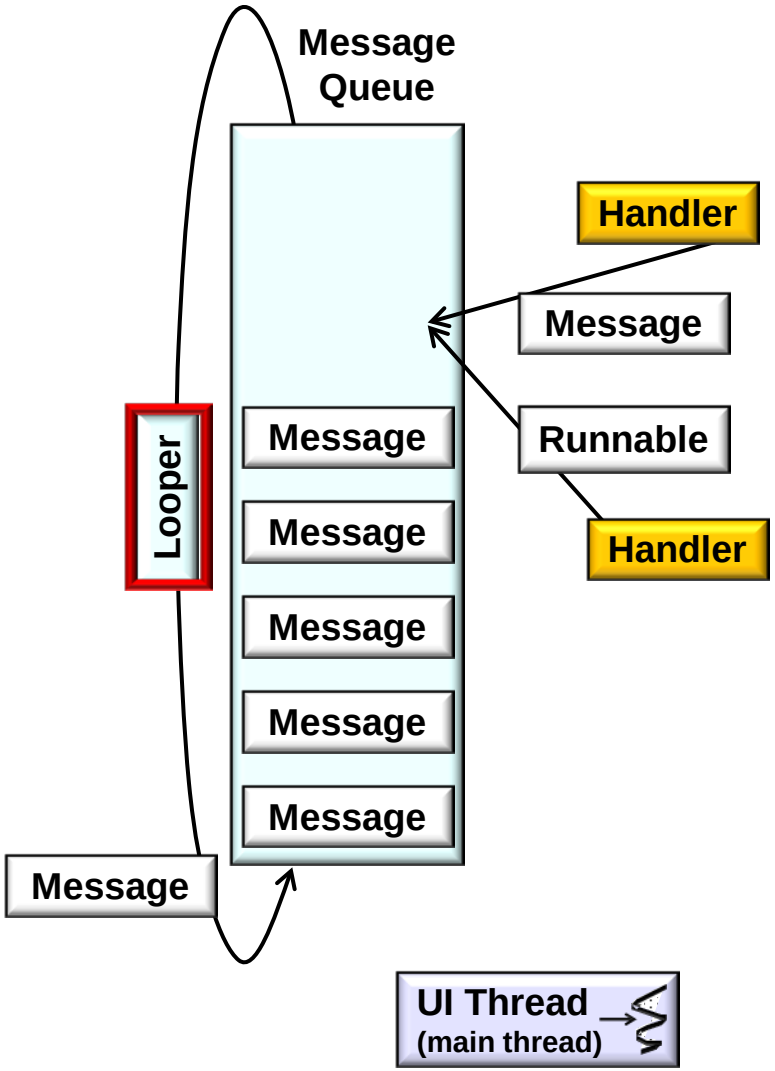
Android Concurrency: Android Looper

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA

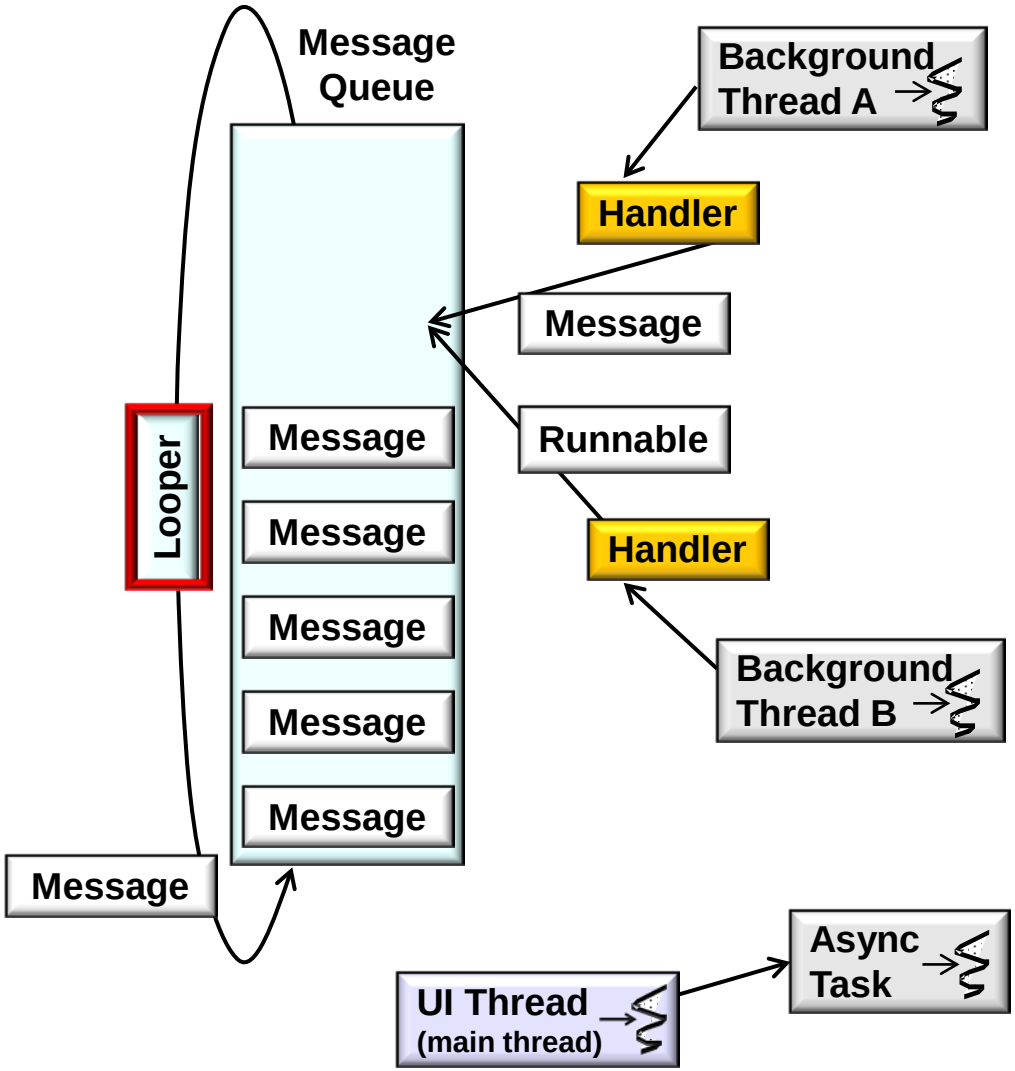
Learning Objectives in this Part of the Module

- Understand how an Android Looper provides a Thread-specific event loop that waits for & dispatches Messages to handlers



Learning Objectives in this Part of the Module

- Understand how an Android Looper provides a Thread-specific event loop that waits for & dispatches Messages to handlers
- Recognize how Loopers are applied in Android applications & concurrency frameworks



The Looper Class

- A Looper has a synchronized MessageQueue that's used to process Messages placed on the queue

Looper

Added in API level 1

extends Object

java.lang.Object
↳ android.os.Looper

Class Overview

Class used to run a message loop for a thread. Threads by default do not have a message loop associated with them; to create one, call `prepare()` in the thread that is to run the loop, and then `loop()` to have it process messages until the loop is stopped.

Most interaction with a message loop is through the `Handler` class.

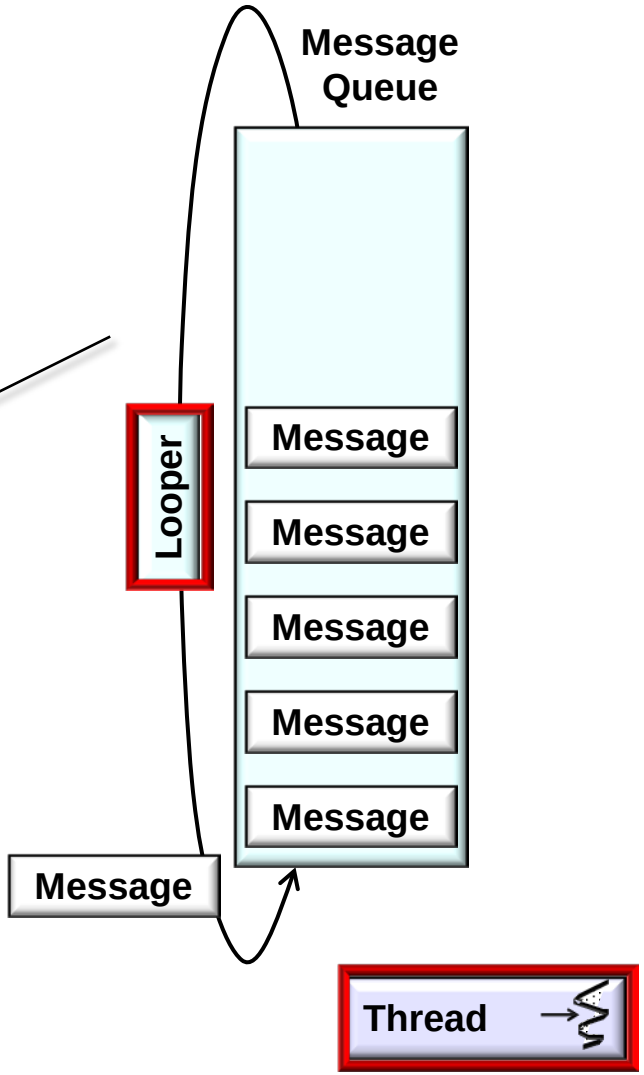
This is a typical example of the implementation of a Looper thread, using the separation of `prepare()` and `loop()` to create an initial Handler to communicate with the Looper.

developer.android.com/reference/android/os/Looper.html has more info

The Looper Class

- A Looper has a synchronized MessageQueue that's used to process Messages placed on the queue
  - It implements a Thread-specific *event loop*

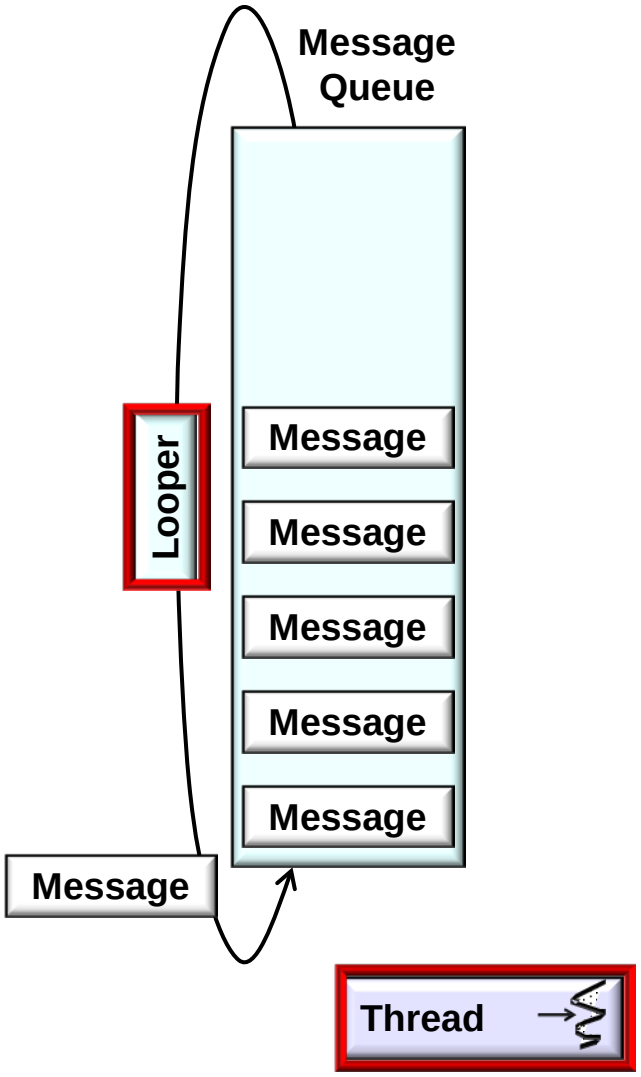
An event loop is a key portion of an event-driven programming model where the control flow of a thread is determined by messages it receives



en.wikipedia.org/wiki/Event_loop & en.wikipedia.org/wiki/Event-driven_programming

