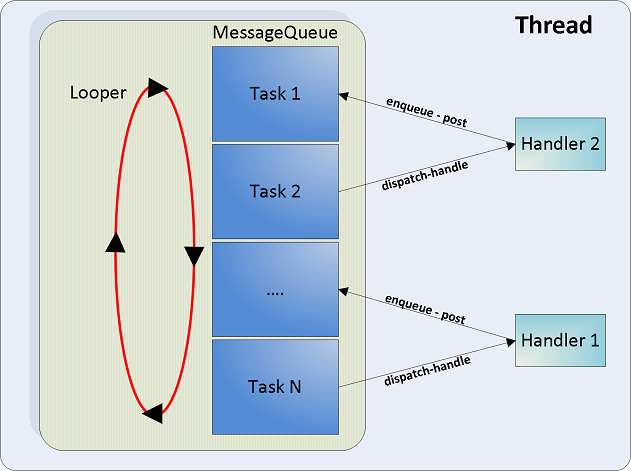
**Looper, Handler, HandlerThread, MessageQueue, Message**

**Looper**

A [Looper](http://developer.android.com/reference/android/os/Looper.html) is a message handling loop: it reads and processes items from a [MessageQueue](http://developer.android.com/reference/android/os/MessageQueue.html). The Looper class is usually used in conjunction with a [HandlerThread](http://developer.android.com/reference/android/os/HandlerThread.html) (a subclass of Thread).

A [Handler](http://developer.android.com/reference/android/os/Handler.html) is a utility class that facilitates interacting with a Looper—mainly by posting messages and Runnable objects to the thread's MessageQueue. When a Handler is created, it is bound to a specific Looper (and associated thread and message queue).



1. ***MessageQueue*** is a queue that has tasks called messages which should be processed.
2. ***Handler*** enqueues task in the MessageQueue using Looper and also executes them when the task comes out of the MessageQueue.
3. ***Looper*** is a worker that keeps a thread alive, loops through MessageQueue and sends messages to the corresponding handler to process.
4. Finally Thread gets terminated by calling Looper’s quit() method.

One thread can have only one unique Looper and can have many unique Handlers associated with it.

1. ***Message***: It is a class that defines various useful methods to deal with message data. To send an object we set the **obj** variable. Message is data or work (runnable) offloaded from, for example, UI thread.

Message msg = **new** Message();

msg.**obj** = **"Ali send message"**;

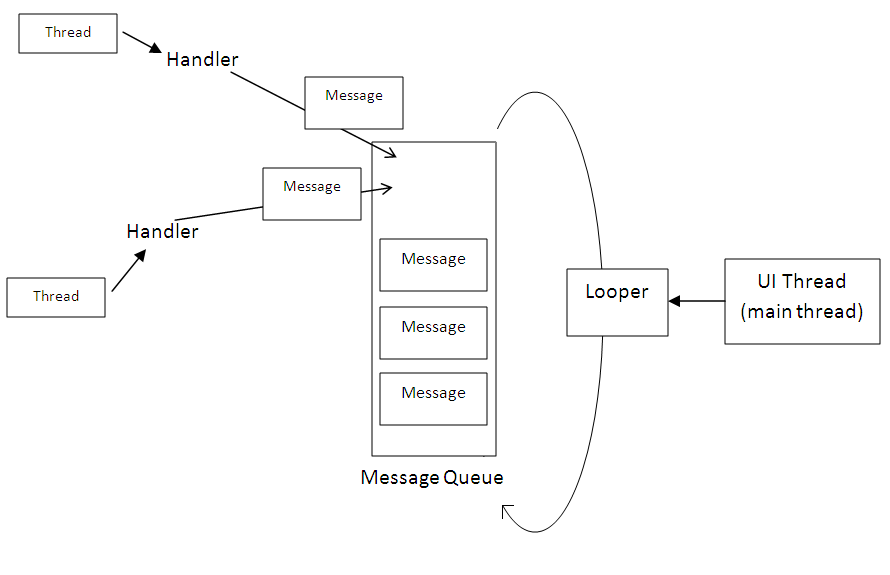
**handler**.sendMessage(msg);

***MessageQueue***: It is a low-level class holding the list of messages to be dispatched by a Looper. Messages are not added directly to a MessageQueue, but rather through Handler objects associated with the Looper.[[3](http://developer.android.com/reference/android/os/MessageQueue.html)]

***Looper***: It loops over a MessageQueue which contains the messages to be dispatched. The actual task of managing the queue is done by the Handler which is responsible for handling (adding, removing, dispatching) messages in the message queue.[[2](http://riteattitude.blogspot.in/2011/11/android-ui-thread.html)]

***Handler***: Handler is the message processor on the worker thread.

It allows you to send and process Message and Runnable objects associated with a thread's MessageQueue. Each Handler instance is associated with a single thread and that thread's message queue.[[4](http://developer.android.com/reference/android/os/Handler.html)]



**HandlerThread**

**Using HandlerThread in Android**

Android offers high level multi-threading options, such as AsyncTask and IntentService to offload long running blocking tasks to a separate thread.

HandlerThread is a low level threading alternative.

HandlerThread allows you to run a worker thread with a single message queue.

