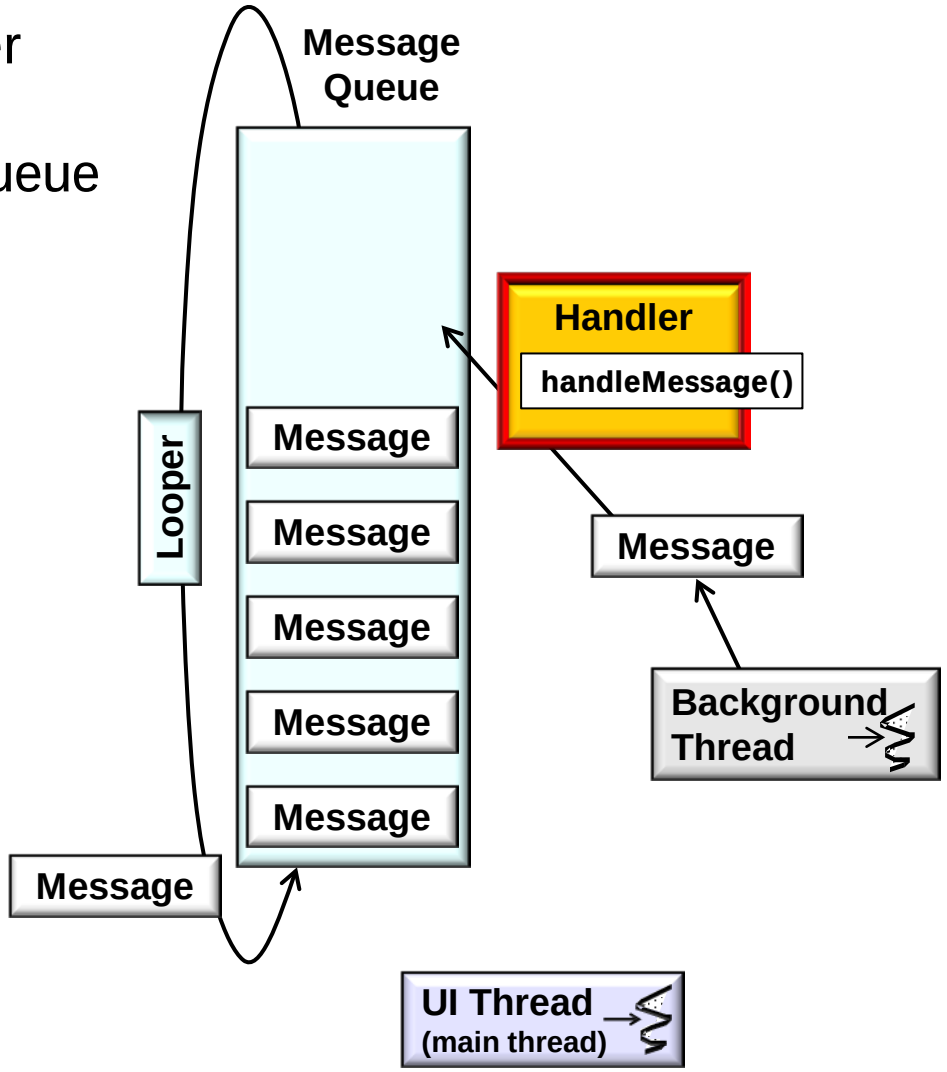
Android Concurrency: Sending & Handling Messages with Android Handler

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

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

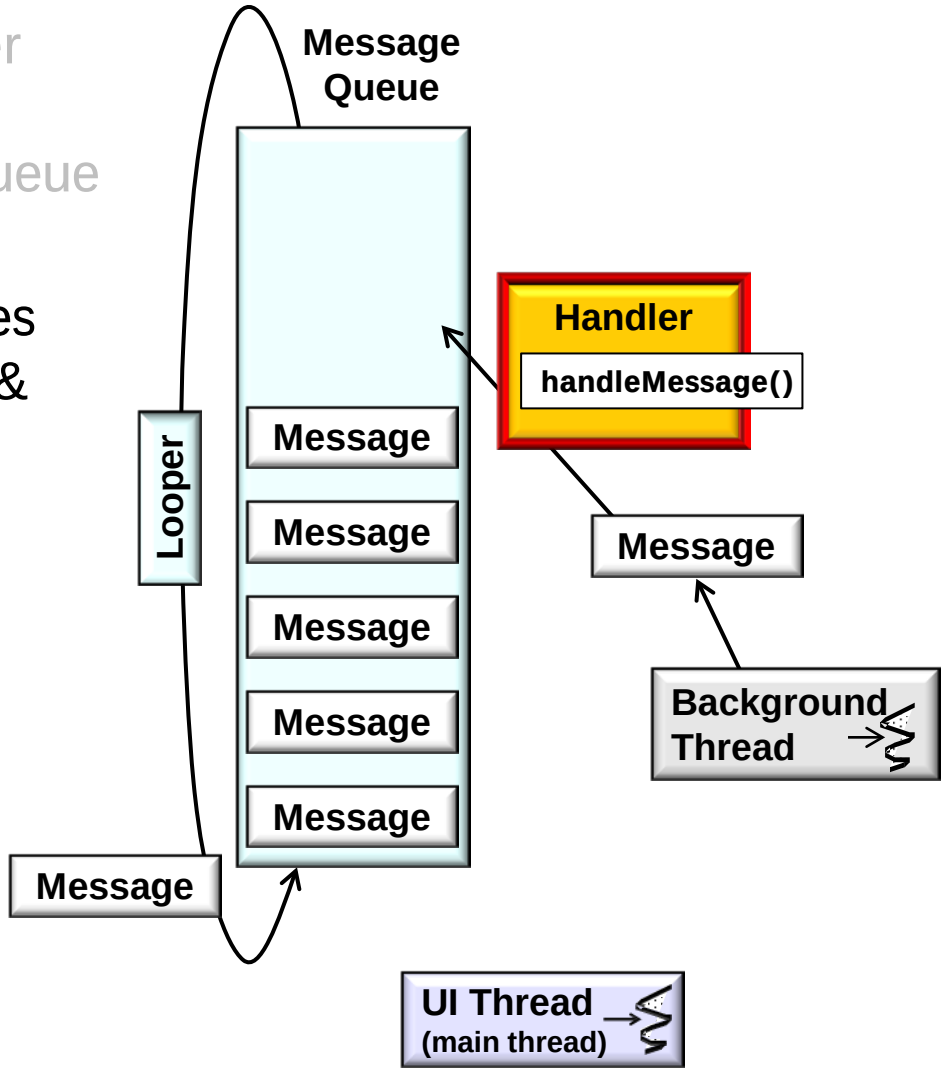
Learning Objectives in this Part of the Module

- Understand how an Android Handler enables the sending & handling of Message objects via the MessageQueue associated with a Thread's Looper



Learning Objectives in this Part of the Module

- Understand how an Android Handler enables the sending & handling of Message objects via the MessageQueue associated with a Thread's Looper
- Recognize how Handlers & Messages are applied in Android applications & its HaMeR concurrency framework



Handler Methods that Send & Handle Messages

- Handler defines methods for sending & removing Messages from Handler's 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.

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

Handler Methods that Send & Handle Messages

- Handler defines methods for sending & removing Messages from Handler's MessageQueue
  - Message contains data send to the Handler

Message

Inherited Methods | [Expand All]

Added in API level 1

extends Object
implements Parcelable

java.lang.Object
↳ android.os.Message

Class Overview

Defines a message containing a description and arbitrary data object that can be sent to a `Handler`. This object contains two extra int fields and an extra object field that allow you to not do allocations in many cases.

> While the constructor of Message is public, the best way to get one of these is to call `Message.obtain()` or one of the `Handler.obtainMessage()` methods, which will pull them from a pool of recycled objects.

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

Handler Methods that Send & Handle Messages

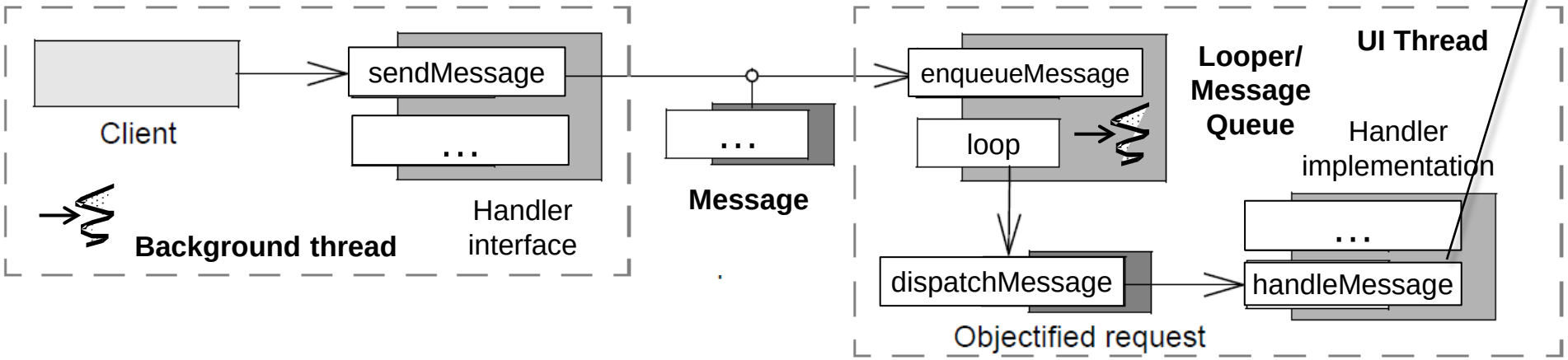
- Handler defines methods for sending & removing Messages from Handler's MessageQueue
  - Message contains data send to the Handler

Fields		
public int	arg1	arg1 and arg2 are lower-cost alternatives to using `setData()` if you only need to store a few integer values.
public int	arg2	arg1 and arg2 are lower-cost alternatives to using `setData()` if you only need to store a few integer values.
public Object	obj	An arbitrary object to send to the recipient.
public Messenger	replyTo	Optional Messenger where replies to this message can be sent.
public int	what	User-defined message code so that the recipient can identify what this message is about.

Handler Methods that Send & Handle Messages

- Handler defines methods for sending & removing Messages from Handler's MessageQueue
  - Message contains data send to the Handler
    - Data processed by Handler's handleMessage() hook method

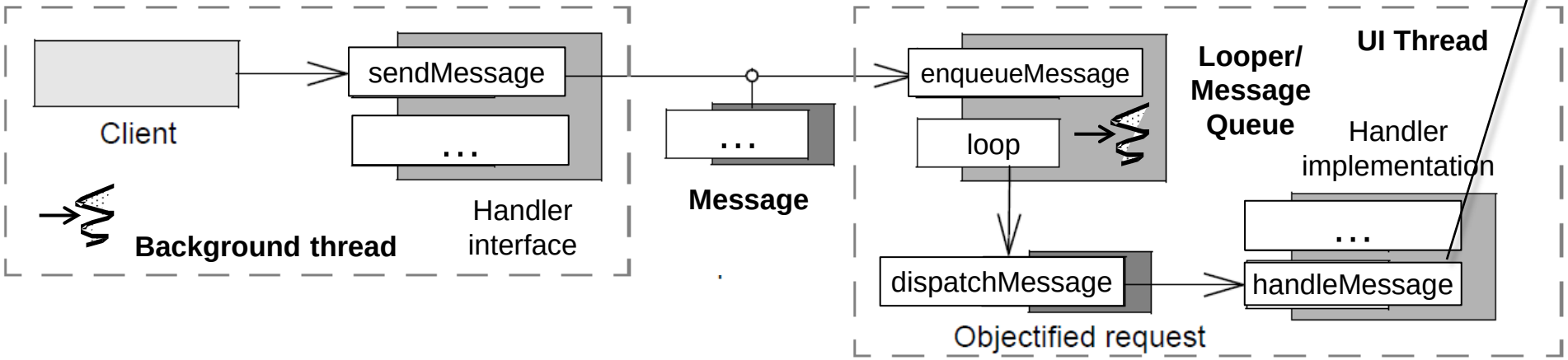
```
Handler handler =
  new Handler() {
    public void handleMessage
                  (Message msg) {
    ...
    onTick(...);
    ...
    onFinished();
    ...
```



Handler Methods that Send & Handle Messages

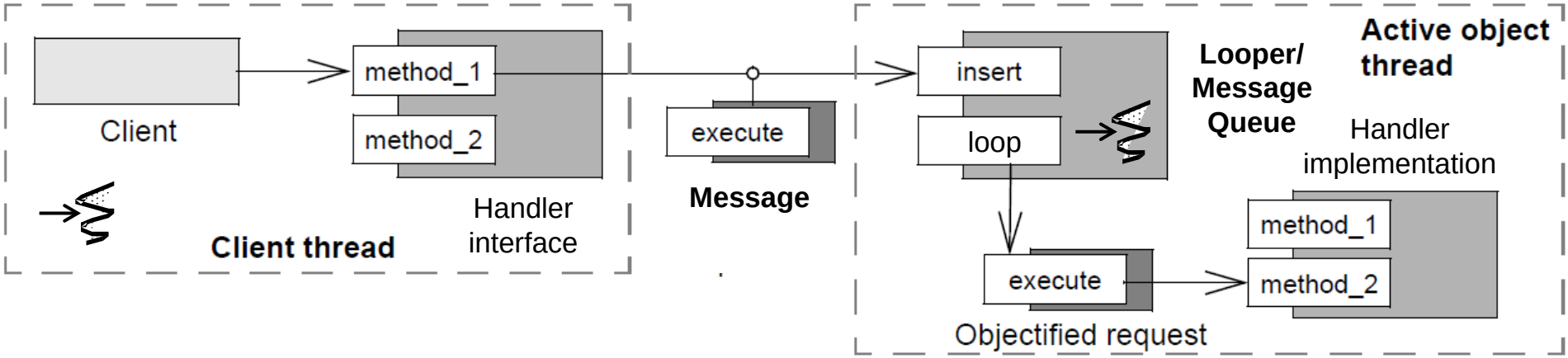
- Handler defines methods for sending & removing Messages from Handler's MessageQueue
  - Message contains data send to the Handler
    - Data processed by Handler's handleMessage() hook method
    - handleMessage() is processed in the Handler Thread

```
Handler handler =
  new Handler() {
    public void handleMessage
                  (Message msg) {
    ...
    onTick(...);
    ...
    onFinished();
    ...
```