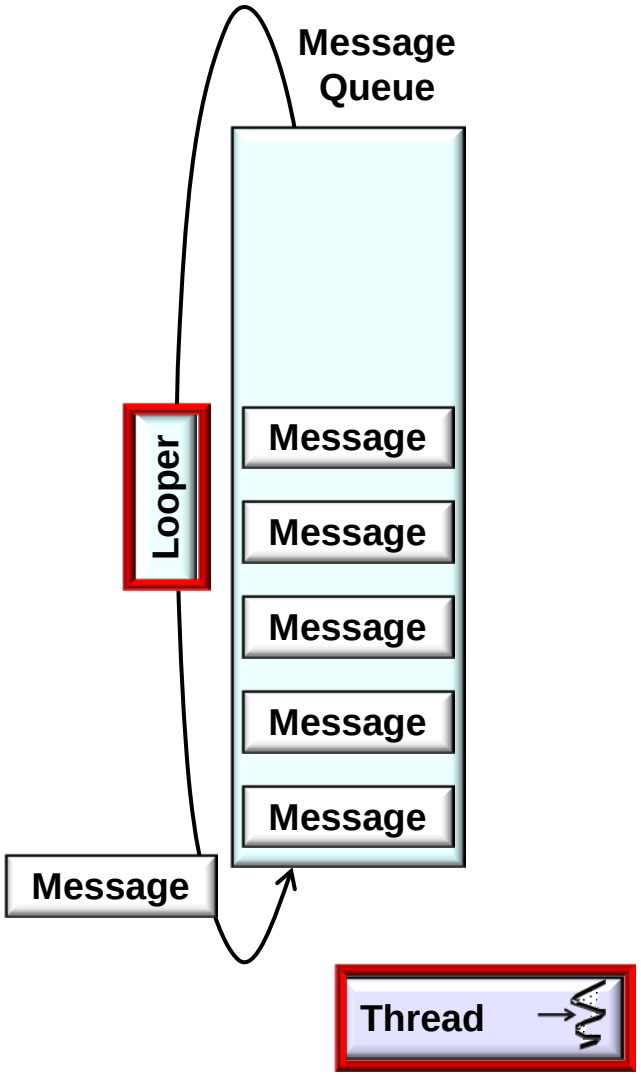
The Looper Class

- A Looper has a synchronized MessageQueue that's used to process Messages placed on the queue
  - It implements a Thread-specific event loop
  - Only one Looper per Thread



The Looper Class

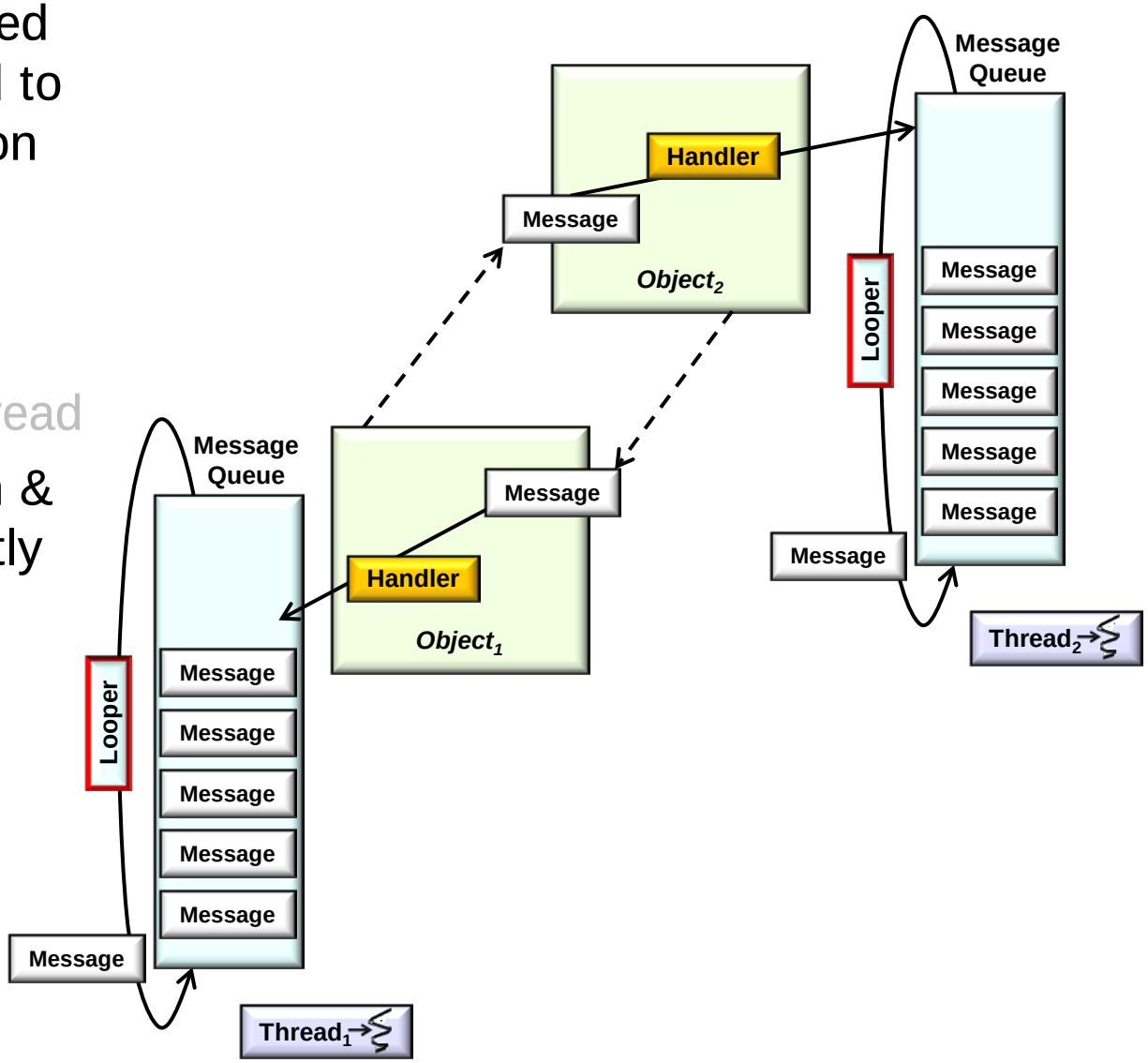
- A Looper has a synchronized MessageQueue that's used to process Messages placed on the queue
  - It implements a Thread-specific event loop
  - Only one Looper per Thread
    - Enforced by the *Thread-Specific Storage* pattern



See upcoming parts on "The *Thread-Specific Storage* Pattern"

The Looper Class

- A Looper has a synchronized MessageQueue that's used to process Messages placed on the queue
  - It implements a Thread-specific event loop
  - Only one Looper per Thread
  - Multiple Loopers can run & communicate concurrently in multiple Threads



The Looper Class

- A Looper has a synchronized MessageQueue that's used to process Messages placed on the queue
- Key methods include prepare(), loop(), & quit()

```
public class Looper {
  ...

  public static void prepare() {
    ...
  }

  public static loop() {
   ...
  }

  public void quit() {
    ...
  }

  ...
```

frameworks/base/core/java/android/os/Looper.java has the source code

The Looper Class

- A Looper has a synchronized MessageQueue that's used to process Messages placed on the queue
- Key methods include prepare(), loop(), & quit()
  - prepare() initializes the current thread as a Looper

```
public class Looper {
  ...

  public static void prepare() {
    ...
    sThreadLocal.set
      (new Looper());
  }

  ...
```

The Looper Class

- A Looper has a synchronized MessageQueue that's used to process Messages placed on the queue
- Key methods include prepare(), loop(), & quit()
  - prepare() initializes the current thread as a Looper

```
public class Looper {
  ...

  public static void prepare() {
    ...
    sThreadLocal.set
      (new Looper());
  }

  ...
```

developer.android.com/reference/java/lang/ThreadLocal.html has more info

The Looper Class

- A Looper has a synchronized MessageQueue that's used to process Messages placed on the queue
- Key methods include prepare(), loop(), & quit()
  - prepare() initializes the current thread as a Looper

```
public class Looper {
  ...

  public static void prepare() {
    ...
    sThreadLocal.set
      (new Looper());
  }

  ...
```

www.dre.vanderbilt.edu/~schmidt/PDF/TSS-pattern.pdf has more info

The Looper Class

- A Looper has a synchronized MessageQueue that's used to process Messages placed on the queue
- Key methods include prepare(), loop(), & quit()
  - prepare() initializes the current thread as a Looper

```
public class Looper {
  ...

  public static void prepare() {
    ...
    sThreadLocal.set
      (new Looper());
  }

  ...
```

prepare() must be called prior to running the event loop

The Looper Class

- A Looper has a synchronized MessageQueue that's used to process Messages placed on the queue
- Key methods include prepare(), loop(), & quit()
  - prepare() initializes the current thread as a Looper
  - loop() runs the event loop

```
public class Looper {
  ...
  final MessageQueue mQueue;

  public static void loop() {
    Looper me = myLooper();
    ...
    MessageQueue queue = me.mQueue;

    for (;;) {
      Message msg =
        queue.next();
      ...
      msg.target.
        dispatchMessage(msg);
      ...
  }
  ...
```

The Looper Class

- A Looper has a synchronized MessageQueue that's used to process Messages placed on the queue
- Key methods include prepare(), loop(), & quit()
  - prepare() initializes the current thread as a Looper
  - loop() runs the event loop
    - Waits for Messages

```
public class Looper {
  ...
  final MessageQueue mQueue;

  public static void loop() {
    Looper me = myLooper();
    ...
    MessageQueue queue = me.mQueue;

    for (;;) {
      Message msg =
        queue.next();
      ...
      msg.target.
        dispatchMessage(msg);
      ...
  }
  ...
```

The Looper Class

- A Looper has a synchronized MessageQueue that's used to process Messages placed on the queue
- Key methods include prepare(), loop(), & quit()
  - prepare() initializes the current thread as a Looper
  - loop() runs the event loop
    - Waits for Messages

```
public class Looper {
  ...
  final MessageQueue mQueue;

  public static void loop() {
    Looper me = myLooper();
    ...
    MessageQueue queue = me.mQueue;

    for (;;) {
      Message msg =
        queue.next();
      ...
      msg.target.
        dispatchMessage(msg);
      ...
  }
  ...
```

The Looper Class

- A Looper has a synchronized MessageQueue that's used to process Messages placed on the queue
- Key methods include prepare(), loop(), & quit()
  - prepare() initializes the current thread as a Looper
  - loop() runs the event loop
    - Waits for Messages

```
public class Looper {
  ...
  final MessageQueue mQueue;

  public static void loop() {
    Looper me = myLooper();
    ...
    MessageQueue queue = me.mQueue;

    for (;;) {
      Message msg =
        queue.next();
      ...
      msg.target.
        dispatchMessage(msg);
      ...
  }
  ...
```

This call can block

frameworks/base/libs/utils/Looper.cpp has the source code

The Looper Class

- A Looper has a synchronized MessageQueue that's used to process Messages placed on the queue
- Key methods include prepare(), loop(), & quit()
  - prepare() initializes the current thread as a Looper
  - loop() runs the event loop
    - Waits for Messages
    - Dispatches them to their Handlers

```
public class Looper {
  ...
  final MessageQueue mQueue;

  public static void loop() {
    Looper me = myLooper();
    ...
    MessageQueue queue = me.mQueue;

    for (;;) {
      Message msg =
        queue.next();
      ...
      msg.target.
        dispatchMessage(msg);
      ...
  }
  ...
```

The Looper Class

- A Looper has a synchronized MessageQueue that's used to process Messages placed on the queue
- Key methods include prepare(), loop(), & quit()
  - prepare() initializes the current thread as a Looper
  - loop() runs the event loop
    - Waits for Messages
    - Dispatches them to their Handlers

