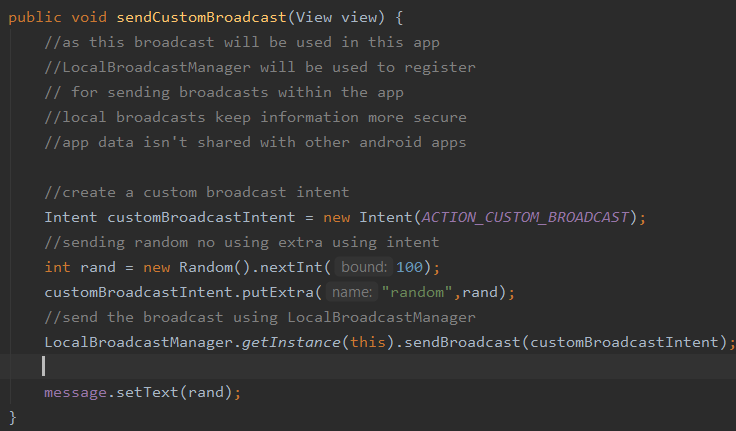


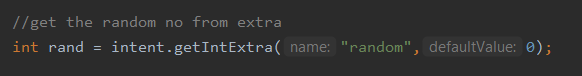




# Sending data using broadcast

* Use intent.putExtra() to put data in the intent
* In the receiver class get the extra from the intent and do operation on it





# Background Services:

Link: <https://developer.android.com/guide/components/services?authuser=1>

A [Service](https://developer.android.com/reference/android/app/Service?authuser=1) 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. Additionally, a component can bind to a service to interact with it and even perform interprocess communication (IPC). For example, a service can handle network transactions, play music, perform file I/O, or interact with a content provider, all from the background.

These are the three different types of services:

* Foreground: A foreground service performs some operation that is noticeable to the user. For example, an audio app would use a foreground service to play an audio track. Foreground services must display a [Notification](https://developer.android.com/guide/topics/ui/notifiers/notifications?authuser=1). Foreground services continue running even when the user isn't interacting with the app.

\*\*\* using WorkManager is preferable to using foreground services directly.

* Background: A background service performs an operation that isn't directly noticed by the user. For example, if an app used a service to compact its storage, that would usually be a background service.
* Bound: A service is bound when an application component binds to it by calling [bindService()](https://developer.android.com/reference/android/content/Context?authuser=1#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.

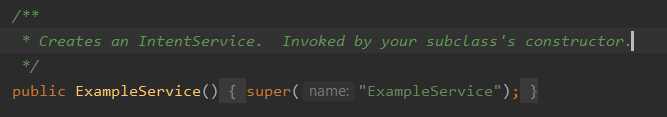
## Service vs thread:

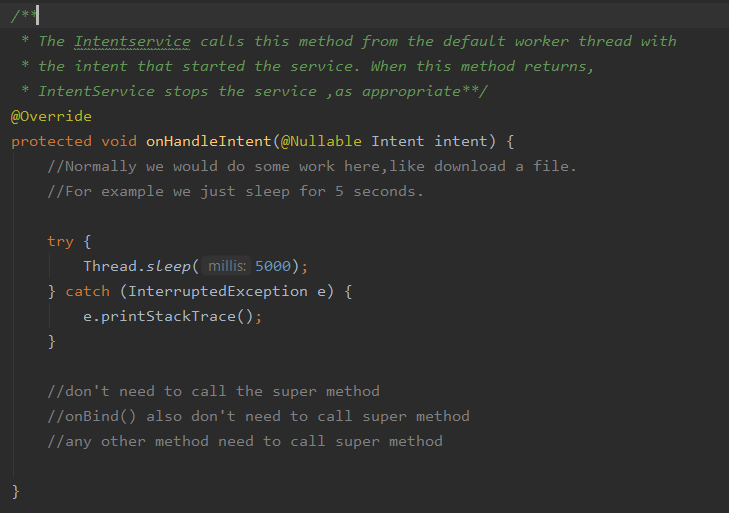
A service is simply a component that can run in the background, even when the user is not interacting with your application, so you should create a service only if that is what you need.

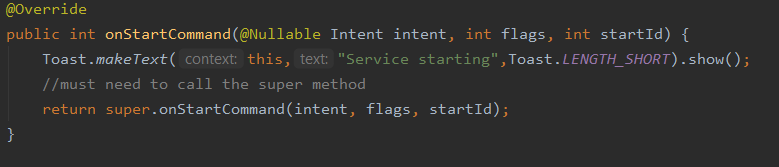
If you must perform work outside of your main thread, but only while the user is interacting with your application, you should instead create a new thread.