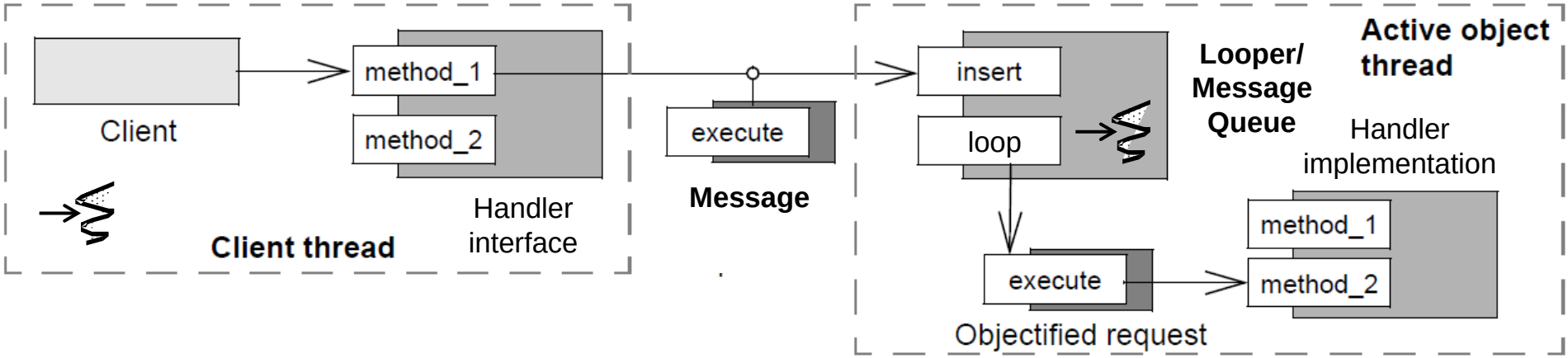
Handler Methods that Send & Handle Messages

- Handler defines methods for sending & removing Messages from Handler's MessageQueue
- Implements a variant of the *Active Object* pattern



Handler Methods that Send & Handle Messages

- Handler defines methods for sending & removing Messages from Handler's MessageQueue
- Implements a variant of the *Active Object* pattern
  - Processes service requests in a different thread than client that invoked the requests



www.dre.vanderbilt.edu/~schmidt/PDF/Act-Obj.pdf has more info

Handler Methods that Send & Handle Messages

Handler Methods that Send & Handle Messages

- Three categories of Handler methods send & handle Messages

Handler Methods that Send & Handle Messages

- Three categories of Handler methods send & handle Messages
  - Sending/removing Messages

`boolean sendMessage(Message msg)`

- Puts msg at rear of queue immediately

`boolean sendMessageAtFrontOfQueue (Message msg)`

- Puts msg at front of queue immediately

`boolean sendMessageDelayed (Message msg, long delayMillis)`

- Puts msg after delay time has passed

`boolean sendMessageAtTime (Message msg, long uptimeMillis)`

- Puts msg on queue at stated time

`boolean sendEmptyMessage(int what)`

- Send an empty message with ‘what’ value

Handler Methods that Send & Handle Messages

- Three categories of Handler methods send & handle Messages
  - Sending/removing Messages
    - Handle Messages as soon as possible

`boolean sendMessage(Message msg)`

- Puts msg at rear of queue immediately

`boolean sendMessageAtFrontOfQueue (Message msg)`

- Puts msg at front of queue immediately

`boolean sendMessageDelayed (Message msg, long delayMillis)`

- Puts msg after delay time has passed

`boolean sendMessageAtTime (Message msg, long uptimeMillis)`

- Puts msg on queue at stated time

`boolean sendEmptyMessage(int what)`

- Send an empty message with ‘what’ value

Handler Methods that Send & Handle Messages

- Three categories of Handler methods send & handle Messages
  - Sending/removing Messages
    - Handle Messages as soon as possible
    - Specify a delay using "relative time"

`boolean sendMessage(Message msg)`

- Puts msg at rear of queue immediately

`boolean sendMessageAtFrontOfQueue (Message msg)`

- Puts msg at front of queue immediately

`boolean sendMessageDelayed (Message msg, long delayMillis)`

- Puts msg after delay time has passed

`boolean sendMessageAtTime (Message msg, long uptimeMillis)`

- Puts msg on queue at stated time

`boolean sendEmptyMessage(int what)`

- Send an empty message with 'what' value

Handler Methods that Send & Handle Messages

- Three categories of Handler methods send & handle Messages
  - Sending/removing Messages
    - Handle Messages as soon as possible
    - Specify a delay using "relative time"
    - Specific a delay using "absolute time"

`boolean sendMessage(Message msg)`

- Puts msg at rear of queue immediately

`boolean sendMessageAtFrontOfQueue (Message msg)`

- Puts msg at front of queue immediately

`boolean sendMessageDelayed (Message msg, long delayMillis)`

- Puts msg after delay time has passed

`boolean sendMessageAtTime (Message msg, long uptimeMillis)`

- Puts msg on queue at stated time

`boolean sendEmptyMessage(int what)`

- Send an empty message with 'what' value

Handler Methods that Send & Handle Messages

- Three categories of Handler methods send & handle Messages
  - Sending/removing Messages
    - Handle Messages as soon as possible
    - Specify a delay using "relative time"
    - Specific a delay using "absolute time"

`boolean sendMessage(Message msg)`

- Puts msg at rear of queue immediately

`boolean sendMessageAtFrontOfQueue (Message msg)`

- Puts msg at front of queue immediately

`boolean sendMessageDelayed (Message msg, long delayMillis)`

- Puts msg after delay time has passed

`boolean sendMessageAtTime (Message msg, long uptimeMillis)`

- Puts msg on queue at stated time

`boolean sendEmptyMessage(int what)`

- Send an empty message with 'what' value

These methods support timing related behavior

Handler Methods that Send & Handle Messages

- Three categories of Handler methods send & handle Messages
  - Sending/removing Messages
    - Handle Messages as soon as possible
    - Specify a delay using "relative time"
    - Specific a delay using "absolute time"
    - Send an "empty" Message

`boolean sendMessage(Message msg)`

- Puts msg at rear of queue immediately

`boolean sendMessageAtFrontOfQueue (Message msg)`

- Puts msg at front of queue immediately

`boolean sendMessageAtTime (Message msg, long uptimeMillis)`

- Puts msg on queue at stated time

`boolean sendMessageDelayed (Message msg, long delayMillis)`

- Puts msg after delay time has passed

`boolean sendEmptyMessage(int what)`

- Send an empty message with 'what' value

Handler Methods that Send & Handle Messages

- Three categories of Handler methods send & handle Messages
  - Sending/removing Messages
    - Handle Messages as soon as possible
    - Specify a delay using "relative time"
    - Specific a delay using "absolute time"
    - Send an "empty" Message
    - There are several variants of remove()

```
final void removeMessages(int
                          what)
```

- Remove any pending posts of messages with code 'what' that are in the MessageQueue

```
final void removeMessages
        (int what, Object object)
```

- Remove any pending posts of messages with code 'what' & whose obj is 'object' that are in the MessageQueue

Handler Methods that Send & Handle Messages

- Three categories of Handler methods send & handle Messages
  - Sending/removing Messages
  - Obtaining Messages

`Message obtainMessage()`

- Returns a new Message from the global message pool

`Message obtainMessage (int what)`

- Same as obtainMessage(), except that it also sets the what member of the returned Message

`Message obtainMessage (int what, int arg1, int arg2, Object obj)`

- Same as obtainMessage(), except that it also sets the what, obj, arg1,and arg2 values on the returned Message

Handler Methods that Send & Handle Messages

- Three categories of Handler methods send & handle Messages
  - Sending/removing Messages
  - Obtaining Messages
    - Implement a *Creational* pattern

`Message obtainMessage()`

- Returns a new Message from the global message pool

`Message obtainMessage (int what)`

- Same as obtainMessage(), except that it also sets the what member of the returned Message

`Message obtainMessage (int what, int arg1, int arg2, Object obj)`

- Same as obtainMessage(), except that it also sets the what, obj, arg1,and arg2 values on the returned Message

See en.wikipedia.org/wiki/Creational_pattern

Handler Methods that Send & Handle Messages

- Three categories of Handler methods send & handle Messages
  - Sending/removing Messages
  - Obtaining Messages
    - Implement a *Creational* pattern

`Message obtainMessage()`

- Returns a new Message from the global message pool

`Message obtainMessage (int what)`

- Same as obtainMessage(), except that it also sets the what member of the returned Message

`Message obtainMessage (int what, int arg1, int arg2, Object obj)`

- Same as obtainMessage(), except that it also sets the what, obj, arg1,and arg2 values on the returned Message

Handler Methods that Send & Handle Messages

- Three categories of Handler methods send & handle Messages
  - Sending/removing Messages
  - Obtaining Messages
  - Dispatching/handling Messages

`void dispatchMessage(Message msg)`

- Invoke the appropriate callback (e.g., run() or handleMessage()) based on the type of the Message

`void handleMessage(Message msg)`

- This method must be overridden to receive messages

Handler Methods that Send & Handle Messages

- Three categories of Handler methods send & handle Messages
  - Sending/removing Messages
  - Obtaining Messages
  - Dispatching/handling Messages
    - Receive & process Messages inserted via sendMessage()

`void dispatchMessage(Message msg)`

- Invoke the appropriate callback (e.g., run() or handleMessage()) based on the type of the Message

`void handleMessage(Message msg)`

- This method must be overridden to receive messages

Handler Methods that Send & Handle Messages

- Three categories of Handler methods send & handle Messages
  - Sending/removing Messages
  - Obtaining Messages
  - Dispatching/handling Messages
    - Receive & process Messages inserted via sendMessage()
    - Runs in Thread associated with Handler instance

`void dispatchMessage(Message msg)`

- Invoke the appropriate callback (e.g., run() or handleMessage()) based on the type of the Message

`void handleMessage(Message msg)`

- This method must be overridden to receive messages

Handler Methods that Send & Handle Messages

- Three categories of Handler methods send & handle Messages
  - Sending/removing Messages
  - Obtaining Messages
  - Dispatching/handling Messages
    - Receive & process Messages inserted via sendMessage()
    - Runs in Thread associated with Handler instance
    - Implements a variant of the *Active Object* pattern

`void dispatchMessage(Message msg)`

- Invoke the appropriate callback (e.g., run() or handleMessage()) based on the type of the Message