Note inversion of control

```
public class Looper {
  ...
  final MessageQueue mQueue;

  public static void loop() {
    Looper me = myLooper();
    ...
    MessageQueue queue = me.mQueue;

    for (;;) {
      Message msg =
        queue.next();
      ...
      msg.target.
        dispatchMessage(msg);
      ...
  }
  ...
```

The Looper Class

- A Looper has a synchronized MessageQueue that's used to process Messages placed on the queue
- Key methods include prepare(), loop(), & quit()
  - prepare() initializes the current thread as a Looper
  - loop() runs the event loop
  - quit() shuts down the event loop

```
public class Looper {
  ...

  public void quit() {
    Message msg =
      Message.obtain();
    mQueue.enqueueMessage(msg,
                          0);
  }

  ...
```

The Looper Class

- A Looper has a synchronized MessageQueue that's used to process Messages placed on the queue
- Key methods include prepare(), loop(), & quit()
  - prepare() initializes the current thread as a Looper
  - loop() runs the event loop
  - quit() shuts down the event loop
    - It enqueues a special Message

```
public class Looper {
  ...

  public void quit() {
    Message msg =
      Message.obtain();
    mQueue.enqueueMessage(msg,
                          0);
  }

  ...
```

The Looper Class

- A Looper has a synchronized MessageQueue that's used to process Messages placed on the queue
- Key methods include prepare(), loop(), & quit()
- A Handler actually manages the MessageQueue

Handler

Methods | [Expand All]
Added in API level 1

extends Object

java.lang.Object
↳ android.os.Handler

▸ Known Direct Subclasses
AsyncQueryHandler, AsyncQueryHandler.WorkerHandler, HttpAuthHandler, SslErrorHandler

Class Overview

A Handler 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. When you create a new Handler, it is bound to the thread / message queue of the thread that is creating it -- from that point on, it will deliver messages and runnables to that message queue and execute them as they come out of the message queue.

There are two main uses for a Handler: (1) to schedule messages and runnables to be executed as some point in the future; and (2) to enqueue an action to be performed on a different thread than your own.

The Looper Class

- A Looper has a synchronized MessageQueue that's used to process Messages placed on the queue
- Key methods include prepare(), loop(), & quit()
- A Handler actually manages the MessageQueue
  - e.g., it adds, removes, & dispatches Messages to their intended targets

Handler

Methods | [Expand All]
Added in API level 1

extends Object

java.lang.Object
↳ android.os.Handler

▸ Known Direct Subclasses
AsyncQueryHandler, AsyncQueryHandler.WorkerHandler, HttpAuthHandler, SslErrorHandler

Class Overview

A Handler 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. When you create a new Handler, it is bound to the thread / message queue of the thread that is creating it -- from that point on, it will deliver messages and runnables to that message queue and execute them as they come out of the message queue.

There are two main uses for a Handler: (1) to schedule messages and runnables to be executed as some point in the future; and (2) to enqueue an action to be performed on a different thread than your own.

developer.android.com/reference/android/os/Handler.html has more info

The Looper Class

- A Looper has a synchronized MessageQueue that's used to process Messages placed on the queue
- Key methods include prepare(), loop(), & quit()
- A Handler actually manages the MessageQueue
  - e.g., it adds, removes, & dispatches Messages to their intended targets

Handler

Methods | [Expand All]
Added in API level 1

extends Object

java.lang.Object
↳ android.os.Handler

▸ Known Direct Subclasses
AsyncQueryHandler, AsyncQueryHandler.WorkerHandler, HttpAuthHandler, SslErrorHandler

Class Overview

A Handler 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. When you create a new Handler, it is bound to the thread / message queue of the thread that is creating it -- from that point on, it will deliver messages and runnables to that message queue and execute them as they come out of the message queue.

There are two main uses for a Handler: (1) to schedule messages and runnables to be executed as some point in the future; and (2) to enqueue an action to be performed on a different thread than your own.

See upcoming parts on the "Android Handler"

Using Looper in Android

Using Looper in Android

- A Thread doesn't have a Looper associated with it

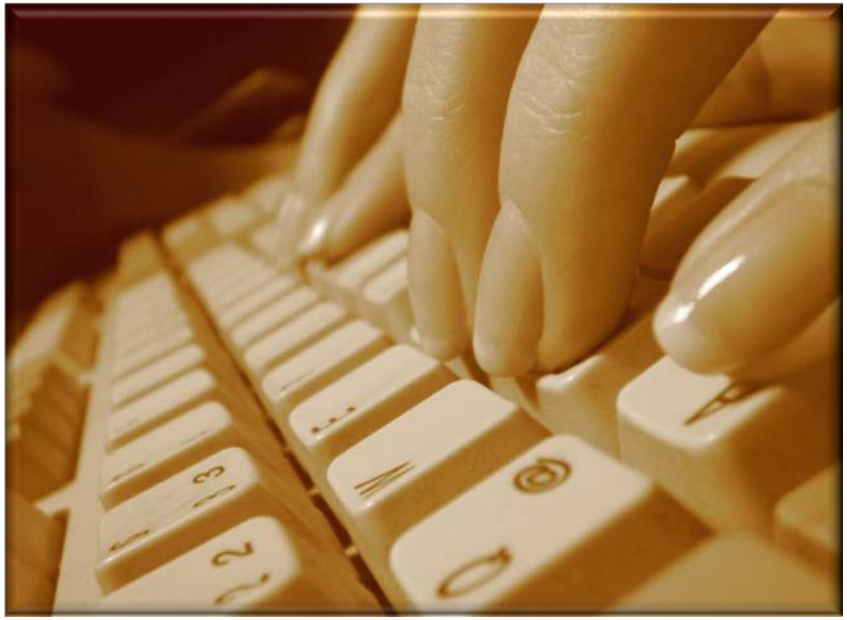
```
public class Thread
        implements Runnable {
  public st
                    curr    Thread() {

  }

  ublic f    l void join() {
    ...
  }

  public void inte    pt() {
    ...


  pu    c synchronized void    art() {
    ...
  }
```

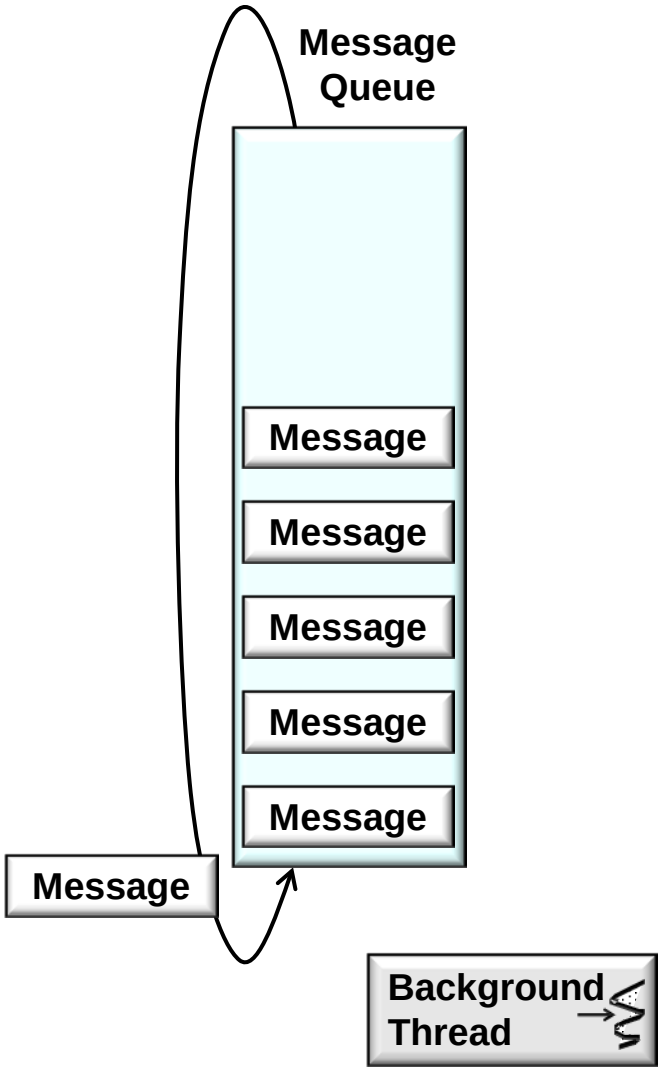
Looper

developer.android.com/reference/java/lang/Thread.html has more info

Using Looper in Android

- A Thread doesn't have a Looper associated with it
- A Looper can be programmed in a Thread via several steps





developer.android.com/reference/android/os/Looper.html shows the steps

Using Looper in Android

- A Thread doesn't have a Looper associated with it
- A Looper can be programmed in a Thread via several steps

com.android.videoeditor

Class VideoEditorActivity

```
java.lang.Object
  └ VideoEditorBaseActivity
      └ com.android.videoeditor.VideoEditorActivity
```

```
public class VideoEditorActivity
extends VideoEditorBaseActivity
```

Main activity of the video editor. It handles video editing of a project.

e.g., a Looper is used in the VideoEditorActivity

See earlier part on "Java Semaphore"

Using Looper in Android

- A Thread doesn't have a Looper associated with it
- A Looper can be programmed in a Thread via several steps

```
public class VideoEditorActivity ... {
  ...
  private class PreviewThread
  ...
```

packages/apps/VideoEditor/src/com/android/videoeditor/VideoEditorActivity.java

Using Looper in Android

- A Thread doesn't have a Looper associated with it
- A Looper can be programmed in a Thread via several steps
  1. Extend Thread

```
public class VideoEditorActivity ... {
  ...
  private class PreviewThread
                extends Thread {
    ...
```

Using Looper in Android

- A Thread doesn't have a Looper associated with it
- A Looper can be programmed in a Thread via several steps
  1. Extend Thread

```
public class VideoEditorActivity ... {
  ...
  private class PreviewThread
                extends Thread {
    ...
    PreviewThread() {
      ...
      start();
      ...
    }
```

Using Looper in Android

- A Thread doesn't have a Looper associated with it
- A Looper can be programmed in a Thread via several steps
  1. Extend Thread

```
public class VideoEditorActivity ... {
  ...
  private class PreviewThread
                extends Thread {
    ...
    PreviewThread() {
      ...
      start();
      ...
    }
```

Using Looper in Android

- A Thread doesn't have a Looper associated with it
- A Looper can be programmed in a Thread via several steps
  1. Extend Thread & implement its run() hook method

```
public class VideoEditorActivity ... {
  ...
  private class PreviewThread
                extends Thread {
    ...
    private final Handler
      mThreadHandler;
    ...
    public void run() {
      ...
      Looper.prepare();
      ...
      mThreadHandler = new Handler();
      ...
      Looper.loop();
      ...
```

Using Looper in Android

- A Thread doesn't have a Looper associated with it
- A Looper can be programmed in a Thread via several steps
  1. Extend Thread & implement its run() hook method
  2. Call Looper.prepare() to initialize current Thread as a Looper

```
public class VideoEditorActivity ... {
  ...
  private class PreviewThread
                extends Thread {
    ...
    private final Handler
      mThreadHandler;
    ...
    public void run() {
      ...
      Looper.prepare();
      ...
      mThreadHandler = new Handler();
      ...
      Looper.loop();
      ...
```

Using Looper in Android

- A Thread doesn't have a Looper associated with it
- A Looper can be programmed in a Thread via several steps
  1. Extend Thread & implement its run() hook method
  2. Call Looper.prepare() to initialize current Thread as a Looper
  3. Create one or more Handlers to process incoming Messages

```
public class VideoEditorActivity ... {
  ...
  private class PreviewThread
                extends Thread {
    ...
    private final Handler
      mThreadHandler;
    ...
    public void run() {
      ...
      Looper.prepare();
      ...
      mThreadHandler = new Handler();
      ...
      Looper.loop();
      ...
```