It becomes very handy when

* You want a really light weight alternative to run some very simple tasks, where AsyncTask and IntentService become an overkill, such as communicate with camera or accelerometer.
* You want to do some short and simple tasks during the life cycle of an activity or fragment, such as file or database operation. It is worth noting that AsyncQueryHandler class provided in Android framework uses HandlerThread to make database CRUD operations asynchronous.
* You need some simple tasks to be processed sequentially on a worker thread.

Message passing between Threads

M1

M2

M3

M4

**Handler**

**M5**

Message Queue

Append Message to MessageQueue

**Looper**

Retrieve Message

Dispatch to handler for processing

**Message Passing in Android**

Handler is the message processor on the worker thread. Message is data or work (runnable) offloaded from, for example, UI thread.

When a message is received on Handler, it appends the message to the Message Queue. Each thread can have only one MessageQueue.

Looper is a message dispatcher which takes one message at a time from the Message Queue and dispatch it to the Handler for processing.

**Avoid Memory Leak**

One thing you should watch out for when using HandlerThread or in fact any multithreading technique is memory leak.

Avoid using non-static inner classes in activity. Non-static inner class will have an implicit reference to the hosting activity and will stop the activity from being destroyed.

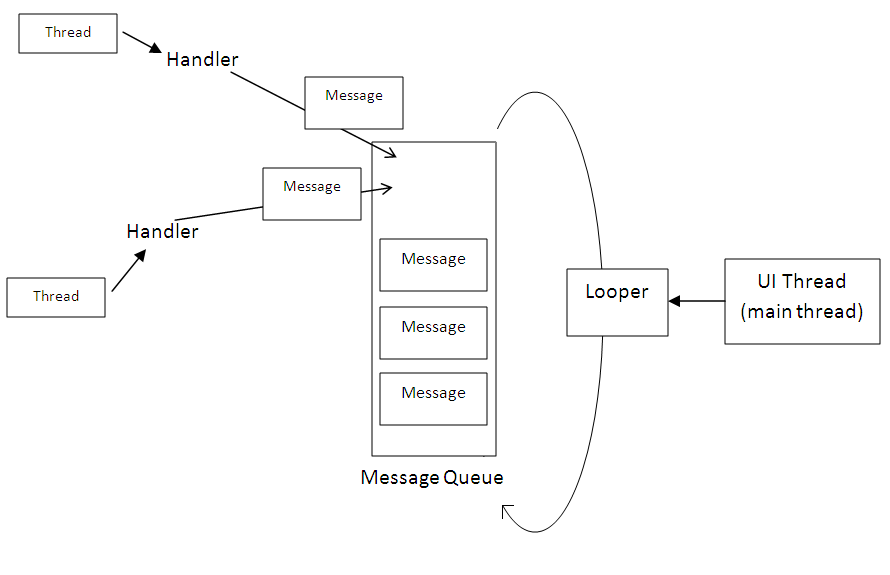
If you really need to have a reference to the activity, use a WeakReference.

**Avoid Mixing Long Running Tasks with Short Ones**

Because HandlerThread has only one message queue and all messages are processed sequentially, long running blocking tasks will make all short running tasks waiting. A better way is to separate long running tasks to a separate HandlerThread or use other threading techniques.

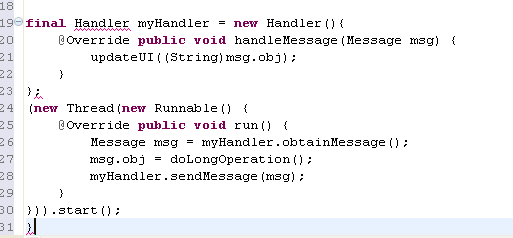
**CONCEPT:**  
In almost every OS, for UI updating, there is a UI thread, which runs an infinite loop and checks a queue to see if there are any pending UI events.  
In case of Android, above concept is implemented with help of:   
1. [Looper](http://developer.android.com/reference/android/os/Looper.html)

2. [MessageQueue](http://www.blogger.com/reference/android/os/MessageQueue.html)

3. [Messages](http://developer.android.com/reference/android/os/Message.html)  
4. [Handler](http://developer.android.com/reference/android/os/Handler.html)  
The Looper loops over a [MessageQueue](http://www.blogger.com/reference/android/os/MessageQueue.html) which contains the [messages](http://developer.android.com/reference/android/os/Message.html) to be dispatched. The actual task of managing the queue is done by a [Handler](http://developer.android.com/reference/android/os/Handler.html) which is responsible for handling (adding, removing, dispatching) messages in the message queue.  


Looper and handler:   
Looper can be associated only with one thread and association cannot be changed.  
Multiple handlers can be associated with a message queue.  
Blocking operation on the UI thread leads to Application not responding (ANR) dialog.

Handler provide bridge for new threads to add messages to messages queue.



*Other methods for accessing message queue:*

handler.sendMessageAtFrontOfQueue ()

handler.sendMessageAtTime ()

handler.sendMessageDelayed ()

handler.sendEmptyMessage ()

**WeakReference Concepts:**

The reason you need a weak reference is so that the Garbage Collector can dispose of the objects when they are no longer needed. If two objects retain a strong reference to each other, then they can't be garbage collected. This is a memory leak.