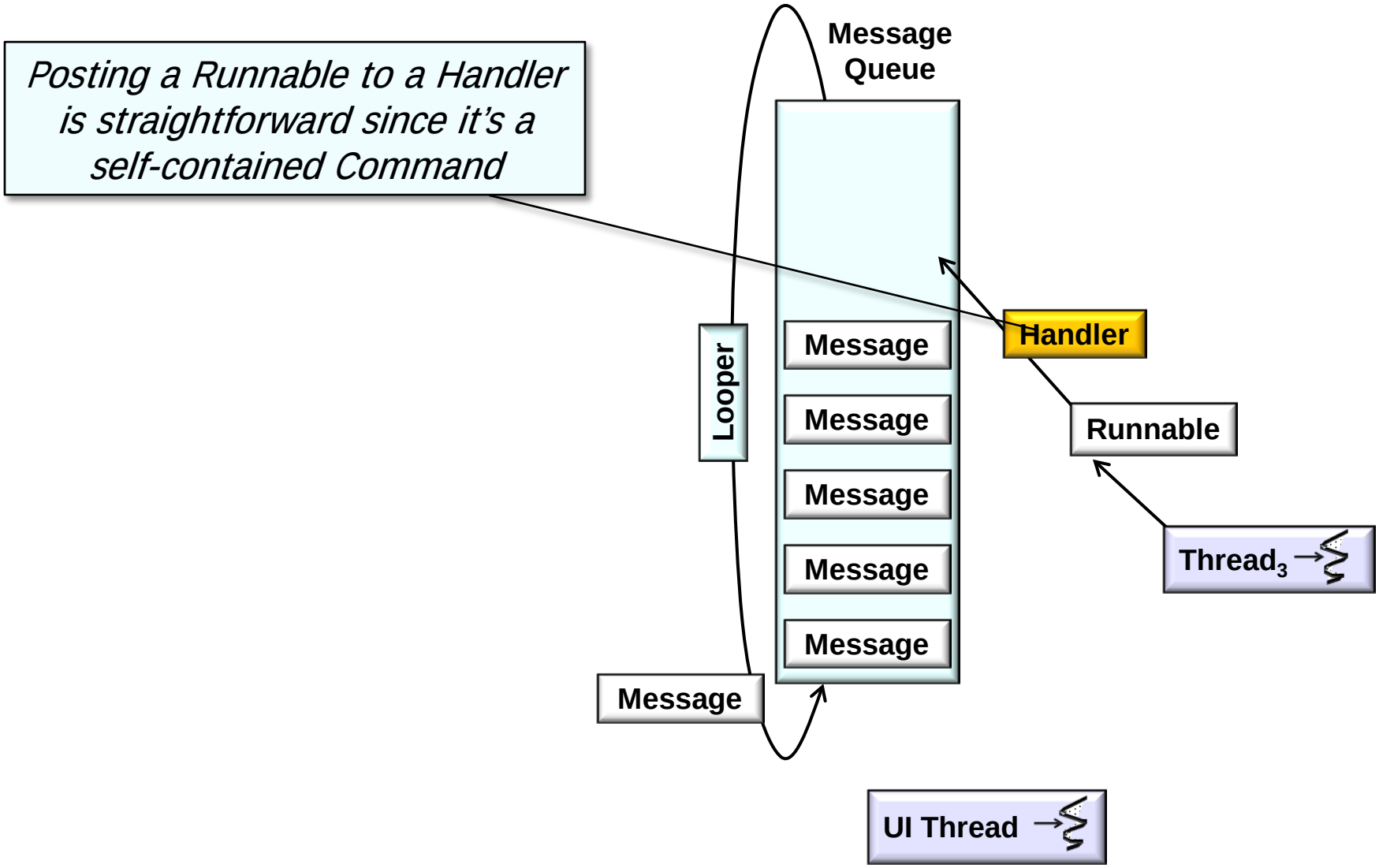
`void handleMessage(Message msg)`

- This method must be overridden to receive messages

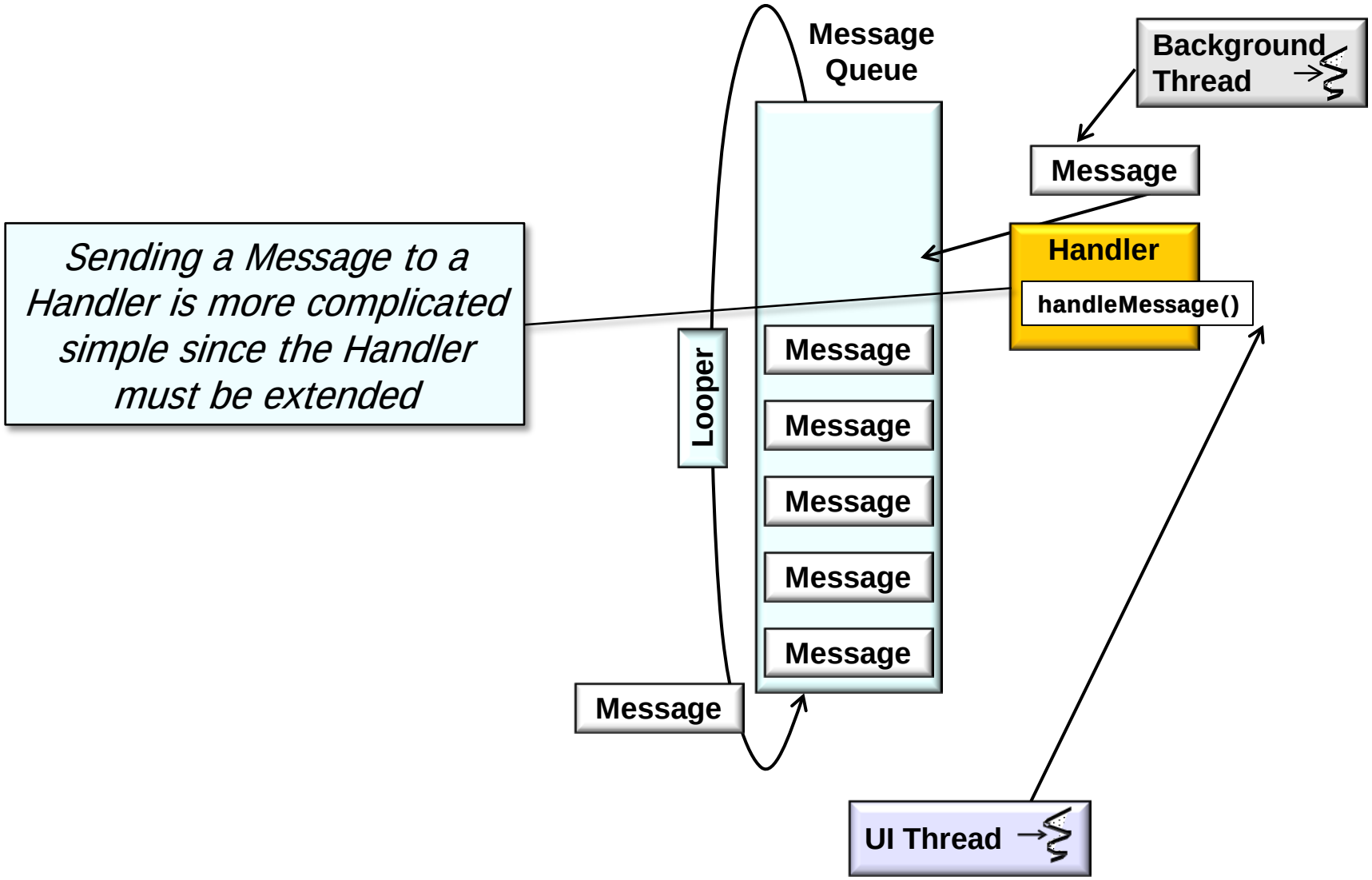
See upcoming parts on the "*Active Object* pattern"

Sending & Handling Messages with the HaMeR Framework (Part 1)