Using Looper in Android

- A Thread doesn't have a Looper associated with it
- A Looper can be programmed in a Thread via several steps
  1. Extend Thread & implement its run() hook method
  2. Call Looper.prepare() to initialize current Thread as a Looper
  3. Create one or more Handlers to process incoming Messages
  4. Call Looper.loop() to process Messages until the loop is told to quit

```
public class VideoEditorActivity ... {
  ...
  private class PreviewThread
                 extends Thread {
    ...
    private final Handler
      mThreadHandler;
    ...
    public void run() {
      ...
      Looper.prepare();
      ...
      mThreadHandler = new Handler();
      ...
      Looper.loop();
      ...
```

Using Looper in Android

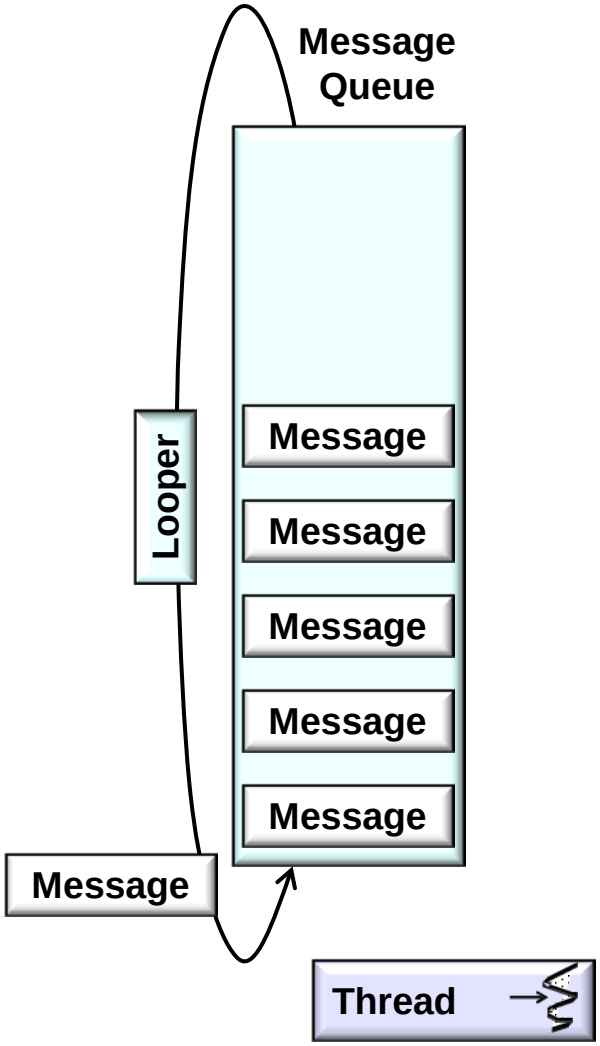
- A Thread doesn't have a Looper associated with it
- A Looper can be programmed in a Thread via several steps
  1. Extend Thread & implement its run() hook method
  2. Call Looper.prepare() to initialize current Thread as a Looper
  3. Create one or more Handlers to process incoming Messages
  4. Call Looper.loop() to process Messages until the loop is told to quit

```
public class VideoEditorActivity ... {
  ...
  private class PreviewThread
                 extends Thread {
    ...
    private final Handler
      mThreadHandler;
    ...
    public void run() {
      ...
      Looper.prepare();
      ...
      mThreadHandler = new Handler();
      ...
      Looper.loop();
      ...
```

These steps are applied to add Loopers to Threads throughout Android

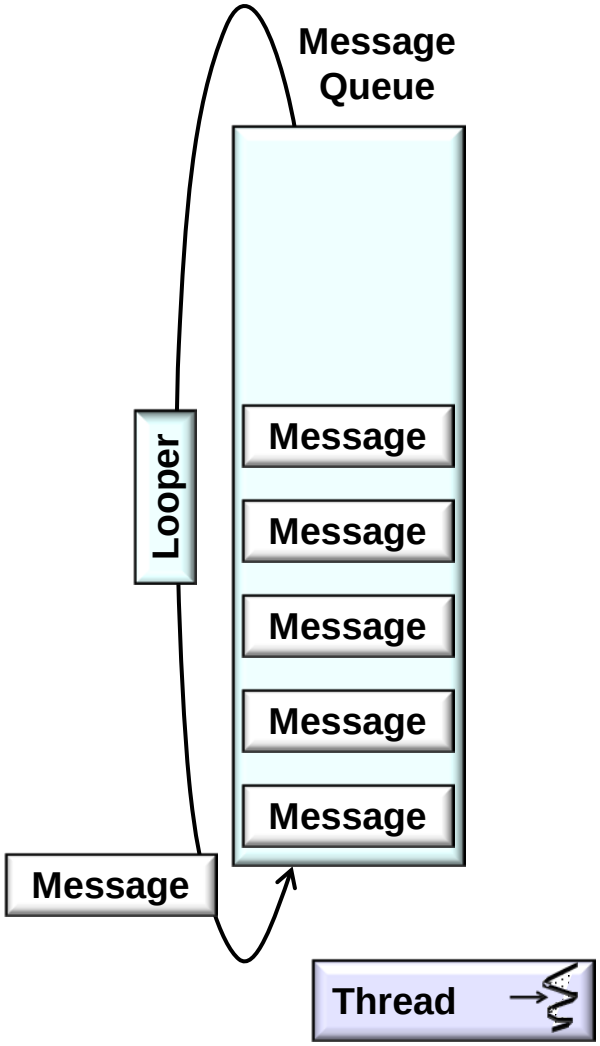
Using the Looper in Android's Concurrency Frameworks

Using Looper in Android Concurrency Frameworks




```
public class Looper {
  ...

  public static void prepare() {
    ...
  }

  public static loop() {
   ...
  }

  public void quit() {
    ...
  }

  ...
```

Using Looper in Android Concurrency Frameworks



com.android.videoeditor

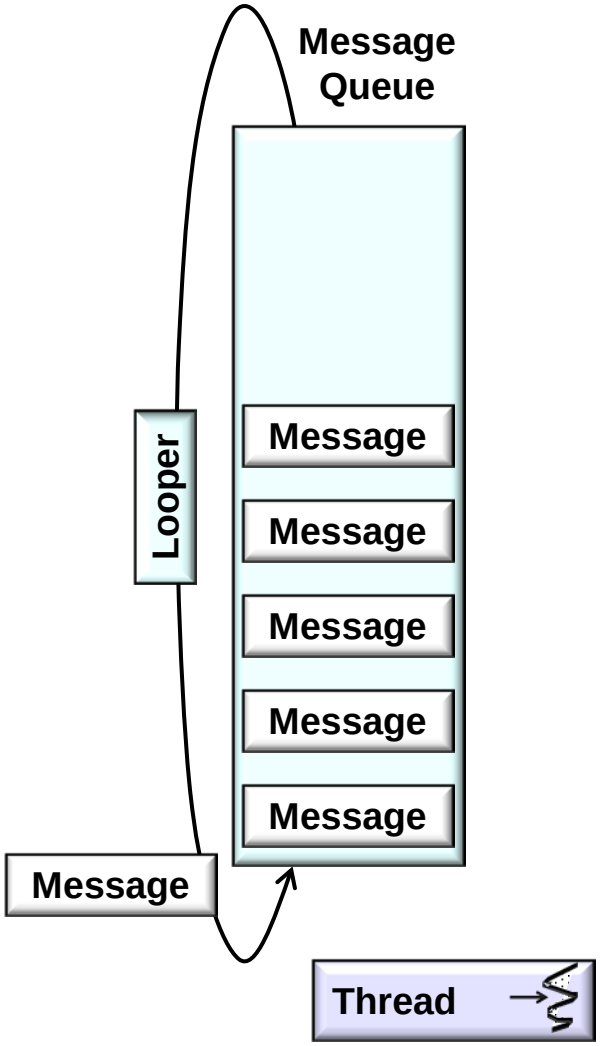
Class VideoEditorActivity

```
java.lang.Object
  └ VideoEditorBaseActivity
      └ com.android.videoeditor.VideoEditorActivity
```

```
public class VideoEditorActivity
extends VideoEditorBaseActivity
```

Main activity of the video editor. It handles video editing of a project.

Using Looper in Android Concurrency Frameworks



android.app

Class ActivityThread

java.lang.Object
↳ android.app.ActivityThread

```
public final class ActivityThread
extends Object
```

This manages the execution of the main thread in an application process, scheduling and executing activities, broadcasts, and other operations on it as the activity manager requests.

Nested Class Summary	
(package private) class	ActivityThread.GcIdler
static class	ActivityThread.PackageInfo

HandlerThread

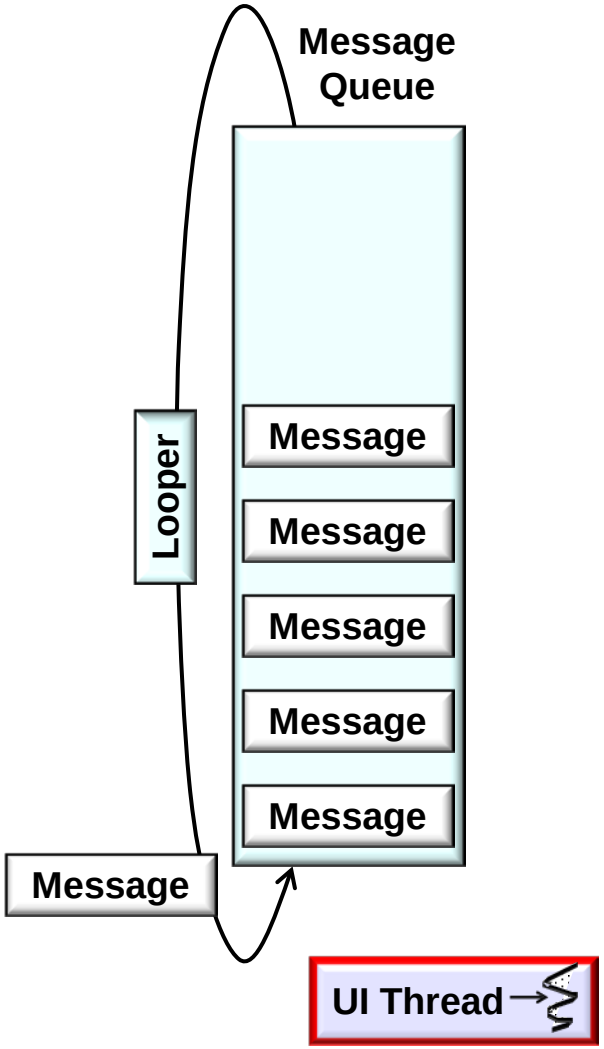
extends Thread

java.lang.Object
↳ java.lang.Thread
↳ android.os.HandlerThread

Class Overview

Handy class for starting a new thread that has a looper. The looper can then be used to create handler classes. Note that start() must still be called.

Using Looper in Android Concurrency Frameworks



android.app

Class ActivityThread

java.lang.Object
↳ android.app.ActivityThread

```
public final class ActivityThread
extends Object
```

This manages the execution of the main thread in an application process, scheduling and executing activities, broadcasts, and other operations on it as the activity manager requests.

