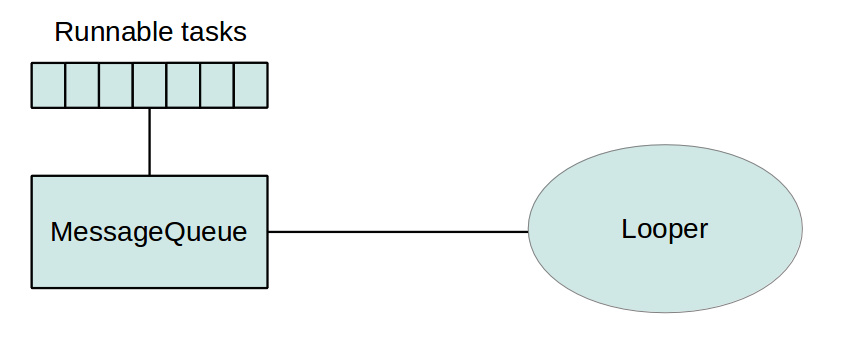
**Main thread**

Android modifies the user interface and handles input events from one single *user interface thread*. This thread is also called the *main thread*.

Android collects all events in a queue and processed an instance of the Looper class.



### Why using concurrency?

* If the programmer does not use any concurrency constructs, all code of an Android application runs in the main thread and every statement is executed after each other.
* If you perform a long lasting operation, for example accessing data from the Internet, the application blocks until the corresponding operation has finished.
* To provide a good user experience all potentially slow running operations in an Android application should run asynchronously, e.g. via some way of concurrency constructs of the Java language or the Android framework. This includes all potential slow operations, like network, file and database access and complex calculations.

## Using Java threading constructs in Android

* Android supports the usage of the Thread class to perform asynchronous processing.
* Android also supplies the java.util.concurrent package to perform something in the background, e.g. using the ThreadPools and Executorclasses.
* If you need to update the user interface from a new Thread, you need to synchronize with the user interface thread.

### Disadvantages of using Java threads in Android

If you use Java threading in Android applications, you have to handle the following requirements in your custom code:

* Synchronization with the main thread, if you post back results to the user interface
* No default for canceling the thread
* No default thread pooling
* No default for handling configuration changes in Android

Because of these restrictions, Android developer typically use Android specific code constructs.

## Background processing and lifecycle handling

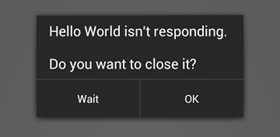
### Retaining state during configuration changes

* One challenge in using threads is to consider the lifecycle of the application. The Android system may kill your activity or trigger a configuration change which will also restart your activity.
* You also need to handle open dialogs, as dialogs are always connected to the activity which created them. In case the activity gets restarted and you access an existing dialog you receive a View not attached to window manager exception.
* To save an object you can use the method onRetainNonConfigurationInstance() method. This method allows you to save one object if the activity will be soon restarted.
* To retrieve this object you can use the getLastNonConfigurationInstance() method. This way can you can save an object, e.g. a running thread, even if the activity is restarted.
* getLastNonConfigurationInstance() returns null if the activity is started the first time or if it has been finished via the finish() method.
* onRetainNonConfigurationInstance() is deprecated as of API 13, it is recommended that you use fragments and the setRetainInstance() method to retain data over configuration changes.

##### **ANDROID UI THREAD AND ANR**

On the Android platform, applications operate, by default, on one thread.  This thread is called the UI thread.  It is often called that because this single thread displays the user interface and listens for events that occur when the user interacts with the app.

Developers quickly learn that if code running on that thread hogs that single thread and prevents user interaction (for more than 5 seconds), it causes Android to throw up the infamous Android Not Responsive (ANR) error.

[](http://cdn.intertech.com/Blog/wp-content/uploads/2014/05/anr.png)

##### **MANY NON-UI TO UI THREAD COMMUNICATION OPTIONS**

Well, as it turns out, there are several ways to have non-UI threads request updates to the UI through the UI thread.  In fact, in the next posts, I plan to show you five ways to have the non-UI thread send UI update requests to be executed on the UI thread.

1. Use runOnUiThread( ) method call
2. Use post( ) method call
3. Use the Handler framework
4. Use a Broadcasts and BroadcastReceiver (optionally with LocalBroadcastManager)
5. Use an AsyncTask’s onProgressUpdate( ) method

As with all options, there are considerations when making a selection from this list.  Much depends on your design decisions about how/where the non-UI thread is created and launched.