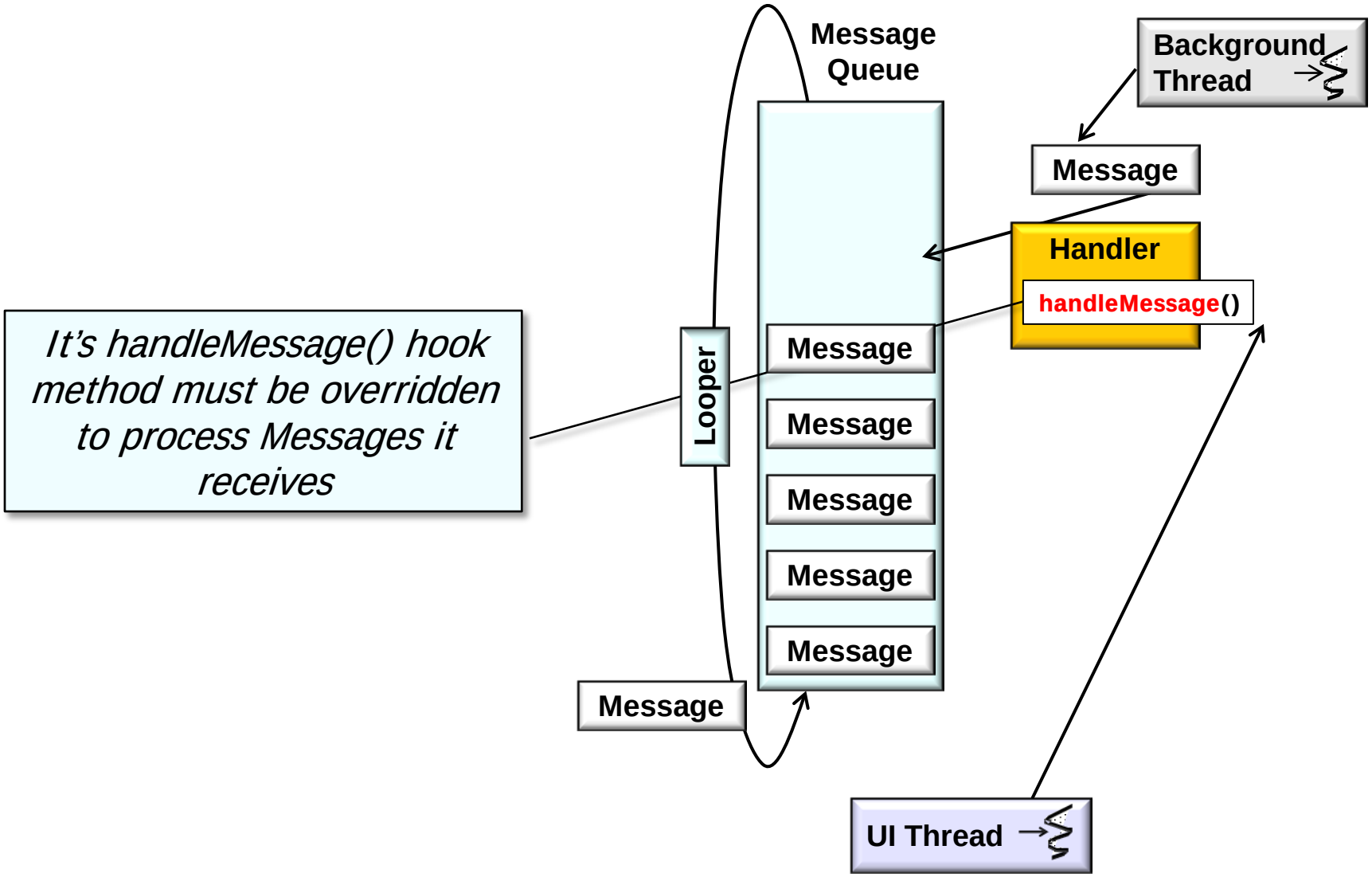
Example of Sending & Handling Messages



Example of Sending & Handling Messages

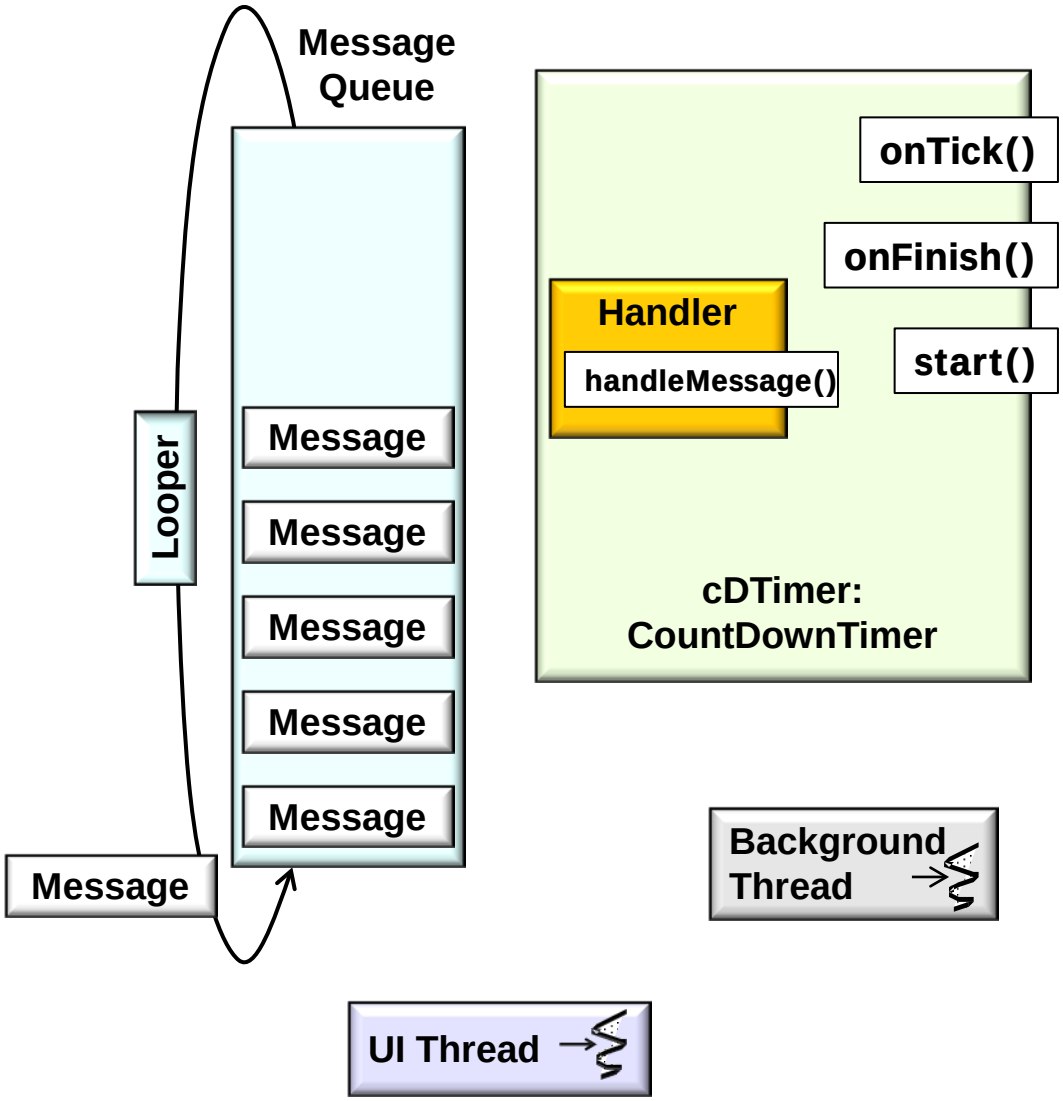


Example of Sending & Handling Messages



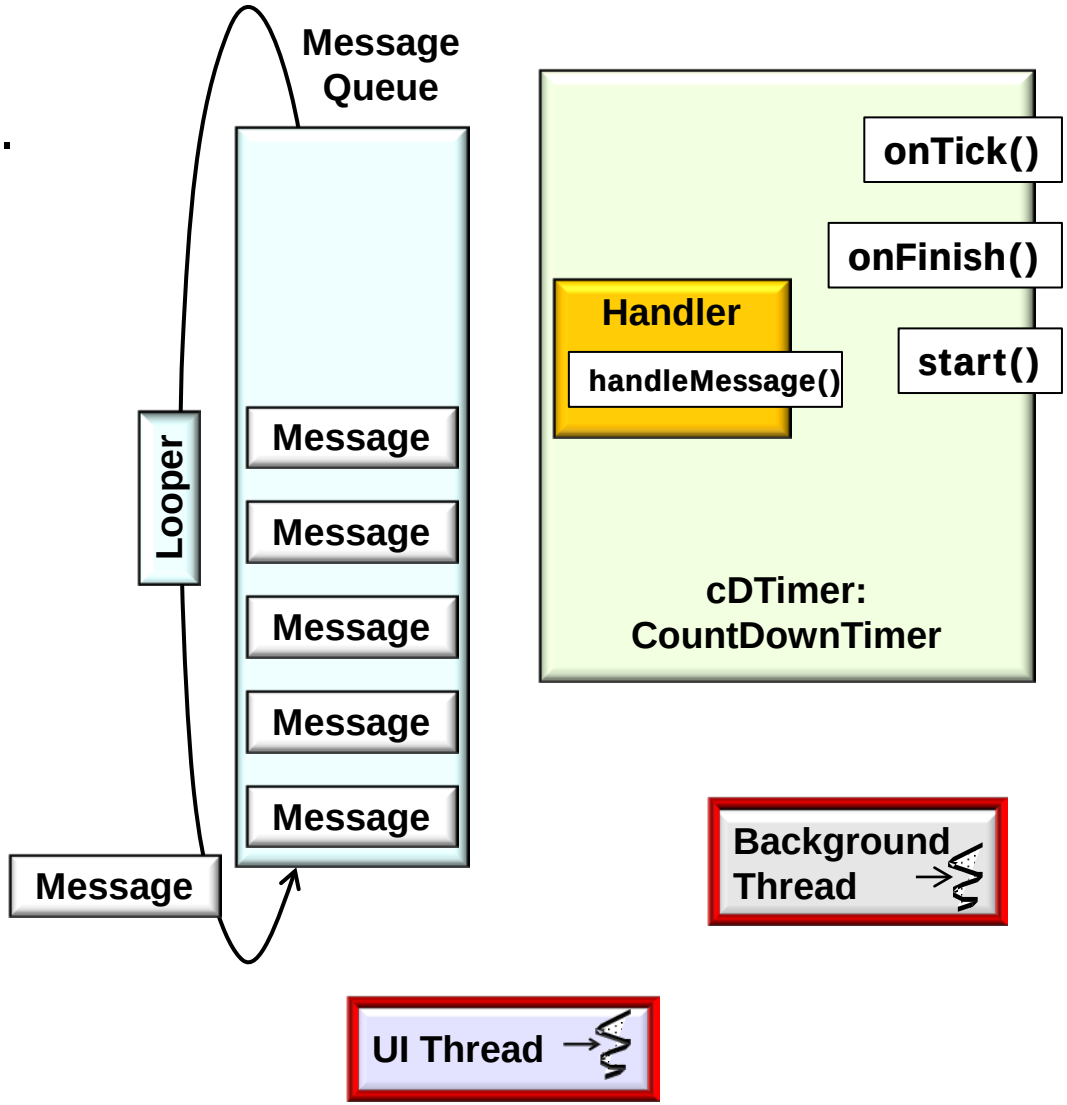
Example of Sending & Handling Messages

- Use a Handler to send & process Messages in several Threads



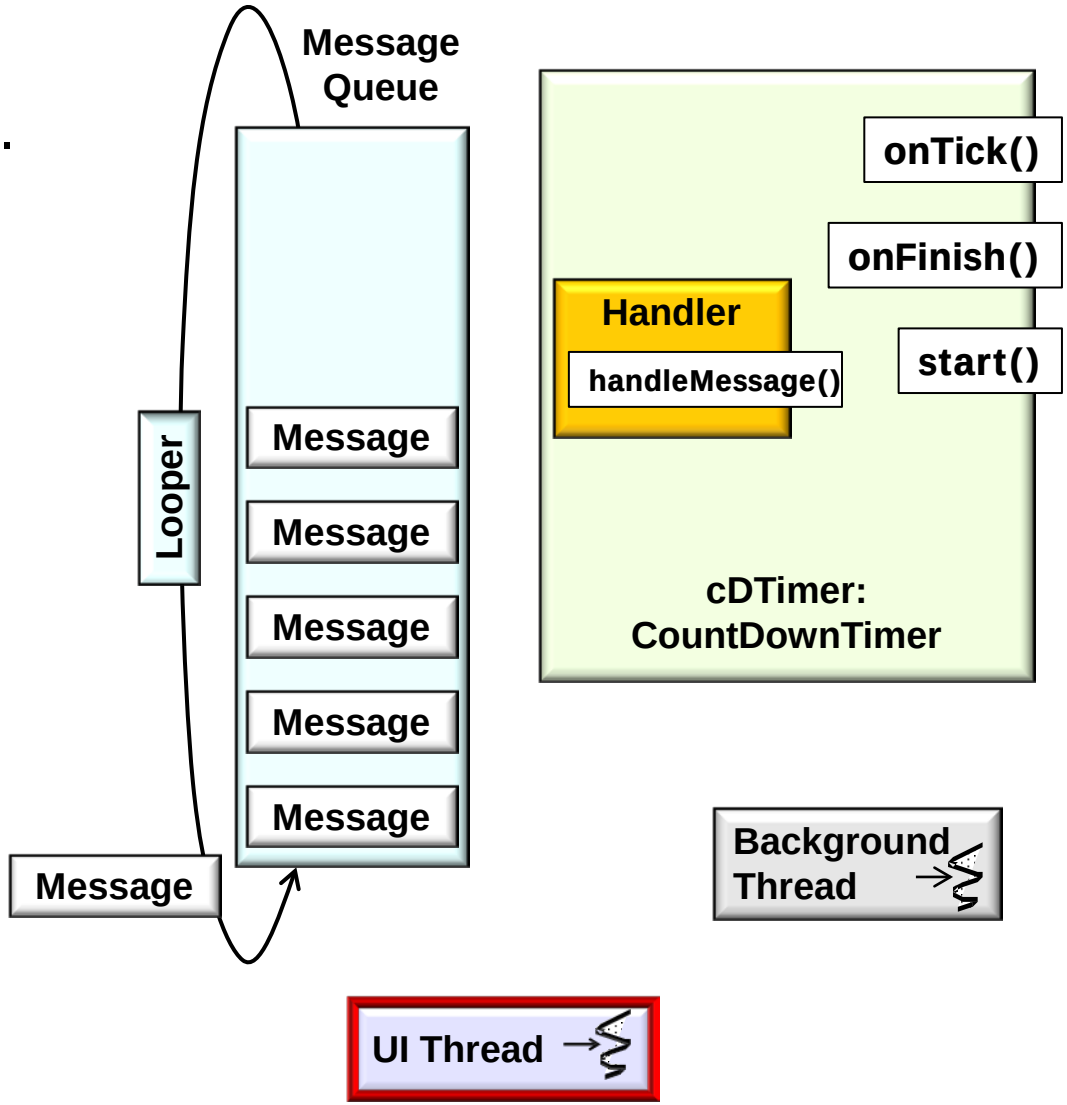
Example of Sending & Handling Messages

- Use a Handler to send & process Messages in several Threads, e.g.
  - From a background Thread to the UI Thread



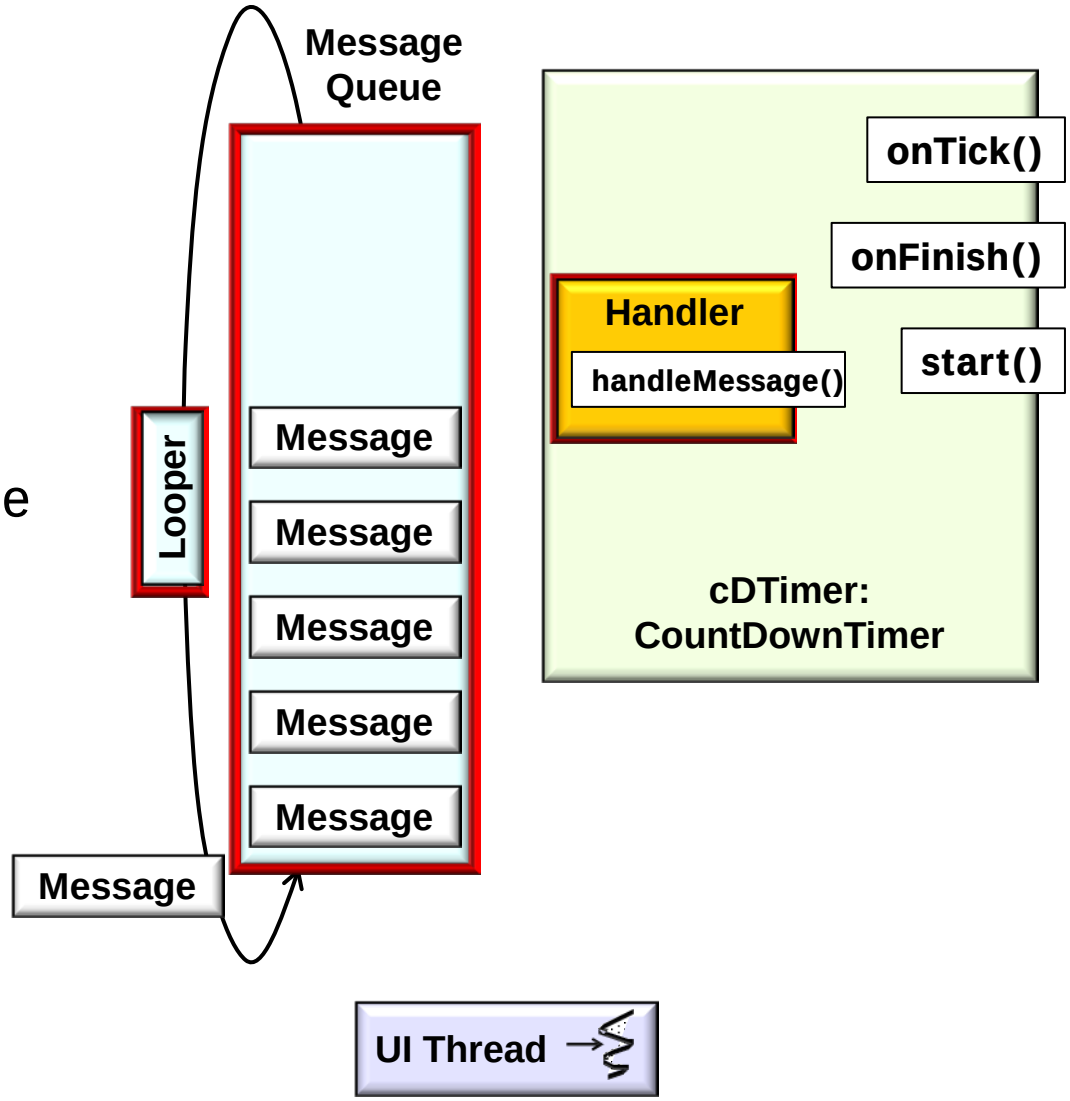
Example of Sending & Handling Messages

- Use a Handler to send & process Messages in several Threads, e.g.
  - From a background Thread to the UI Thread
  - From UI Thread to itself at periodic time intervals



Example of Sending & Handling Messages

- Use a Handler to send & process Messages in several Threads
  - From a background Thread to the UI Thread
  - From UI Thread to itself at periodic time intervals
  - It also shows how classes in the Android HaMeR concurrency framework collaborate



See previous part on “Posting & Processing Runnables with Android Handler”

Example of Sending & Handling Messages

- Use a Handler to send & process Messages in several Threads
- A CountDownTimer schedules a countdown until a future time

CountDownTimer

Added in API level 1

extends Object

java.lang.Object
↳ android.os.CountDownTimer

Class Overview

Schedule a countdown until a time in the future, with regular notifications on intervals along the way. Example of showing a 30 second countdown in a text field:

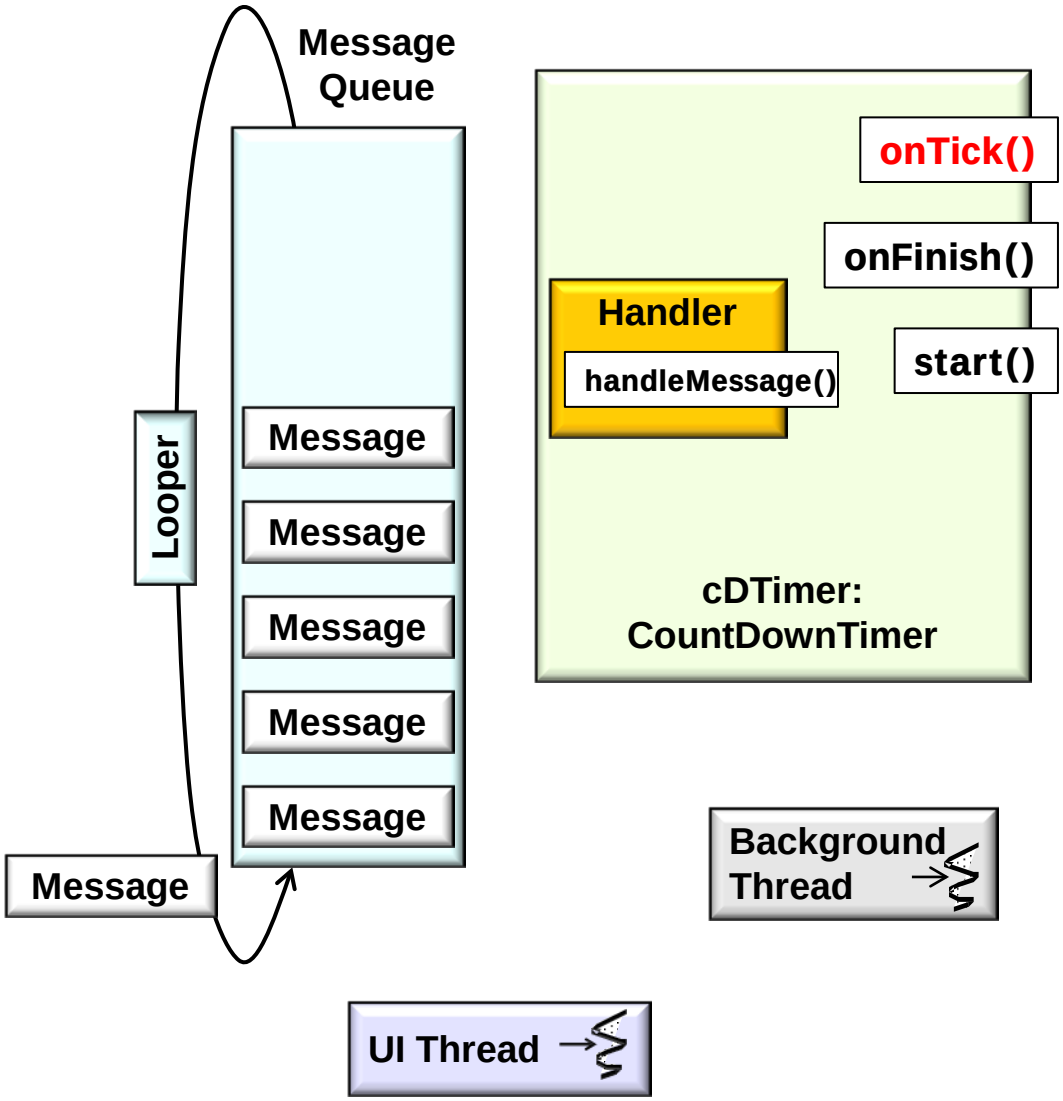
```
new CountDownTimer(30000, 1000) {

    public void onTick(long millisUntilFinished) {
        mTextField.setText("seconds remaining: " + millisUntilFir
    }

    public void onFinish() {
        mTextField.setText("done!");
    }
}.start();
```

The calls to `onTick(long)` are synchronized to this object so that one call to `onTick(long)` won't ever occur before the previous callback is complete. This is only relevant when the implementation of `onTick(long)` takes an amount of time to execute that is significant compared to the countdown interval.