Nested Class Summary	
(package private) class	ActivityThread.GcIdler
static class	ActivityThread.PackageInfo

HandlerThread

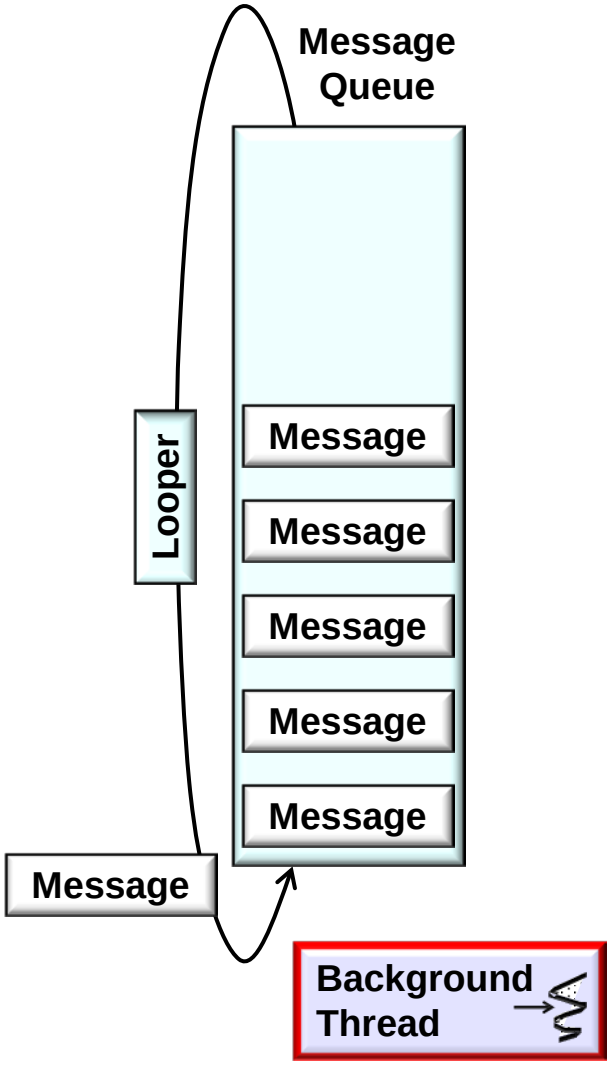
extends Thread

java.lang.Object
↳ java.lang.Thread
↳ android.os.HandlerThread

Class Overview

Handy class for starting a new thread that has a looper. The looper can then be used to create handler classes. Note that start() must still be called.

Using Looper in Android Concurrency Frameworks



android.app

Class ActivityThread

java.lang.Object
↳ android.app.ActivityThread

```
public final class ActivityThread
extends Object
```

This manages the execution of the main thread in an application process, scheduling and executing activities, broadcasts, and other operations on it as the activity manager requests.

Nested Class Summary	
(package private) class	ActivityThread.GcIdler
static class	ActivityThread.PackageInfo

HandlerThread

extends Thread

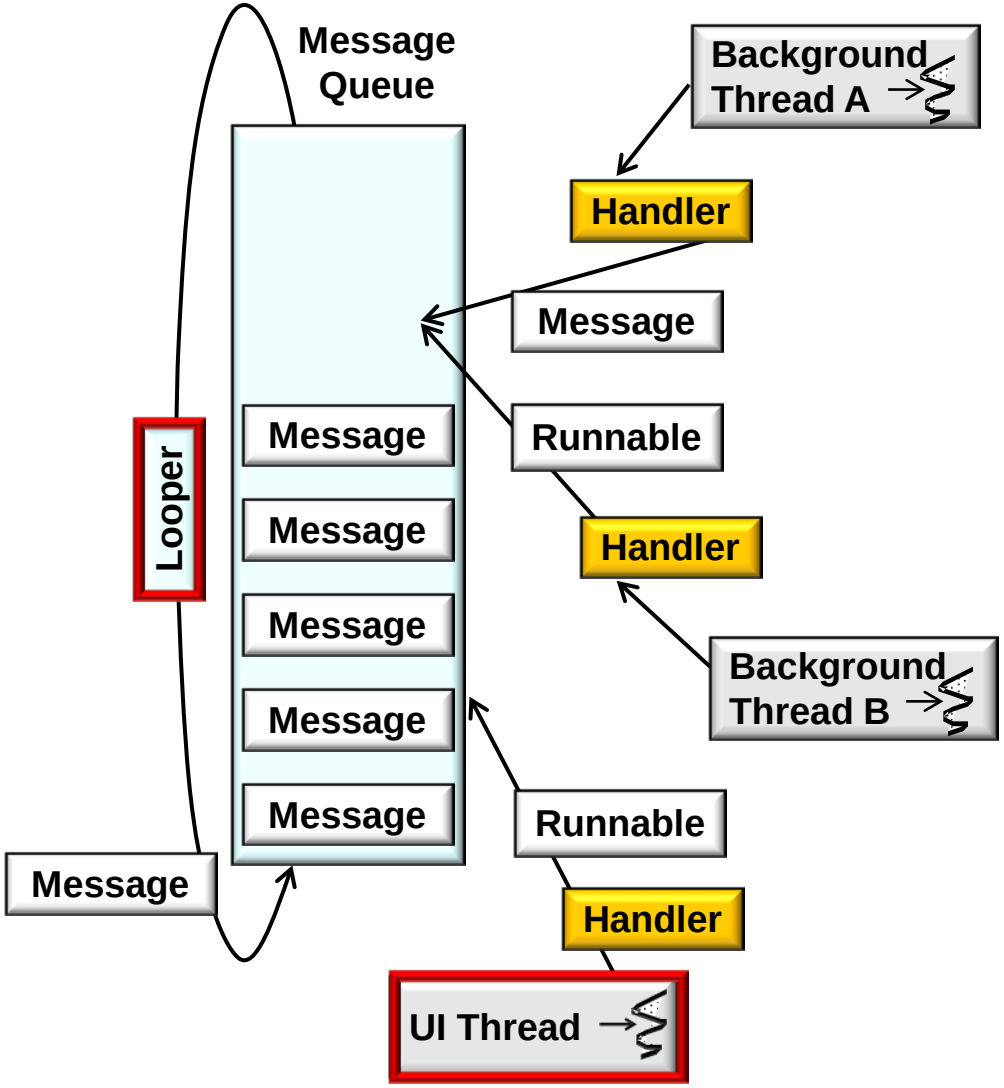
java.lang.Object
↳ java.lang.Thread
↳ android.os.HandlerThread

Class Overview

Handy class for starting a new thread that has a looper. The looper can then be used to create handler classes. Note that start() must still be called.

Using Looper in Android Concurrency Frameworks

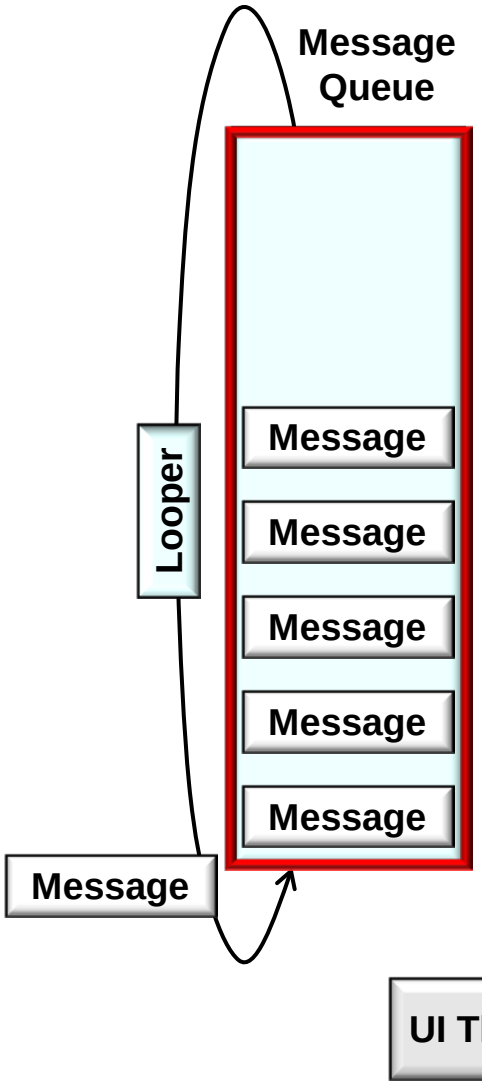
- The UI Thread has a Looper used to receive messages from itself & other background threads



developer.android.com/training/multiple-threads/communicate-ui.html#Handler

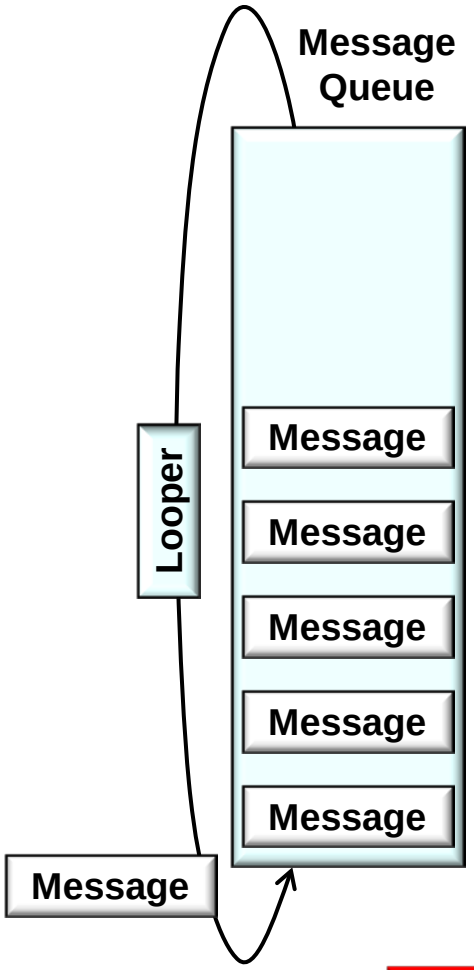
Using Looper in Android Concurrency Frameworks

- The UI Thread has a Looper used to receive messages from itself & other background threads
  - This Looper's MessageQueue helps serialize access to UI components



Using Looper in Android Concurrency Frameworks

- The UI Thread has a Looper used to receive messages from itself & other background threads
  - This Looper's MessageQueue helps serialize access to UI components



Using Looper in Android Concurrency Frameworks

- The UI Thread has a Looper used to receive messages from itself & other background threads
  - This Looper's MessageQueue helps serialize access to UI components
  - ActivityThread contains the UI Thread's Looper

```
class ActivityThread {
  ...
  final Looper mLooper =
    Looper.myLooper();
  ...
  public static void
            main(String[] args)
  {
    ...
    Looper.prepareMainLooper();
    ...
    Looper.loop();
    ...
```

Using Looper in Android Concurrency Frameworks

- The UI Thread has a Looper used to receive messages from itself & other background threads
  - This Looper's MessageQueue helps serialize access to UI components
  - ActivityThread contains the UI Thread's Looper

```
class ActivityThread {
  ...
  final Looper mLooper =
    Looper.myLooper();
  ...
  public static void
            main(String[] args)
  {
    ...
    Looper.prepareMainLooper();
    ...
    Looper.loop();
    ...
```

frameworks/base/core/java/android/app/ActivityThread.java

Using Looper in Android Concurrency Frameworks

- The UI Thread has a Looper used to receive messages from itself & other background threads
  - This Looper's MessageQueue helps serialize access to UI components
  - ActivityThread contains the UI Thread's Looper

```
class ActivityThread {
  ...
  final Looper mLooper =
    Looper.myLooper();
  ...
  public static void
            main(String[] args)
  {
    ...
    Looper.prepareMainLooper();
    ...
    Looper.loop();
    ...
```

Using Looper in Android Concurrency Frameworks

- The UI Thread has a Looper used to receive messages from itself & other background threads
  - This Looper's MessageQueue helps serialize access to UI components
  - ActivityThread contains the UI Thread's Looper

```
class ActivityThread {
  ...
  final Looper mLooper =
    Looper.myLooper();
  ...
  public static void
            main(String[] args)
  {
    ...
    Looper.prepareMainLooper();
    ...
    Looper.loop();
    ...
```

Initialize the current Thread as the main Looper in the application

Using Looper in Android Concurrency Frameworks

- The UI Thread has a Looper used to receive messages from itself & other background threads
  - This Looper's MessageQueue helps serialize access to UI components
  - ActivityThread contains the UI Thread's Looper

```
class ActivityThread {
  ...
  final Looper mLooper =
    Looper.myLooper();
  ...
  public static void
            main(String[] args)
  {
    ...
    Looper.prepareMainLooper();
    ...
    Looper.loop();
    ...
```