## Procedure:

* Extend one of the following to create a started service:
  + **Service:** Base class for all services. Must have to create a new thread in which the service can complete all of it’s work. Otherwise it will use the main thread that will slow down application.
  + **IntentService:** This is a subclass of Service that uses a worker thread to handle all of the start requests, one at a time. This is the best option if service don’t handle multiple request simultaneously. It is preferred option
* override some callback methods that handle key aspects of the service lifecycle and provide a mechanism that allows the components to bind to the service
  + [OnStartCommand ()](https://developer.android.com/reference/android/app/Service?authuser=1#onStartCommand(android.content.Intent,%20int,%20int)) - The system invokes this method by calling [startService ()](https://developer.android.com/reference/android/content/Context?authuser=1#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.  to stop the service when its work is complete by calling [stopSelf()](https://developer.android.com/reference/android/app/Service?authuser=1#stopSelf()) or [stopService()](https://developer.android.com/reference/android/content/Context?authuser=1#stopService(android.content.Intent))
  + [onBind()](https://developer.android.com/reference/android/app/Service?authuser=1#onBind(android.content.Intent)) - The system invokes this method by calling [bindService()](https://developer.android.com/reference/android/content/Context?authuser=1#bindService(android.content.Intent,%20android.content.ServiceConnection,%20int)) when another component wants to bind with the service (such as to perform RPC). provide an interface to communicate with the service by returning an [IBinder](https://developer.android.com/reference/android/os/IBinder?authuser=1).
  + [onCreate()](https://developer.android.com/reference/android/app/Service?authuser=1#onCreate()) - The system invokes this method to perform one-time setup procedures when the service is initially created. If the service is already running, this method is not called.
  + [onDestroy()](https://developer.android.com/reference/android/app/Service?authuser=1#onDestroy()) - The system invokes this method when the service is no longer used and is being destroyed.
* Declare the service in the manifest
* To start the service,use an explicit intent with [startService()](https://developer.android.com/reference/android/content/Context?authuser=1#startService(android.content.Intent)).

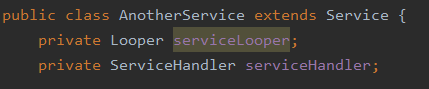




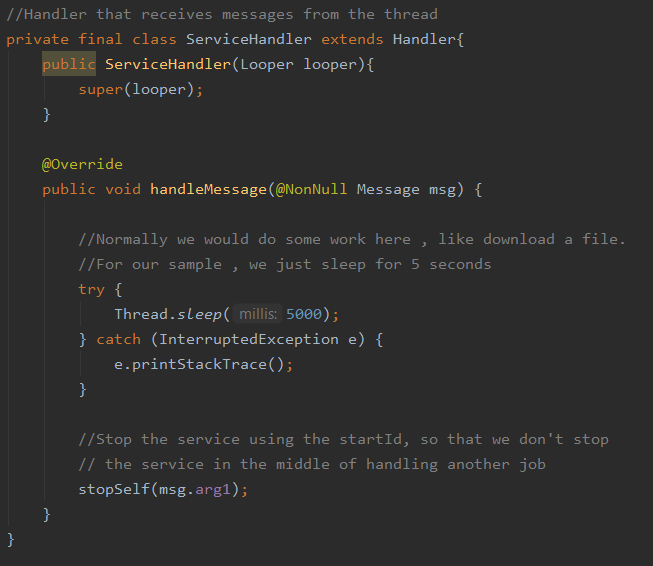


Using [IntentService](https://developer.android.com/reference/android/app/IntentService?authuser=1) makes your implementation of a started service very simple. If, however, you require your service to perform multi-threading (instead of processing start requests through a work queue), you can extend the [Service](https://developer.android.com/reference/android/app/Service?authuser=1) class to handle each intent.

Create a member object of Looper to loop through the thread messages. And a user defined ServiceHandler that extends Handler to handle multi-threading.