See developer.android.com/reference/android/os/CountDownTimer.html

Example of Sending & Handling Messages

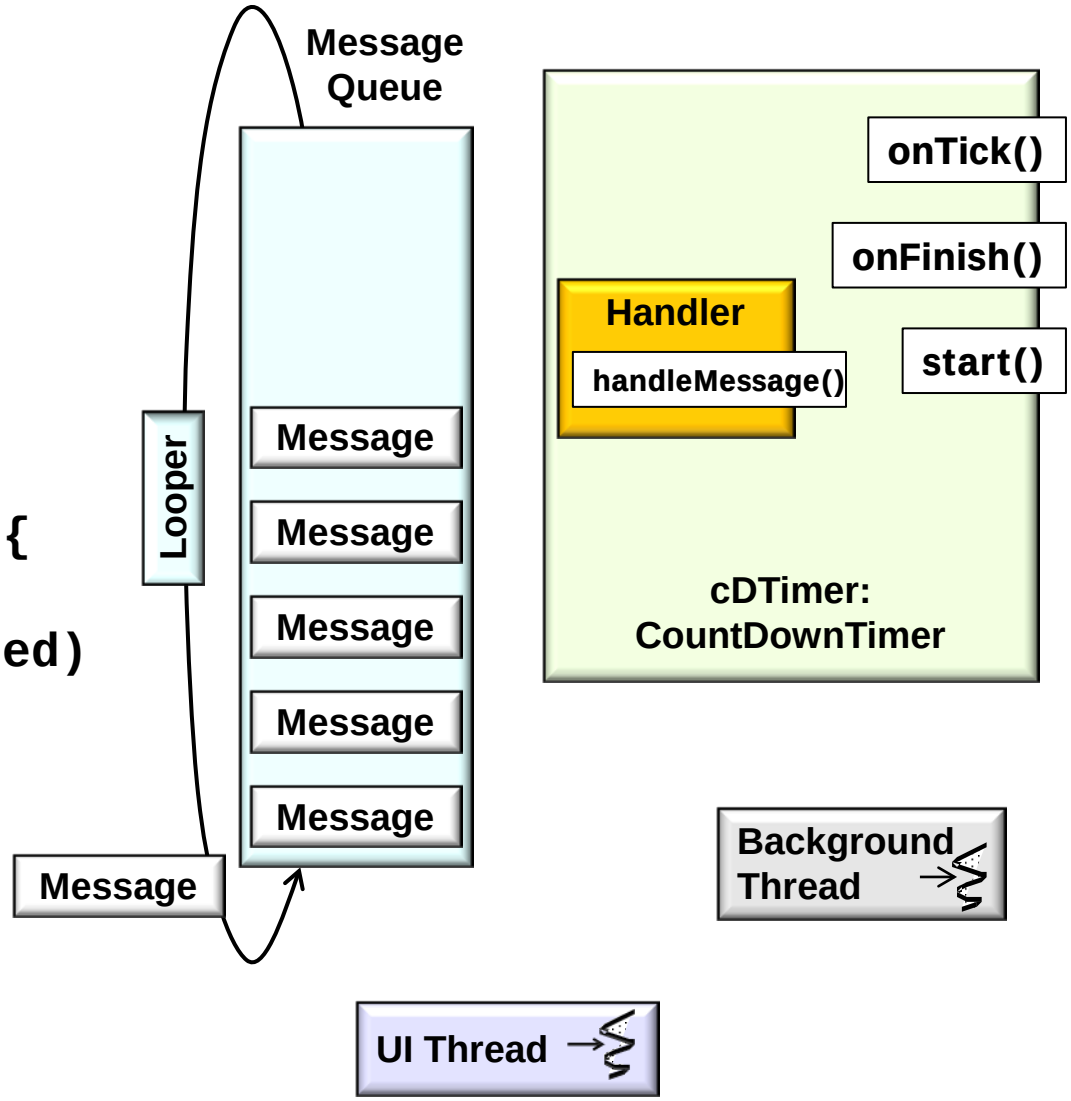
- Use a Handler to send & process Messages in several Threads
- A CountDownTimer schedules a countdown until a future time



Example of Sending & Handling Messages

- Use a Handler to send & process Messages in several Threads
- A CountDownTimer schedules a countdown until a future time

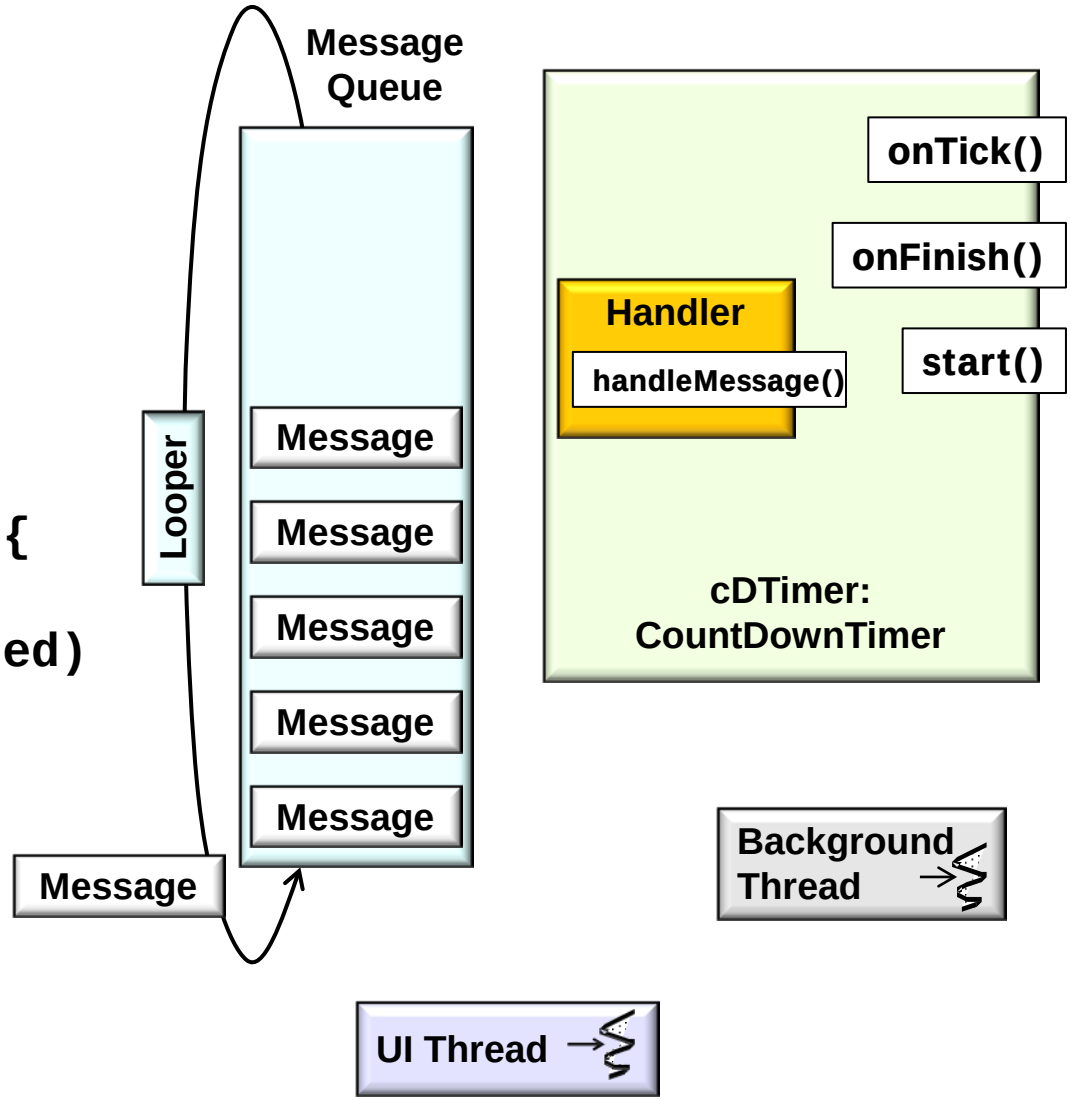
```
// UI Thread
CountDownTimer cDTimer =
  new CountDownTimer(30000,
                     1000) {
  public void onTick
    (long millisUntilFinished)
  { ... }

  public void onFinish()
  { ... }
};
```



Example of Sending & Handling Messages

- Use a Handler to send & process Messages in several Threads
- A CountDownTimer schedules a countdown until a future time

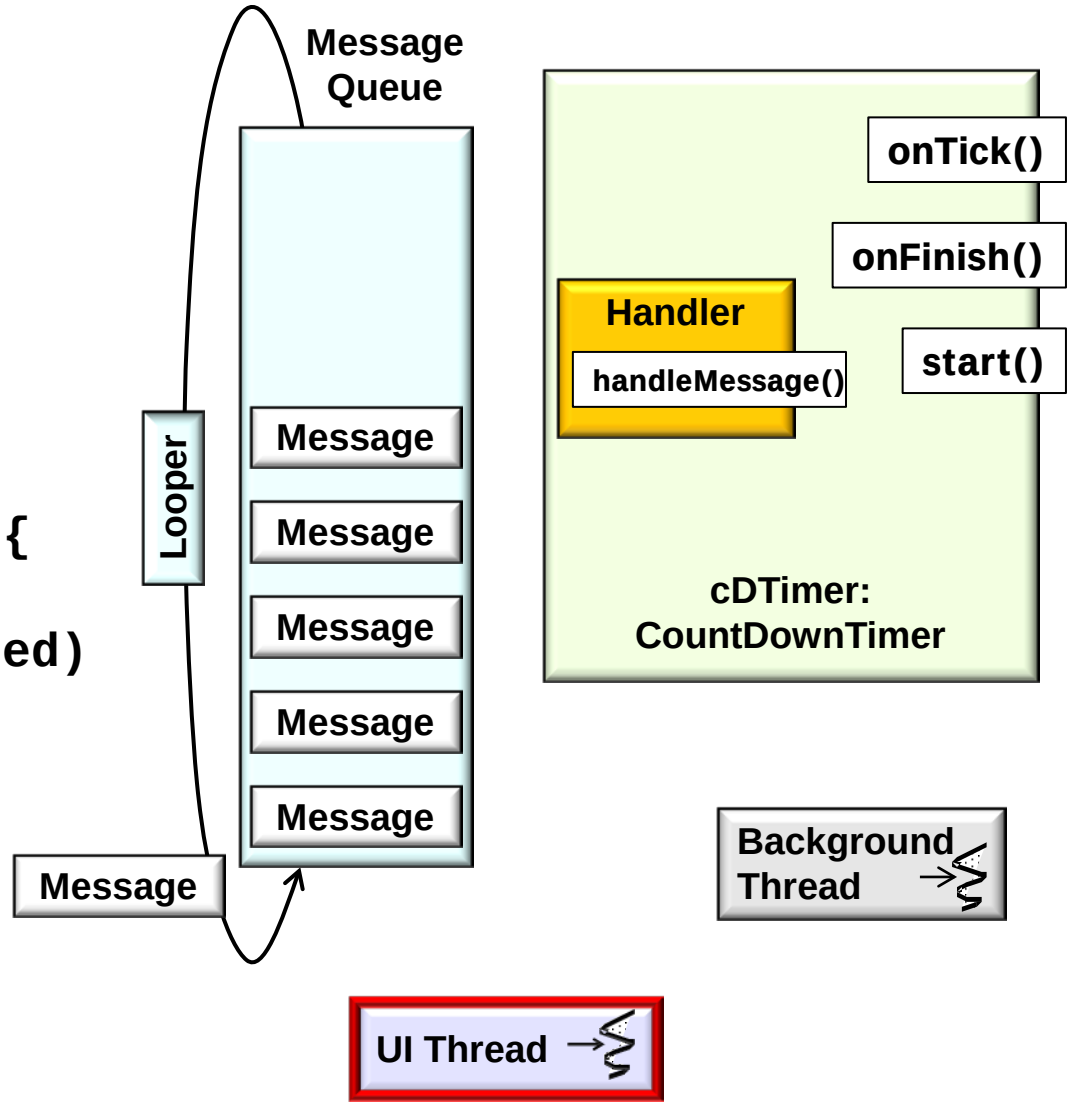
```
// UI Thread
CountDownTimer cDTimer =
  new CountDownTimer(30000,
                     1000) {
  public void onTick
    (long millisUntilFinished)
  { ... }

  public void onFinish()
  { ... }
};
```



Example of Sending & Handling Messages

- Use a Handler to send & process Messages in several Threads
- A CountDownTimer schedules a countdown until a future time