Android's runtime calls prepareMainLooper(), so it needn't be called directly

Using Looper in Android Concurrency Frameworks

- The UI Thread has a Looper used to receive messages from itself & other background threads
- Loopers can also be used in non-UI Threads

```
class ServerThread
      extends Thread {
  ...
  public void run() {
    Looper.prepare();
    ...
    Looper.run();
  }
  ...
```

Many examples throughout Android's frameworks, applications, & providers

Using Looper in Android Concurrency Frameworks

- The UI Thread has a Looper used to receive messages from itself & other background threads
- Loopers can also be used in non-UI Threads

```
class ServerThread
      extends Thread {
  ...
  public void run() {
    Looper.prepare();
    ...
    Looper.loop();
  }
  ...
```

Using Looper in Android Concurrency Frameworks

- The UI Thread has a Looper used to receive messages from itself & other background threads
- Loopers can also be used in non-UI Threads

```
class ServerThread
      extends Thread {
  ...
  public void run() {
    Looper.prepare();
    ...
    Looper.loop();
  }
  ...
```

Using Looper in Android Concurrency Frameworks

- The UI Thread has a Looper used to receive messages from itself & other background threads
- Loopers can also be used in non-UI Threads

```
class ServerThread
      extends Thread {
  ...
  public void run() {
    Looper.prepare();
    ...
    Looper.loop();
  }
  ...
```

Using Looper in Android Concurrency Frameworks

- The UI Thread has a Looper used to receive messages from itself & other background threads
- Loopers can also be used in non-UI Threads
  - e.g., HandlerThread is a helper class for starting a new Thread that contains a Looper

Added in API level 1

HandlerThread

extends Thread

java.lang.Object
↳ java.lang.Thread
↳ android.os.HandlerThread

Class Overview

Handy class for starting a new thread that has a looper. The looper can then be used to create handler classes. Note that start() must still be called.

developer.android.com/reference/android/os/HandlerThread.html has more info

Using Looper in Android Concurrency Frameworks

- The UI Thread has a Looper used to receive messages from itself & other background threads
- Loopers can also be used in non-UI Threads
  - e.g., HandlerThread is a helper class for starting a new Thread that contains a Looper

```
class HandlerThread extends Thread {
  Looper mLooper;
  ...

  public void run() {
    Looper.prepare();
    synchronized (this) {
      mLooper = Looper.myLooper();
      ...
    }
    ...
    onLooperPrepared();
    Looper.loop();
    ...

  protected void onLooperPrepared() {
  }
}
```

en.wikipedia.org/wiki/Template_method_pattern has more info

Using Looper in Android Concurrency Frameworks

- The UI Thread has a Looper used to receive messages from itself & other background threads
- Loopers can also be used in non-UI Threads
  - e.g., HandlerThread is a helper class for starting a new Thread that contains a Looper

```
class HandlerThread extends Thread {
  Looper mLooper;
  ...

  public void run() {
    Looper.prepare();
    synchronized (this) {
      mLooper = Looper.myLooper();
      ...
    }
    ...
    onLooperPrepared();
    Looper.loop();
    ...

  protected void onLooperPrepared() {
  }
}
```

The Template Method pattern is used to perform fixed steps in the algorithm

en.wikipedia.org/wiki/Template_method_pattern has more info

Using Looper in Android Concurrency Frameworks

- The UI Thread has a Looper used to receive messages from itself & other background threads
- Loopers can also be used in non-UI Threads
  - e.g., HandlerThread is a helper class for starting a new Thread that contains a Looper

```
class HandlerThread extends Thread {
  Looper mLooper;
  ...

  public void run() {
    Looper.prepare();
    synchronized (this) {
      mLooper = Looper.myLooper();
      ...
    }
    ...
    onLooperPrepared();
    Looper.loop();
    ...

  protected void onLooperPrepared() {
  }
}
```

run() is a template method

en.wikipedia.org/wiki/Template_method_pattern has more info

Using Looper in Android Concurrency Frameworks

- The UI Thread has a Looper used to receive messages from itself & other background threads
- Loopers can also be used in non-UI Threads
  - e.g., HandlerThread is a helper class for starting a new Thread that contains a Looper

```
class HandlerThread extends Thread {
  Looper mLooper;
  ...

  public void run() {
    Looper.prepare();
    synchronized (this) {
      mLooper = Looper.myLooper();
      ...
    }
    ...
    onLooperPrepared();
    Looper.loop();
    ...

  protected void onLooperPrepared() {
  }
}
```

This hook method enables subclasses to create Handlers

Using Looper in Android Concurrency Frameworks

- The UI Thread has a Looper used to receive messages from itself & other background threads
- Loopers can also be used in non-UI Threads
  - e.g., HandlerThread is a helper class for starting a new Thread that contains a Looper
    - The start() method must still be called by client code to launch the thread

```
public abstract class IntentService
                 extends Service {

  public void onCreate() {
    ...
    HandlerThread thread = new
      HandlerThread("IntentService["
                    + mName + "]");
      thread.start();
      ...
```

Using Looper in Android Concurrency Frameworks

- The UI Thread has a Looper used to receive messages from itself & other background threads
- Loopers can also be used in non-UI Threads
  - e.g., HandlerThread is a helper class for starting a new Thread that contains a Looper
    - The start() method must still be called by client code to launch the thread

```
public abstract class IntentService
                      extends Service {

  public void onCreate() {
    ...
    HandlerThread thread = new
      HandlerThread("IntentService["
                      + mName + "]");
      thread.start();
      ...
```

developer.android.com/reference/android/app/IntentService.html has more info

Using Looper in Android Concurrency Frameworks

- The UI Thread has a Looper used to receive messages from itself & other background threads
- Loopers can also be used in non-UI Threads

```
frameworks/base/core/java/android/app/IntentService.java
frameworks/base/core/java/android/content/AsyncQueryHandler.java
frameworks/base/core/java/android/content/SyncManager.java
frameworks/base/core/java/android/speech/tts/TextToSpeechService.java
frameworks/base/core/java/android/webkit/WebViewWorker.java
frameworks/base/core/java/android/widget/Filter.java
frameworks/base/drm/java/android/drm/DrmManagerClient.java
frameworks/base/services/java/com/android/server/usb/UsbDeviceManager.java
frameworks/base/services/java/com/android/server/BackupManagerService.java
frameworks/base/services/java/com/android/server/ConnectivityService.java
frameworks/base/services/java/com/android/server/MountService.java
frameworks/base/services/java/com/android/server/NativeDaemonConnector.java
frameworks/base/services/java/com/android/server/PowerManagerService.java
frameworks/base/services/java/com/android/server/WifiService.java
frameworks/base/telephony/java/com/android/internal/telephony/RIL.java
```

Many examples throughout Android's frameworks, applications, & providers

Using Looper in Android Concurrency Frameworks

- The UI Thread has a Looper used to receive messages from itself & other background threads
- Loopers can also be used in non-UI Threads

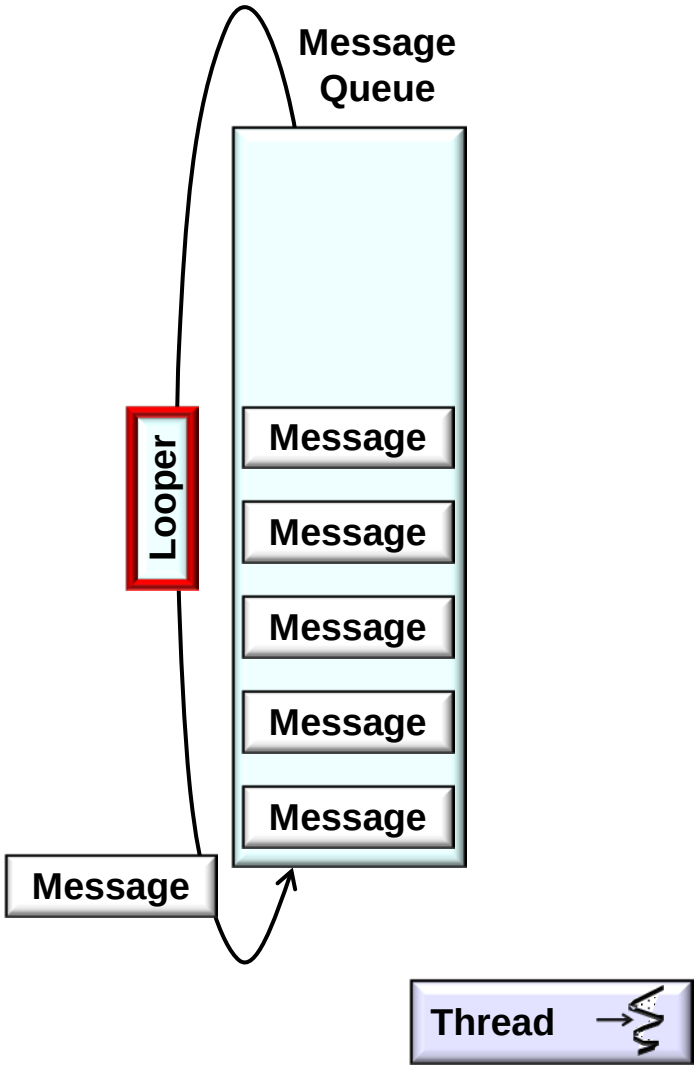
```
packages/apps/Browser/src/com/android/browser/BackgroundHandler.java
packages/apps/Calendar/src/com/android/calendar/alerts/AlertService.java
packages/apps/Camera/src/com/android/camera/CameraHolder.java
packages/apps/Contacts/src/com/android/contacts/model/AccountTypeManager.java
packages/apps/Contacts/src/com/android/contacts/ContactPhotoManager.java
packages/apps/DeskClock/src/com/android/deskclock/AsyncHandler.java
packages/apps/Launcher2/src/com/android/launcher2/LauncherModel.java
packages/apps/Mms/src/com/android/mms/transaction/SmsReceiverService.java
packages/apps/Mms/src/com/android/mms/transaction/TransactionService.java
packages/providers/ApplicationsProvider/src/com/android/providers/
        applications/ApplicationsProvider.java
packages/providers/ContactsProvider/src/com/android/providers/contacts/
        ContactsProvider2.java
packages/providers/MediaProvider/src/com/android/providers/media/
        MediaProvider.java
```