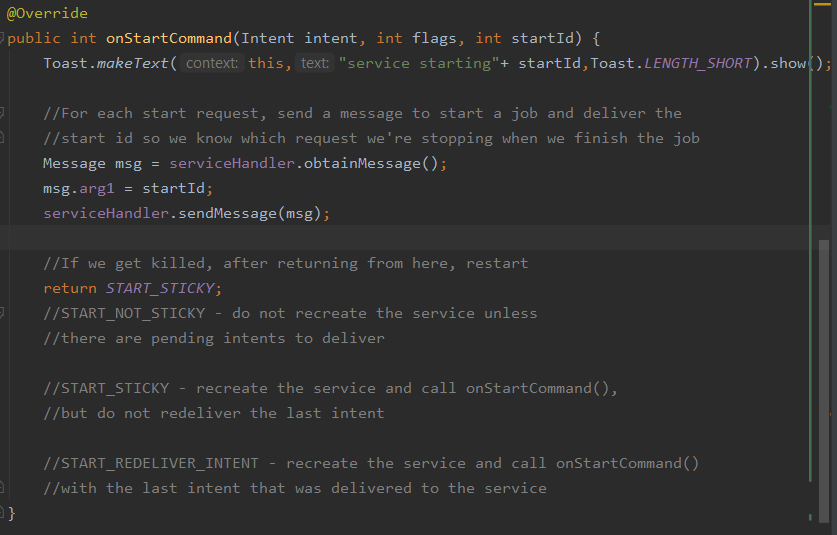
ServiceHandler class that extends Handler will receive the message for thread and will do some work in it’s handleMessage method. This is needed because Service class runs on the main thread that we don’t want . So we have to run the processes in different threads.



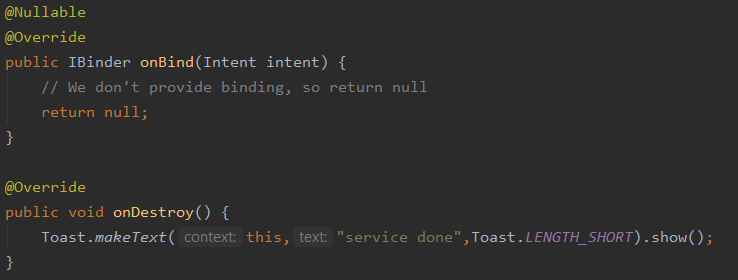
Create thread for the processing and get the thread’s Looper and send it to the ServiceHandler



After sending the Looper to the Handler in onCreate() method, onStartCommand is called and here we create message to start a job with startId so we can know which job is we are killing after finish.



onDestroy() will called after stopSelf(msg.arg1) is executed on the handleMessage method of Handler class



## When the service killed:

* Memory is low and to recover system resources for other focused activity
* If the service is bound to a focused activity it’s less likely to be killed
* If the service is decleared to run in the foreground it’s rarely killed
* After long running of the service it becomes highly susceptible to killing

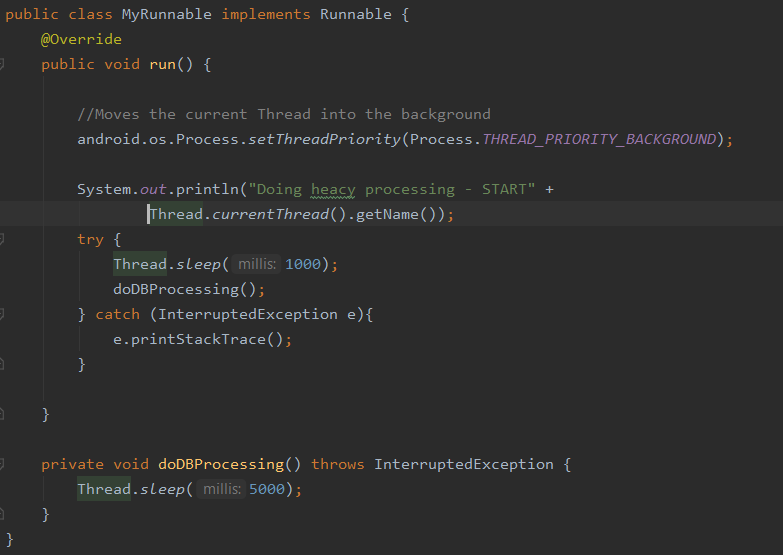
Life Cycle of the service and bounds



# Thread continues:

 set up and use multiple threads in an Android app, using a thread pool object. how to define code to run on a thread and how to communicate between one of these threads and the UI thread.

## Runnable thread:



## Create a manager for multiple thread:

If you want to run a task repeatedly on different sets of data, but you only need one execution running at a time, an [IntentService](https://developer.android.com/reference/android/app/IntentService?authuser=1) suits your needs. To automatically run tasks as resources become available, or to allow multiple tasks to run at the same time (or both), you need to provide a managed collection of threads. To do this, use an instance of [ThreadPoolExecutor](https://developer.android.com/reference/java/util/concurrent/ThreadPoolExecutor?authuser=1), which runs a task from a queue when a thread in its pool becomes free. To run a task, all you have to do is add it to the queue.

 Enclose variables that can be accessed by more than one thread in a synchronized block. This approach will prevent one thread from reading the variable while another is writing to it. Typically, this situation arises with static variables, but it also occurs in any object that is only instantiated once.