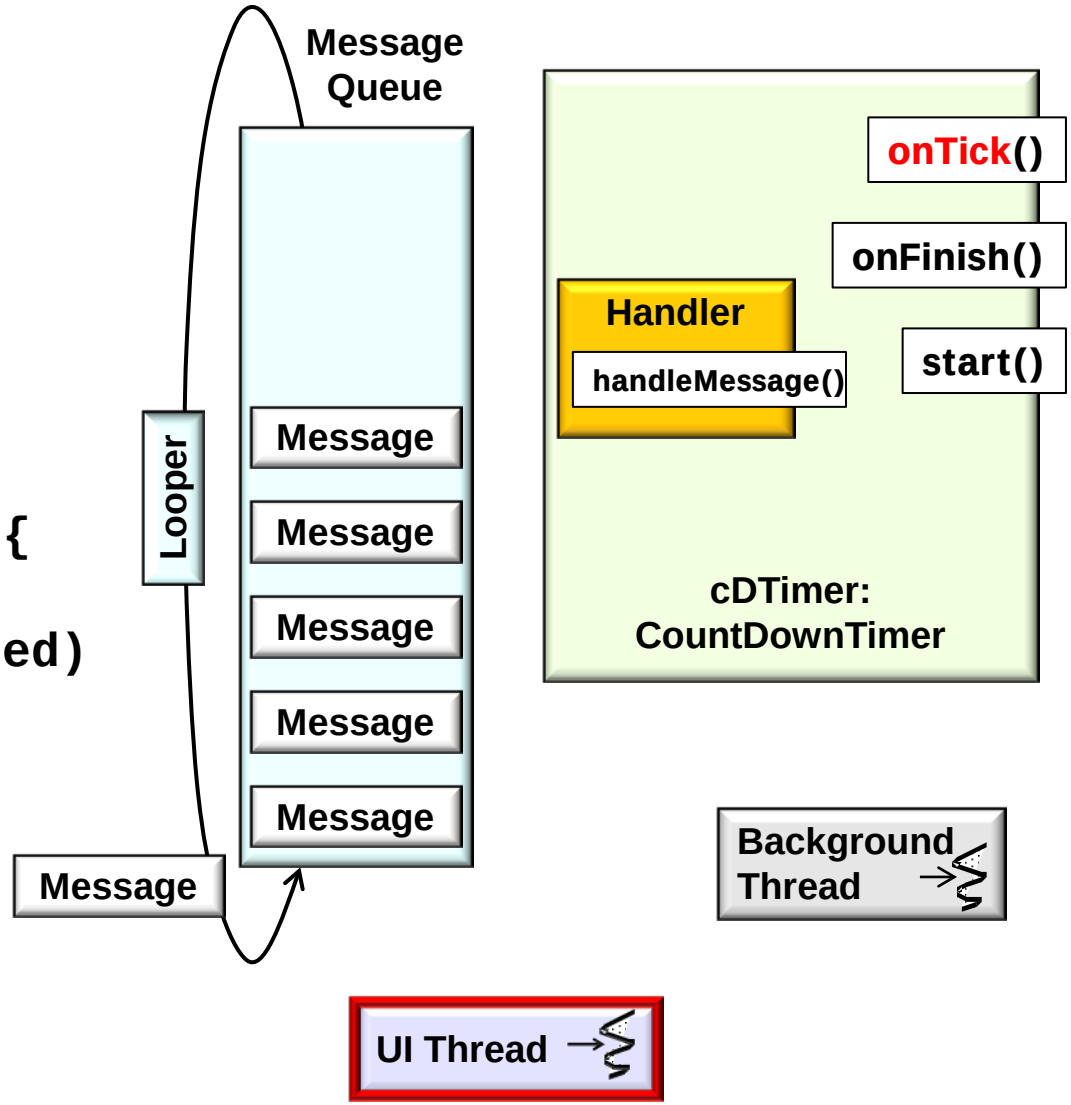
```
// UI Thread
CountDownTimer cDTimer =
  new CountDownTimer(30000,
                     1000) {
  public void onTick
    (long millisUntilFinished)
  { ... }

  public void onFinish()
  { ... }
};
```



Example of Sending & Handling Messages

- Use a Handler to send & process Messages in several Threads
- A CountDownTimer schedules a countdown until a future time

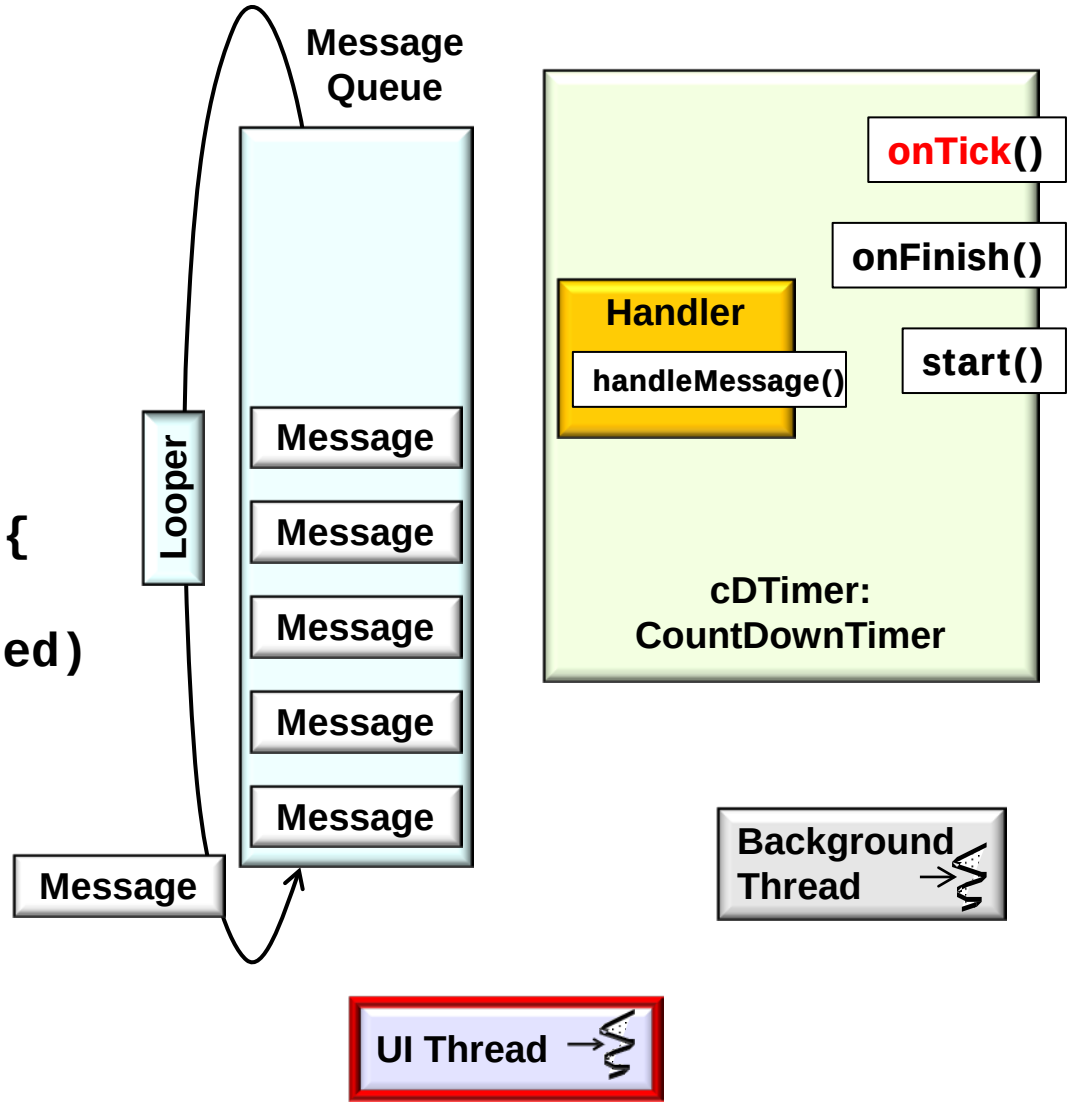
```
// UI Thread
CountDownTimer cDTimer =
  new CountDownTimer(30000,
                     1000) {
  public void onTick
    (long millisUntilFinished)
  { ... }

  public void onFinish()
  { ... }
};
```



Example of Sending & Handling Messages

- Use a Handler to send & process Messages in several Threads
- A CountDownTimer schedules a countdown until a future time

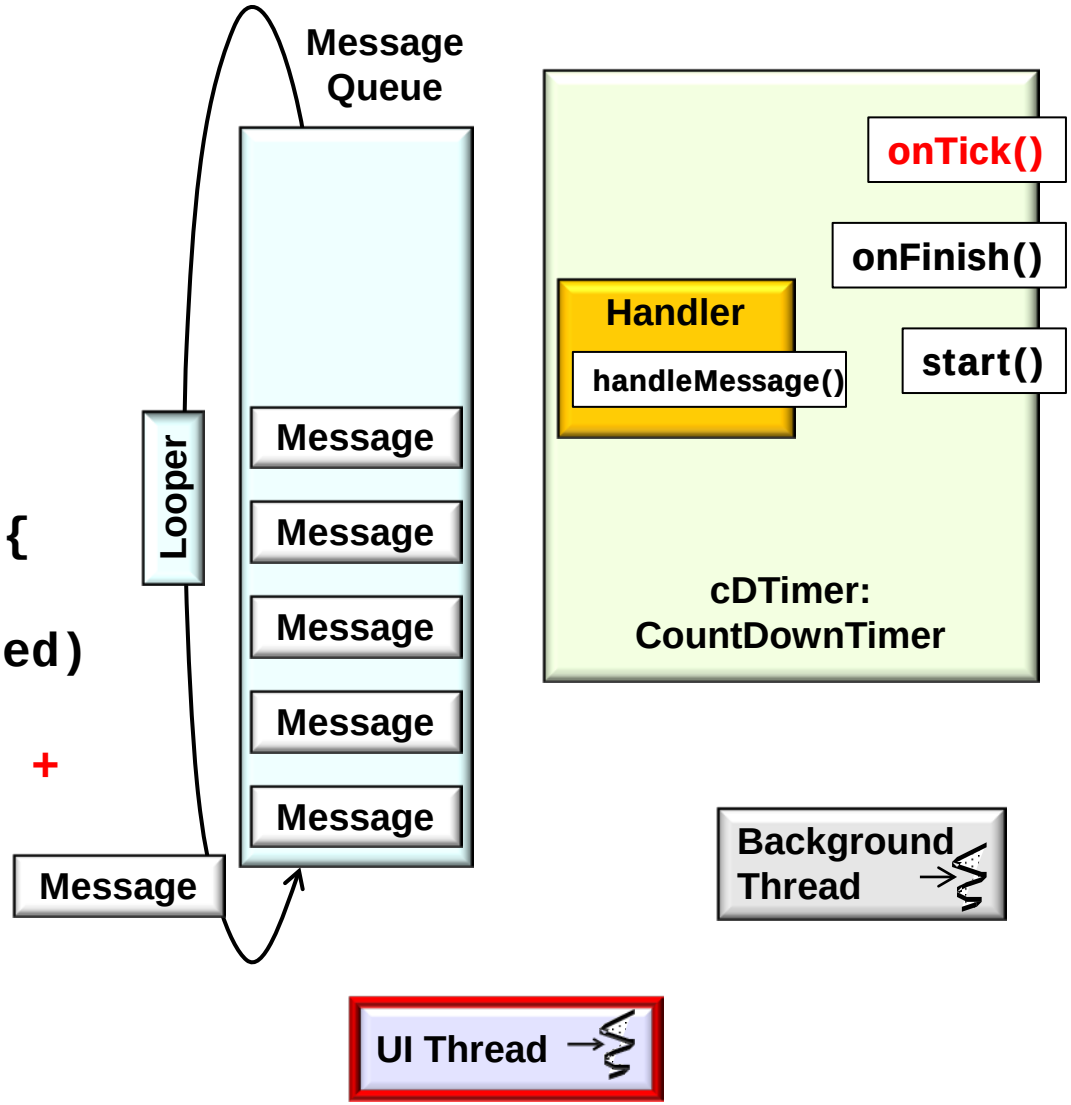
```
// UI Thread
CountDownTimer cDTimer =
  new CountDownTimer(30000,
                     1000) {
  public void onTick
    (long millisUntilFinished)
  { ... }

  public void onFinish()
  { ... }
};
```



Example of Sending & Handling Messages

- Use a Handler to send & process Messages in several Threads
- A CountDownTimer schedules a countdown until a future time

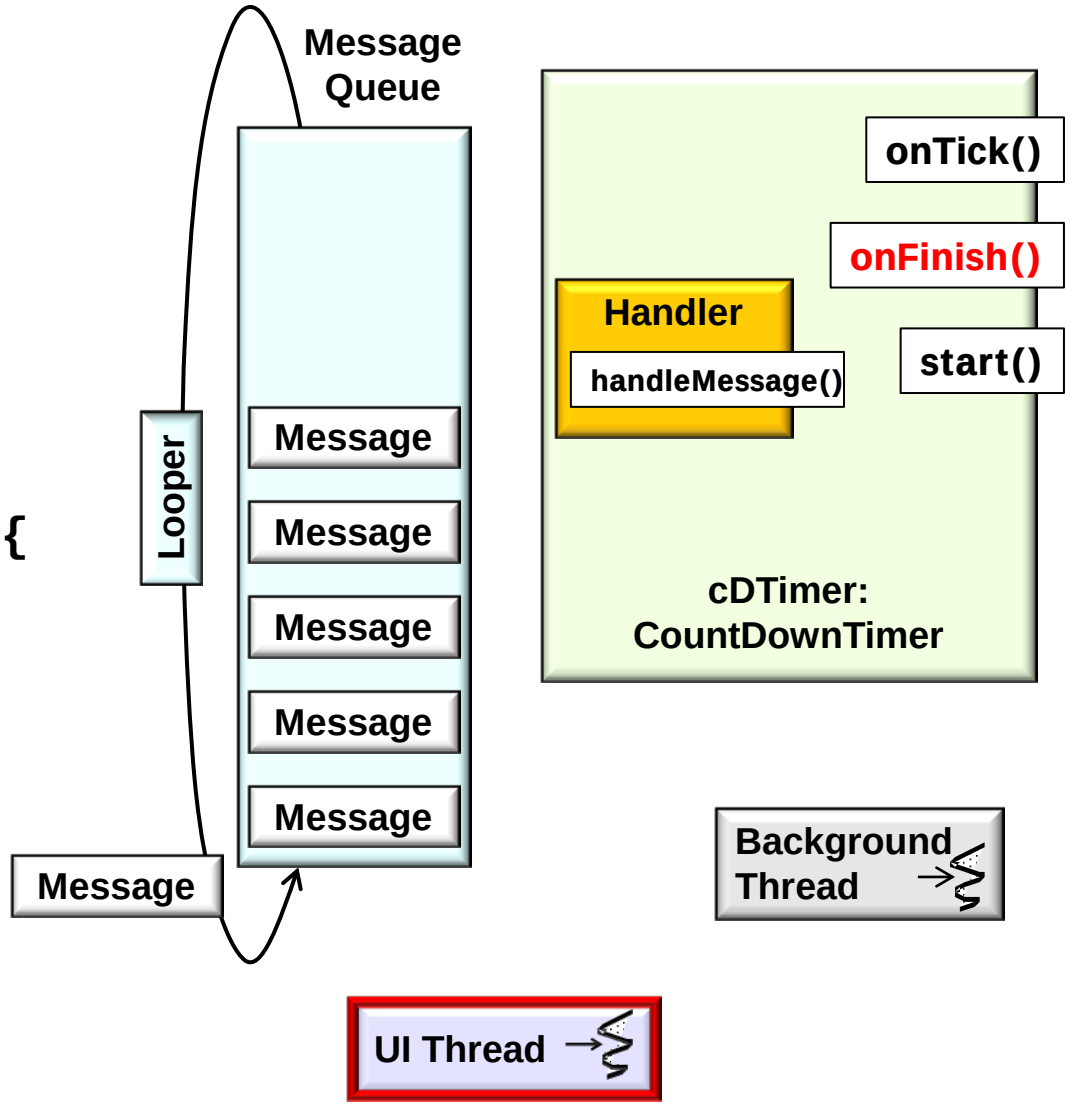
```
// UI Thread
CountDownTimer cDTimer =
  new CountDownTimer(30000,
                     1000) {
  public void onTick
    (long millisUntilFinished)
  { mTextField.setText
     ("seconds remaining: " +
      millisUntilFinished
      / 1000);
  }
  ...
```