Many examples throughout Android's frameworks, applications, & providers

Summary

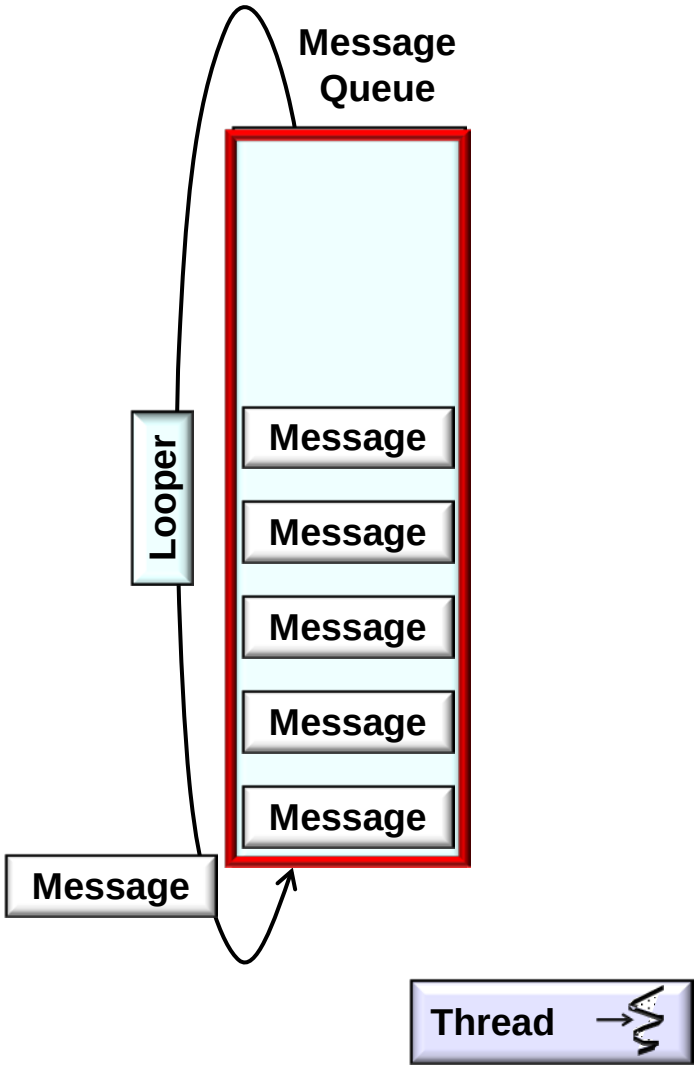
Summary

- A Thread can contain a Looper



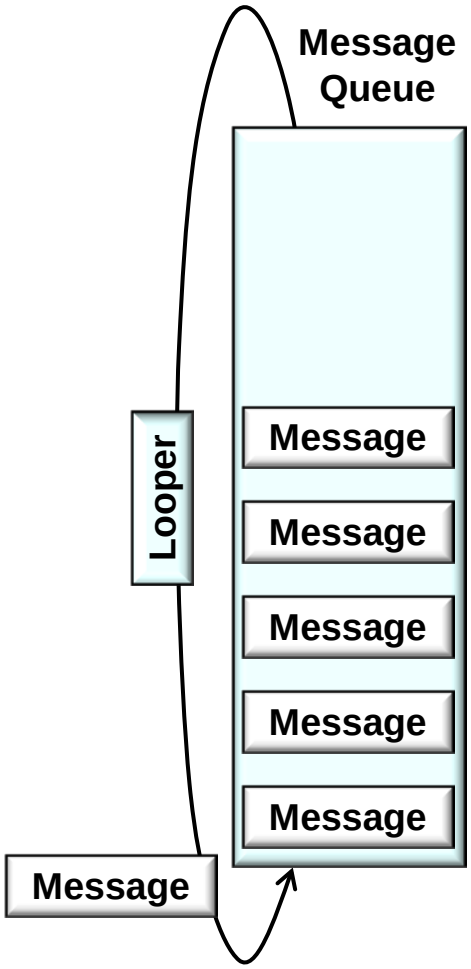
Summary

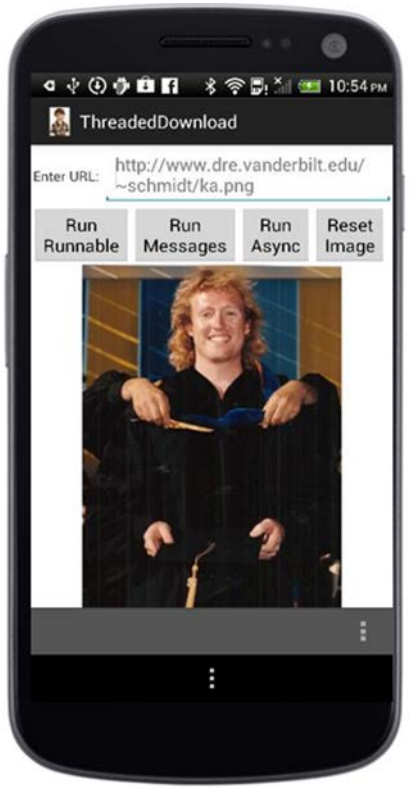
- A Thread can contain a Looper
  - The Looper, in turn, contains a MessageQueue



Summary

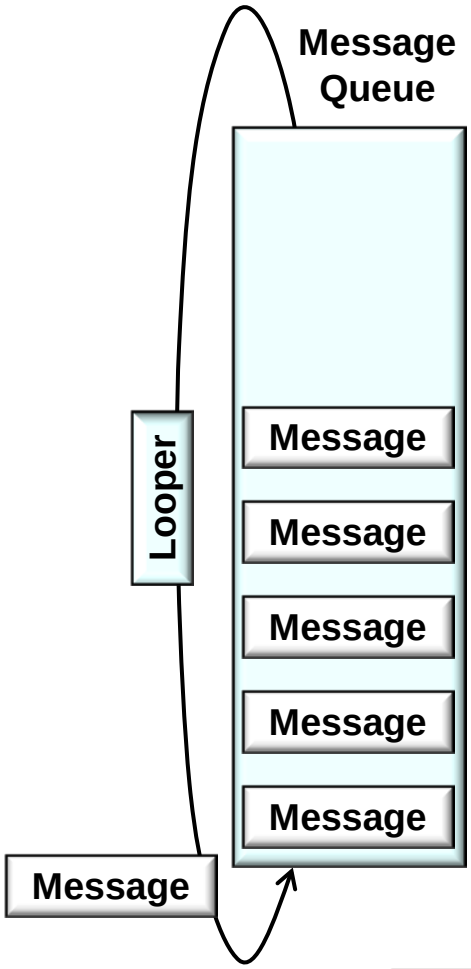
- A Thread can contain a Looper
  - The Looper, in turn, contains a MessageQueue
  - Applications use these elements to implement a Thread-specific event loop

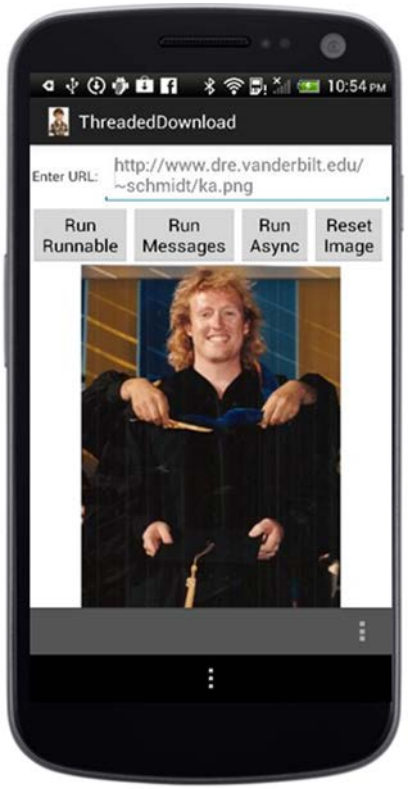




Summary

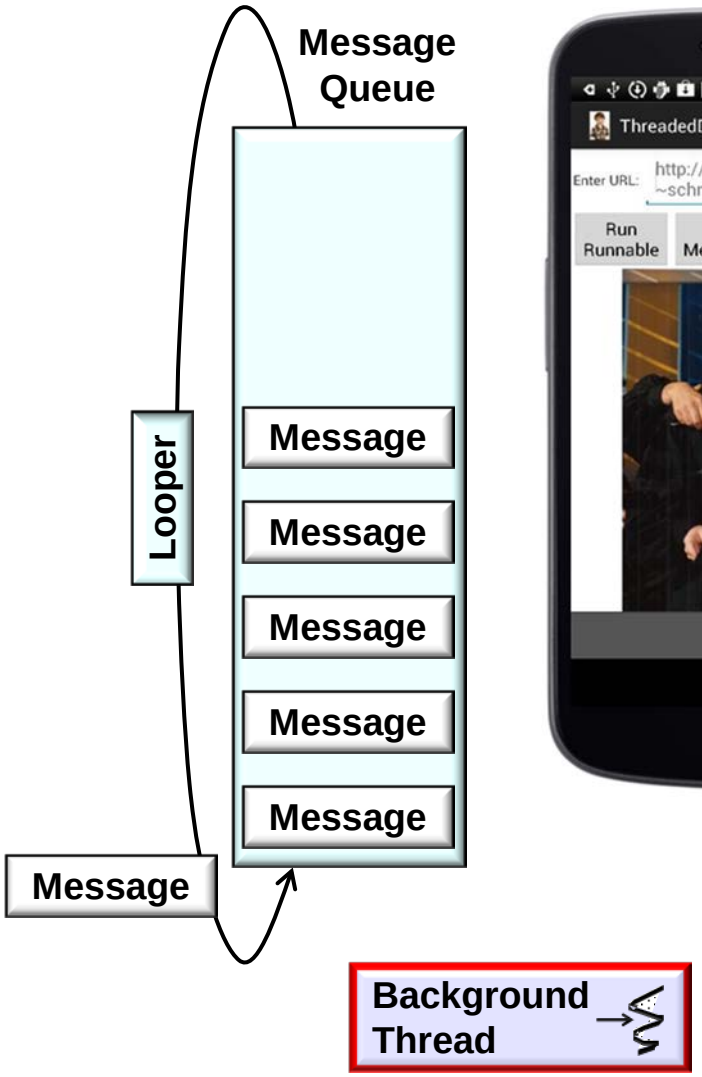
- A Thread can contain a Looper
  - The Looper, in turn, contains a MessageQueue
  - Applications use these elements to implement a Thread-specific event loop

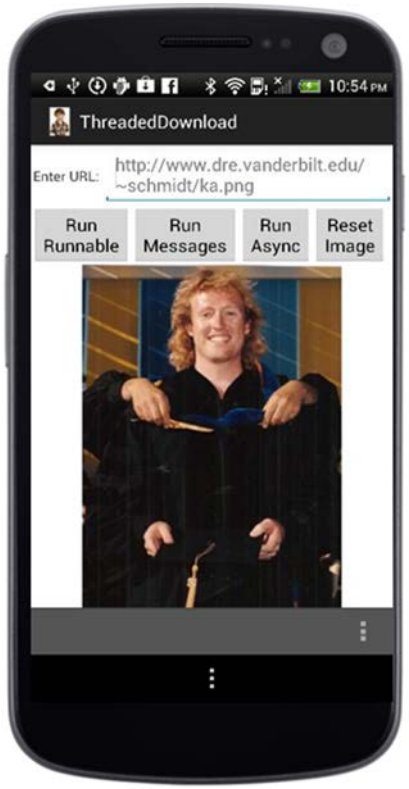




Summary

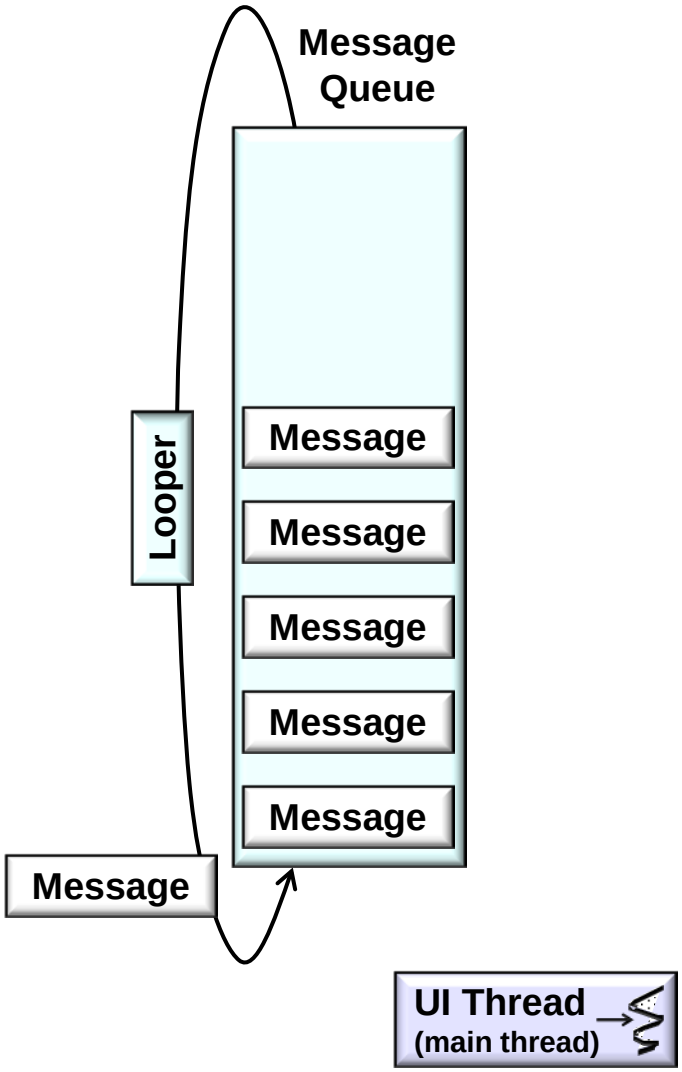
- A Thread can contain a Looper
  - The Looper, in turn, contains a MessageQueue
  - Applications use these elements to implement a Thread-specific event loop





Summary

- A Thread can contain a Looper
- A Looper is often invisible to application software components



Summary

- A Thread can contain a Looper
- A Looper is often invisible to application software components

Handler

Methods | [Expand All]
Added in API level 1

extends Object

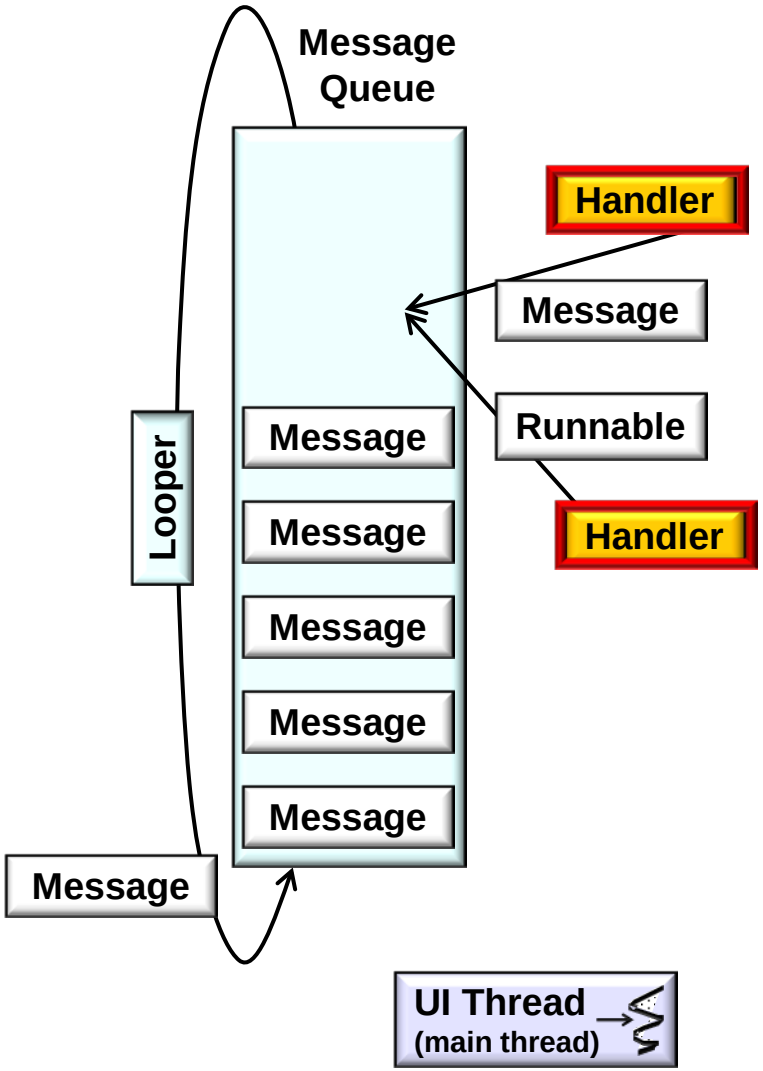
java.lang.Object
↳ android.os.Handler

▸ Known Direct Subclasses
AsyncQueryHandler, AsyncQueryHandler.WorkerHandler, HttpAuthHandler, SslErrorHandler

Class Overview

A Handler 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. When you create a new Handler, it is bound to the thread / message queue of the thread that is creating it -- from that point on, it will deliver messages and runnables to that message queue and execute them as they come out of the message queue.

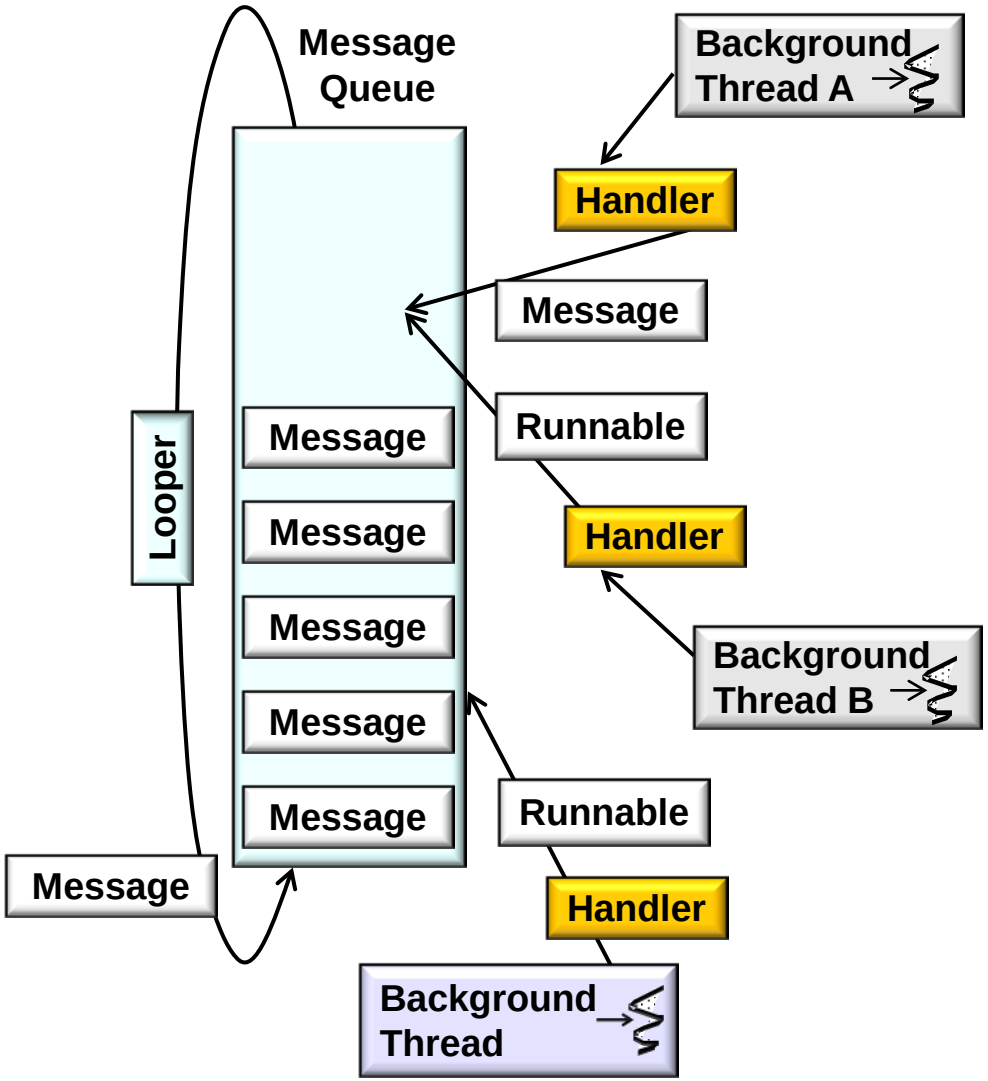
There are two main uses for a Handler: (1) to schedule messages and runnables to be executed as some point in the future; and (2) to enqueue an action to be performed on a different thread than your own.



See next part on “Android Handler”

Summary

- A Thread can contain a Looper
- A Looper is often invisible to application software components
  - Needed for more sophisticated concurrency models than the default single threaded model



Summary

- A Thread can contain a Looper
- A Looper is often invisible to application software components
- Loopers are used throughout Android

packages/apps/Email/src/com/android/email/NotificationController.java
packages/apps/Music/src/com/android/music/MediaPlaybackActivity.java
packages/apps/Phone/src/com/android/phone/EmergencyCallbackModeExitDialog.java
packages/apps/Phone/src/com/android/phone/Ringer.java
packages/apps/Phone/src/com/android/phone/PhoneInterfaceManager.java
packages/apps/Stk/src/com/android/stk/StkAppService.java
packages/apps/VideoEditor/src/com/android/videoeditor/VideoEditorActivity.java

Summary

- A Thread can contain a Looper
- A Looper is often invisible to application software components
- Loopers are used throughout Android

frameworks/base/core/java/android/provider/Browser.java
frameworks/base/core/java/android/webkit/WebSyncManager.java
frameworks/base/core/java/android/webkit/WebViewCore.java
frameworks/base/media/java/android/media/AudioService.java
frameworks/base/services/java/com/android/server/am/ActivityManagerService.java
frameworks/base/services/java/com/android/server/location/GpsLocationProvider.java
frameworks/base/services/java/com/android/server/wm/WindowManagerService.java
frameworks/base/services/java/com/android/server/CountryDetectorService.java
frameworks/base/services/java/com/android/server/LocationManagerService.java
frameworks/base/services/java/com/android/server/NotificationPlayer.java
frameworks/base/services/java/com/android/server/SystemServer.java
frameworks/base/tools/layoutlib/bridge/src/android/os/HandlerThread_Delegate.java
frameworks/base/tools/layoutlib/bridge/src/com/android/layoutlib/bridge/Bridge.java

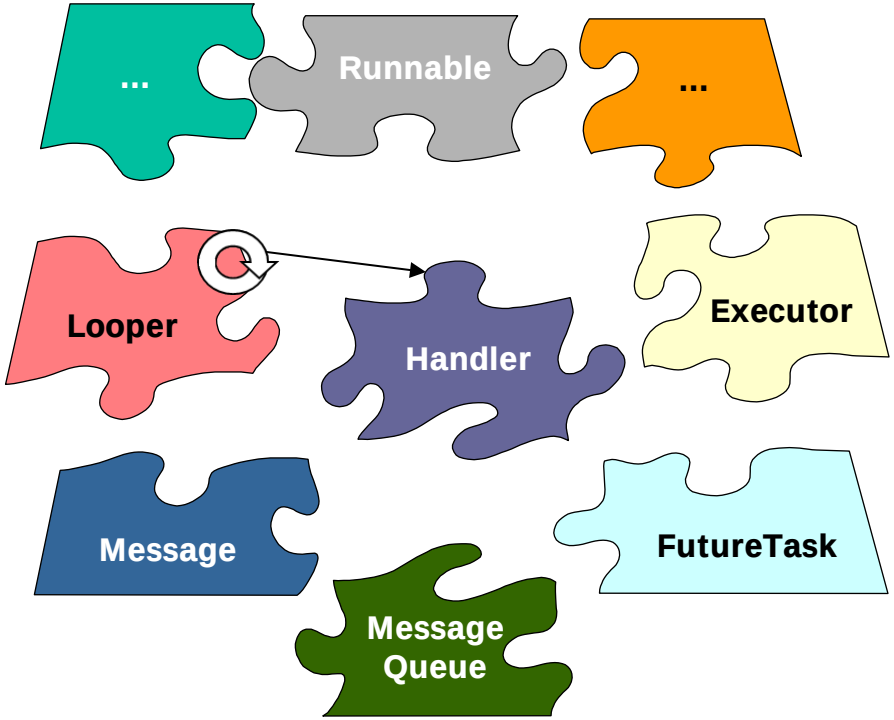
Summary

- A Thread can contain a Looper
- A Looper is often invisible to application software components
- Loopers are used throughout Android

frameworks/base/core/java/android/os/storage/StorageManager.java
frameworks/base/core/java/android/os/AsyncTask.java
frameworks/base/core/java/android/os/Handler.java
frameworks/base/core/java/android/os/HandlerThread.java
frameworks/base/core/java/android/os/Looper.java
frameworks/base/core/java/android/os/MessageQueue.java

Summary

- A Thread can contain a Looper
- A Looper is often invisible to application software components
- Loopers are used throughout Android



They play a key role in providing inversion of control to concurrency frameworks