Example of Sending & Handling Messages

- Use a Handler to send & process Messages in several Threads
- A CountDownTimer schedules a countdown until a future time

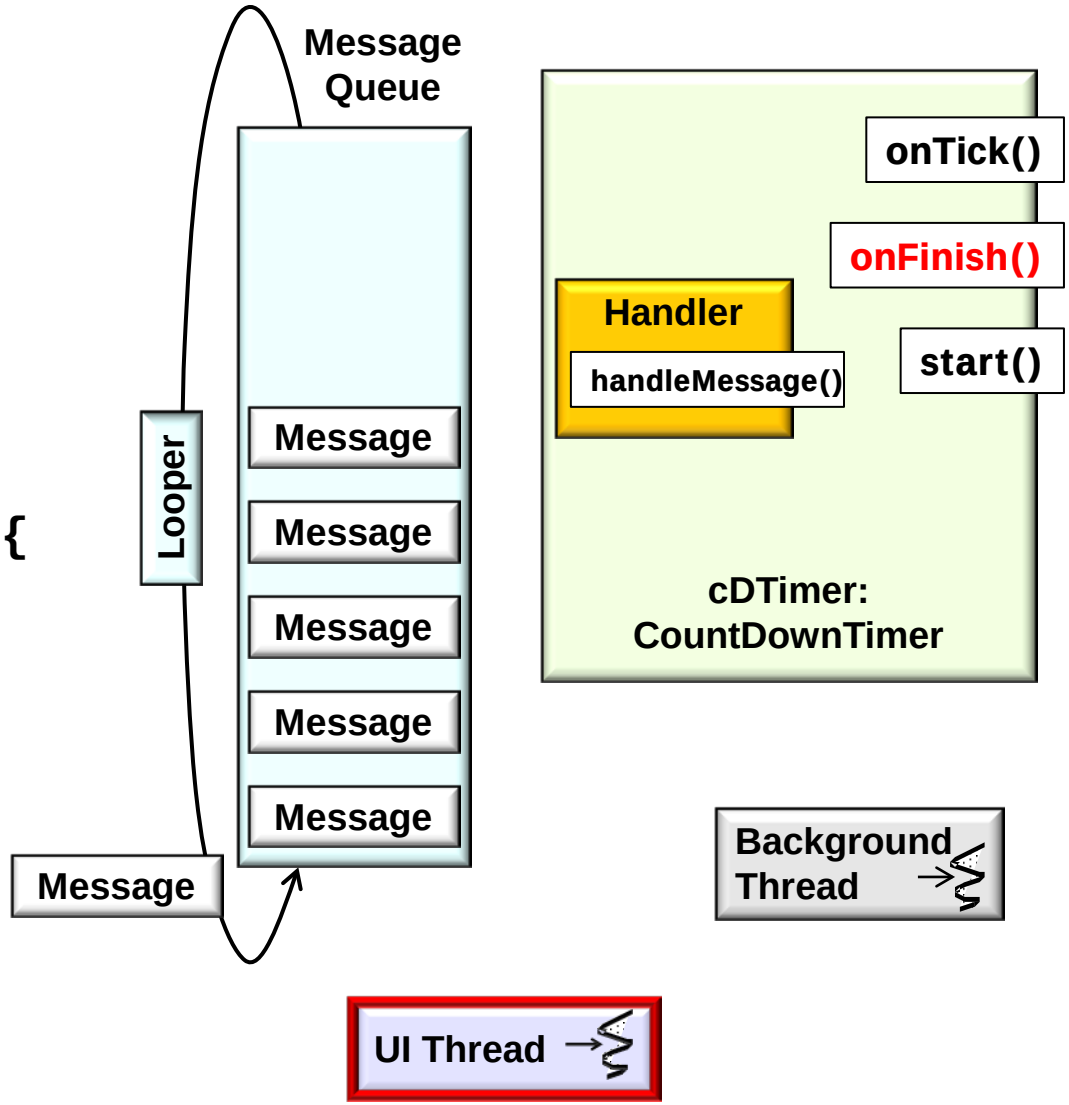
```
// UI Thread
CountDownTimer cDTimer =
  new CountDownTimer(30000,
                     1000) {
  public void onFinish()
  { mTextField.setText
       ("done!");
  }
};
```



Example of Sending & Handling Messages

- Use a Handler to send & process Messages in several Threads
- A CountDownTimer schedules a countdown until a future time

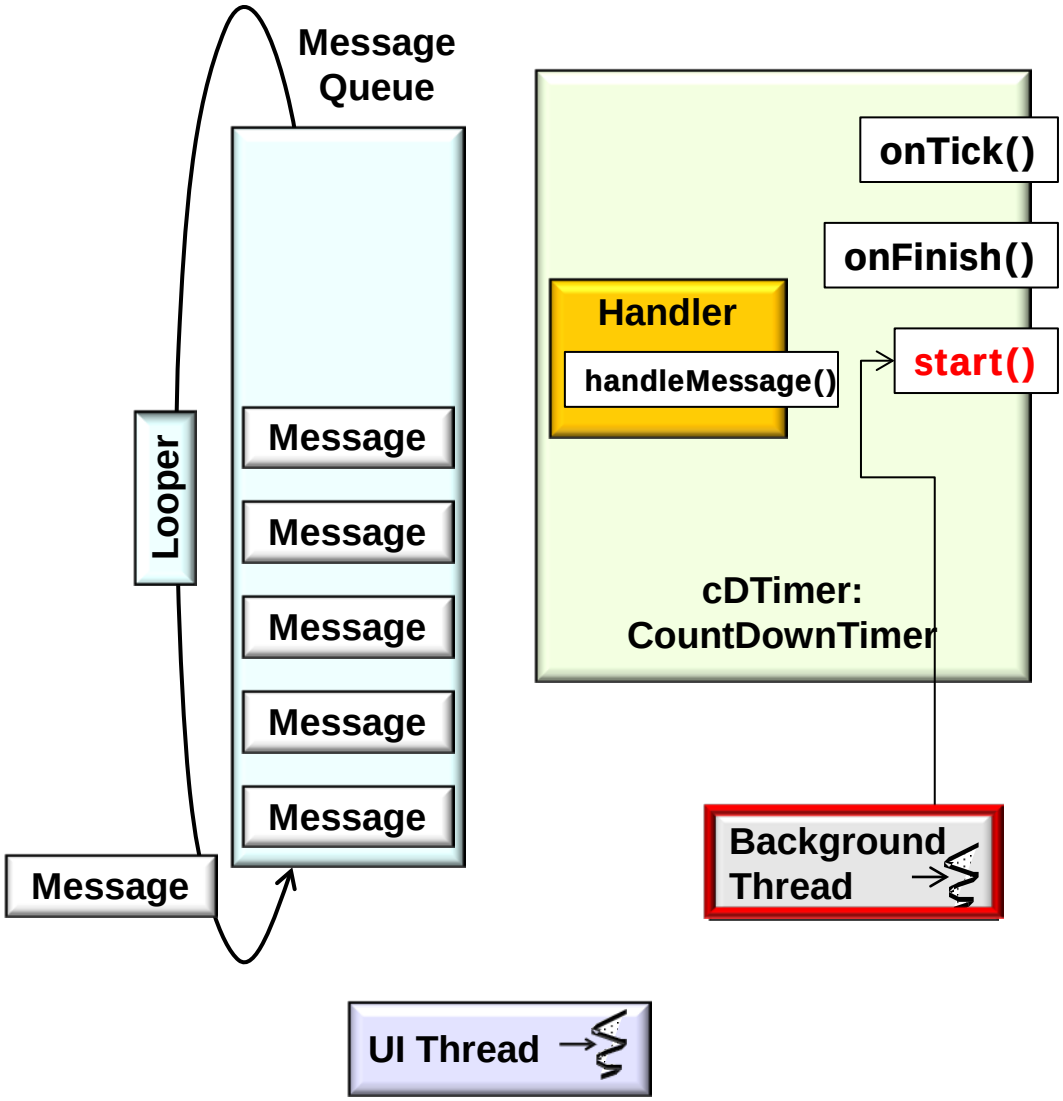
```
// UI Thread
CountDownTimer cDTimer =
  new CountDownTimer(30000,
                     1000) {
  public void onFinish()
  { mTextField.setText
      ("done!");
  }
};
```



Example of Sending & Handling Messages

- Use a Handler to send & process Messages in several Threads
- A CountDownTimer schedules a countdown until a future time

```
// Background Thread
cDTimer.start();
```



Sending & Handling Messages with the HaMeR Framework (Part 2)

Example of Sending & Handling Messages

- Use a Handler to send & process Messages in several Threads
- A CountDownTimer schedules a countdown until a future time
- CountDownTimer is an abstract class

```
public abstract class
                    CountDownTimer {
  ...
  public abstract void onTick
    (long millisUntilFinished);

  public abstract void onFinish();
  ...
```

Example of Sending & Handling Messages

- Use a Handler to send & process Messages in several Threads
- A CountDownTimer schedules a countdown until a future time
- CountDownTimer is an abstract class
  - Its hook methods must be overridden

```
public abstract class
                CountDownTimer {
  ...
  public abstract void onTick
    (long millisUntilFinished);

  public abstract void onFinish();
  ...
```

frameworks/base/core/java/android/os/CountDownTimer.java has the code

Example of Sending & Handling Messages

- Use a Handler to send & process Messages in several Threads
- A CountDownTimer schedules a countdown until a future time
- CountDownTimer is an abstract class
  - Its hook methods must be overridden

source.developer.android has all the Android source code

Example of Sending & Handling Messages

- Use a Handler to send & process Messages in several Threads
- A CountDownTimer schedules a countdown until a future time
- CountDownTimer is an abstract class
  - Its hook methods must be overridden
  - It's constructor stores the future time & count down interval

```
public abstract class
                CountDownTimer {
  ...
  public CountDownTimer
      (long millisInFuture,
       long countDownInterval) {
    mMillisInFuture =
      millisInFuture;
    mCountdownInterval =
      countDownInterval;
  }
  ...
```

Example of Sending & Handling Messages

- Use a Handler to send & process Messages in several Threads
- A CountDownTimer schedules a countdown until a future time
- CountDownTimer is an abstract class
  - Its hook methods must be overridden
  - It's constructor stores the future time & count down interval
  - Its Handler is associated with Thread Looper where it's created

```
public abstract class
                CountDownTimer {
  ...
  private Handler mHandler =
    new Handler() {
      ...
```

Example of Sending & Handling Messages

- Use a Handler to send & process Messages in several Threads
- A CountDownTimer schedules a countdown until a future time
- CountDownTimer is an abstract class
  - Its hook methods must be overridden
  - It's constructor stores the future time & count down interval
  - Its Handler is associated with Thread Looper where it's created

```
public abstract class
                    CountDownTimer {
  ...
  private Handler mHandler =
    new Handler() {
      public void handleMessage
        (Message msg) {
          ...
```

Subclasses must override this hook method to process messages

Example of Sending & Handling Messages

- Use a Handler to send & process Messages in several Threads
- A CountDownTimer schedules a countdown until a future time
- CountDownTimer is an abstract class
  - Its hook methods must be overridden
  - It's constructor stores the future time & count down interval
  - Its Handler is associated with Thread Looper where it's created

```
public abstract class
                    CountDownTimer {
  ...
  private Handler mHandler =
    new Handler() {
      public void handleMessage
        (Message msg) {
          ...
```

This hook method must be overridden to process messages

Example of Sending & Handling Messages

- Use a Handler to send & process Messages in several Threads
- A CountDownTimer schedules a countdown until a future time
- CountDownTimer is an abstract class
  - Its hook methods must be overridden
  - It's constructor stores the future time & count down interval
  - Its Handler is associated with Thread Looper where it's created

```
public abstract class
                CountDownTimer {
  ...
  private Handler mHandler =
    new Handler() {
      public void handleMessage
        (Message msg) {
          ...
          onTick(...);
          ...
          onFinished();
          ...
```

Example of Sending & Handling Messages

- Use a Handler to send & process Messages in several Threads
- A CountDownTimer schedules a countdown until a future time
- CountDownTimer is an abstract class
  - Its hook methods must be overridden
  - It's constructor stores the future time & count down interval
  - Its Handler is associated with Thread Looper where it's created

```
public abstract class
                CountDownTimer {
  ...
  public synchronized final
    CountDownTimer start() {
      ...
      mHandler.sendMessage
        (mHandler.
              obtainMessage(MSG));
      ...
```

Example of Sending & Handling Messages

- Use a Handler to send & process Messages in several Threads
- A CountDownTimer schedules a countdown until a future time
- CountDownTimer is an abstract class
  - Its hook methods must be overridden
  - It's constructor stores the future time & count down interval
  - Its Handler is associated with Thread Looper where it's created

```
public abstract class
                CountDownTimer {
  ...
  public synchronized final
    CountDownTimer start() {
      ...
      mHandler.sendMessage
        (mHandler.
              obtainMessage(MSG));
      ...
```

Example of Sending & Handling Messages

- Use a Handler to send & process Messages in several Threads
- A CountDownTimer schedules a countdown until a future time
- CountDownTimer is an abstract class
  - Its hook methods must be overridden
  - It's constructor stores the future time & count down interval
  - Its Handler is associated with Thread Looper where it's created

```
public abstract class
                CountDownTimer {
  ...
  public synchronized final
    CountDownTimer start() {
      ...
      mHandler.sendMessage
        (mHandler.
              obtainMessage(MSG));
      ...
```

Example of Sending & Handling Messages

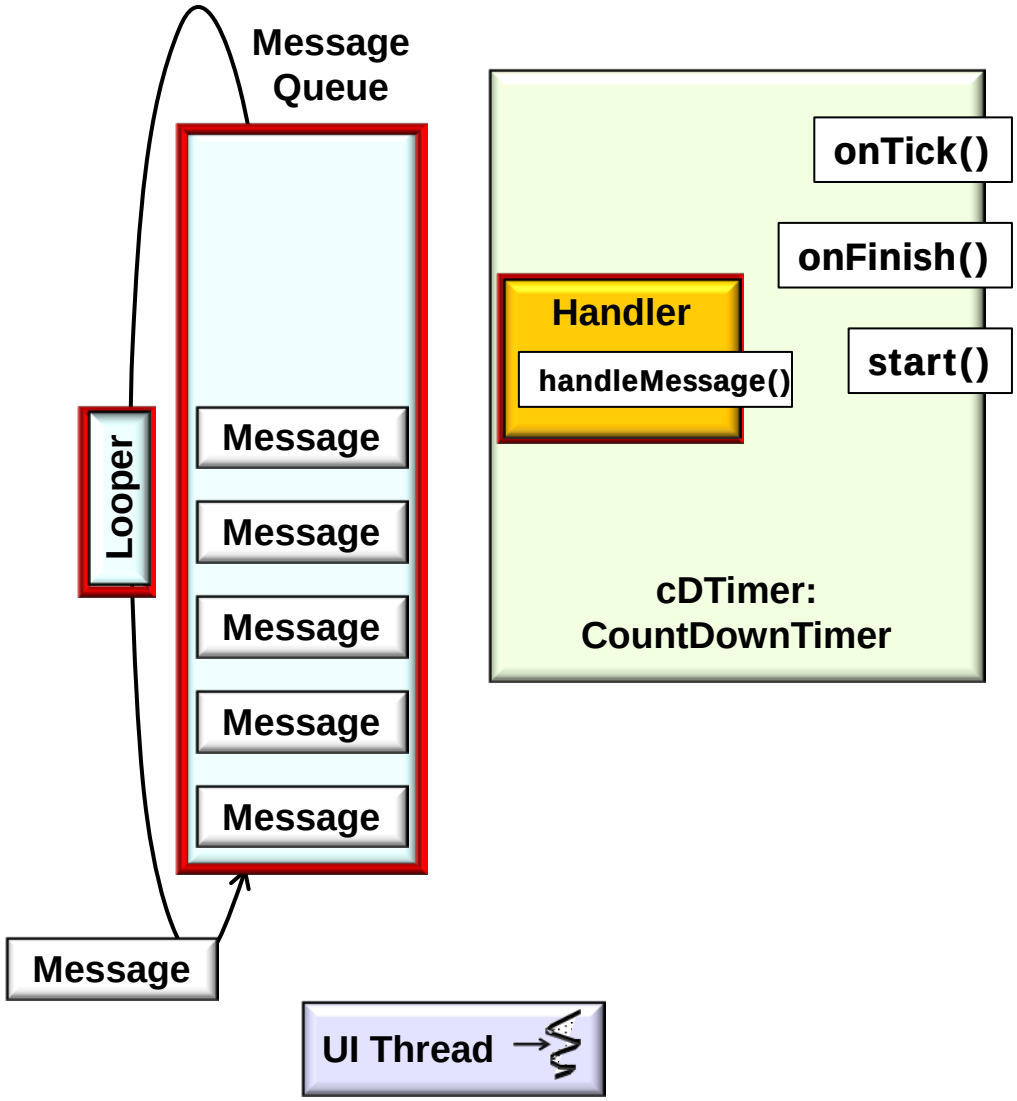
- Use a Handler to send & process Messages in several Threads
- A CountDownTimer schedules a countdown until a future time
- CountDownTimer is an abstract class
  - Its hook methods must be overridden
  - It's constructor stores the future time & count down interval
  - Its Handler is associated with Thread Looper where it's created

```
public class Handler {
  ...
  public final boolean
      sendMessage(Message msg) {
    return sendMessageDelayed
      (msg, 0);
  }
```

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

Example of Sending & Handling Messages

- Use a Handler to send & process Messages in several Threads
- A CountDownTimer schedules a countdown until a future time
- CountDownTimer is an abstract class
  - Its hook methods must be overridden
  - It's constructor stores the future time & count down interval
  - Its Handler is associated with Thread Looper where it's created



See previous part on "Posting & Processing Runnables with Android Handler"

Example of Sending & Handling Messages

- Use a Handler to send & process Messages in several Threads
- A CountDownTimer schedules a countdown until a future time
- CountDownTimer is an abstract class
  - Its hook methods must be overridden
  - It's constructor stores the future time & count down interval
  - Its Handler is associated with Thread Looper where it's created

```
public class Handler {
  ...
  public void dispatchMessage
                (Message msg) {
    if (msg.callback != null) {
       handleCallback(msg);
    } else {
      if (mCallback != null) {
        if (mCallback.
            handleMessage(msg))
          return;
      }
      handleMessage(msg);
    }
```

Example of Sending & Handling Messages

- Use a Handler to send & process Messages in several Threads
- A CountDownTimer schedules a countdown until a future time
- CountDownTimer is an abstract class
  - Its hook methods must be overridden
  - It's constructor stores the future time & count down interval
  - Its Handler is associated with Thread Looper where it's created

```
public class Handler {
  ...
  public void dispatchMessage
                (Message msg) {
    if (msg.callback != null) {
       handleCallback(msg);
    } else {
      if (mCallback != null) {
        if (mCallback.
            handleMessage(msg))
          return;
      }
      handleMessage(msg);
    }
```

The handleMessage() hook method runs in the Handler's Thread

Example of Sending & Handling Messages

- Use a Handler to send & process Messages in several Threads
- A CountDownTimer schedules a countdown until a future time
- CountDownTimer is an abstract class
  - Its hook methods must be overridden
  - It's constructor stores the future time & count down interval
  - Its Handler is associated with Thread Looper where it's created
  - The handleMessage() hook method implements the count down timer callback logic

```
public abstract class
                    CountDownTimer {
  ...
  private Handler mHandler =
    new Handler() {
      public void handleMessage
                    (Message msg) {
        synchronized (this) {
          long millisLeft = ...

          if (millisLeft <= 0)
            onFinish();
          ...
```

Example of Sending & Handling Messages

- Use a Handler to send & process Messages in several Threads
- A CountDownTimer schedules a countdown until a future time
- CountDownTimer is an abstract class
  - Its hook methods must be overridden
  - It's constructor stores the future time & count down interval
  - Its Handler is associated with Thread Looper where it's created
  - The handleMessage() hook method implements the count down timer callback logic

```
public abstract class
                    CountDownTimer {
  ...
  private Handler mHandler =
    new Handler() {
      public void handleMessage
                    (Message msg) {
        synchronized (this) {
          long millisLeft = ...

          if (millisLeft <= 0)
            onFinish();
          ...
```

Example of Sending & Handling Messages

- Use a Handler to send & process Messages in several Threads
- A CountDownTimer schedules a countdown until a future time
- CountDownTimer is an abstract class
  - Its hook methods must be overridden
  - It's constructor stores the future time & count down interval
  - Its Handler is associated with Thread Looper where it's created
  - The handleMessage() hook method implements the count down timer callback logic

```
public abstract class
                    CountDownTimer {
  ...
  private Handler mHandler =
    new Handler() {
      public void handleMessage
                    (Message msg) {
        synchronized (this) {
          long millisLeft = ...
          ...
          else if (millisLeft <
             mCountdownInterval)
            sendMessageDelayed(
              obtainMessage(MSG),
                     millisLeft);
          ...
```

Example of Sending & Handling Messages

- Use a Handler to send & process Messages in several Threads
- A CountDownTimer schedules a countdown until a future time
- CountDownTimer is an abstract class
  - Its hook methods must be overridden
  - It's constructor stores the future time & count down interval
  - Its Handler is associated with Thread Looper where it's created
  - The handleMessage() hook method implements the count down timer callback logic

```
public abstract class
                    CountDownTimer {
  ...
  private Handler mHandler =
    new Handler() {
      public void handleMessage
                    (Message msg) {
        synchronized (this) {
          long millisLeft = ...
          ...
          else {
            onTick(millisLeft);
            ...
            sendMessageDelayed
              (obtainMessage(MSG),
                          delay);
          ...
```

Example of Sending & Handling Messages

- Use a Handler to send & process Messages in several Threads
- A CountDownTimer schedules a countdown until a future time
- CountDownTimer is an abstract class
  - Its hook methods must be overridden
  - It's constructor stores the future time & count down interval
  - Its Handler is associated with Thread Looper where it's created
  - The handleMessage() hook method implements the count down timer callback logic

```
public abstract class
                    CountDownTimer {
  ...
  private Handler mHandler =
    new Handler() {
      public void handleMessage
                    (Message msg) {
        synchronized (this) {
          long millisLeft = ...
          ...
          else {
            onTick(millisLeft);
            ...
            sendMessageDelayed
              (obtainMessage(MSG),
                          delay);
          ...
```

Example of Sending & Handling Messages

- Use a Handler to send & process Messages in several Threads
- A CountDownTimer schedules a countdown until a future time
- CountDownTimer is an abstract class
  - Its hook methods must be overridden
  - It's constructor stores the future time & count down interval
  - Its Handler is associated with Thread Looper where it's created
  - The handleMessage() hook method implements the count down timer callback logic

```
public abstract class
                    CountDownTimer {
  ...
  private Handler mHandler =
    new Handler() {
      public void handleMessage
                    (Message msg) {
        synchronized (this) {
          long millisLeft = ...
          ...
          else {
            onTick(millisLeft);
            ...
            sendMessageDelayed
              (obtainMessage(MSG),
                          delay);
          ...
```

Example of Sending & Handling Messages

- Use a Handler to send & process Messages in several Threads
- A CountDownTimer schedules a countdown until a future time
- CountDownTimer is an abstract class

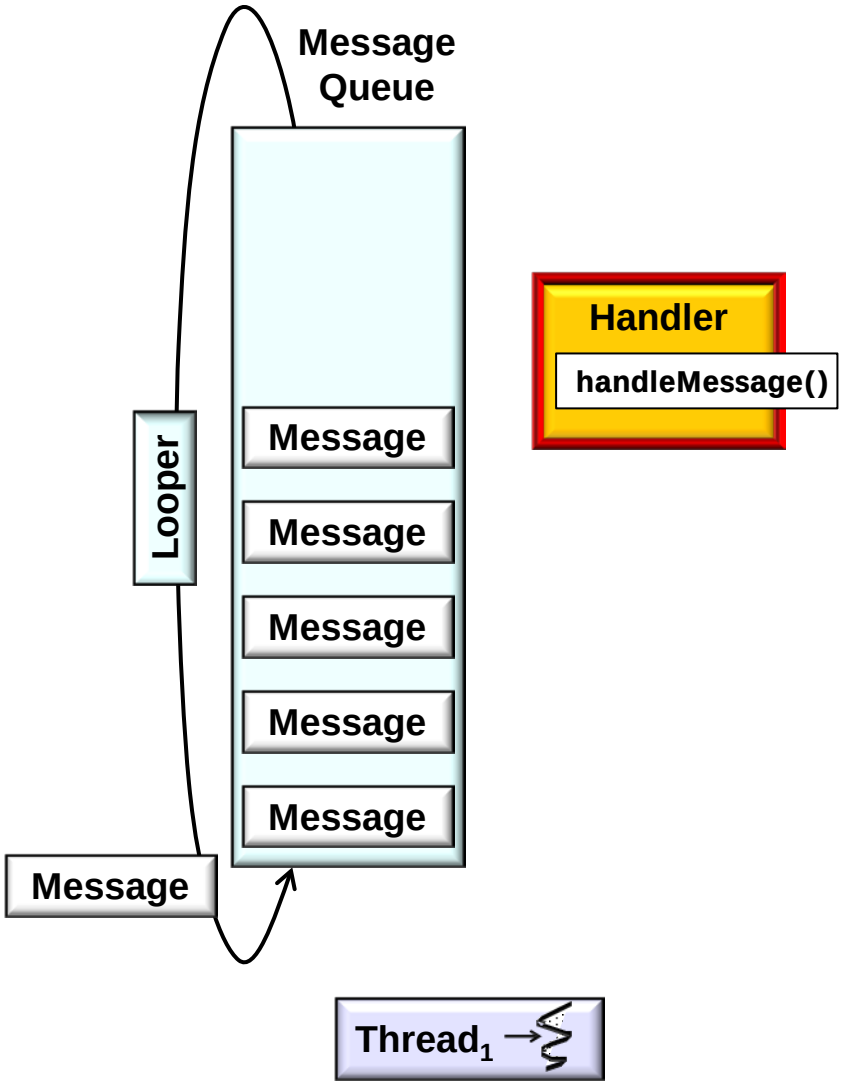
```
packages/apps/Phone/src/com/android/phone/
          EmergencyCallbackModeExitDialog.java
packages/apps/Phone/src/com/android/phone/
          EmergencyCallbackModeService.java
packages/apps/Settings/src/com/android/settings/
          ConfirmLockPattern.java
frameworks/base/policy/src/com/android/internal/policy/impl/
          PatternUnlockScreen.java
Frameworks/base/policy/src/com/android/internal/policy/impl/
          PasswordUnlockScreen.jav
```

CountDownTimer is used throughout Android

Summary

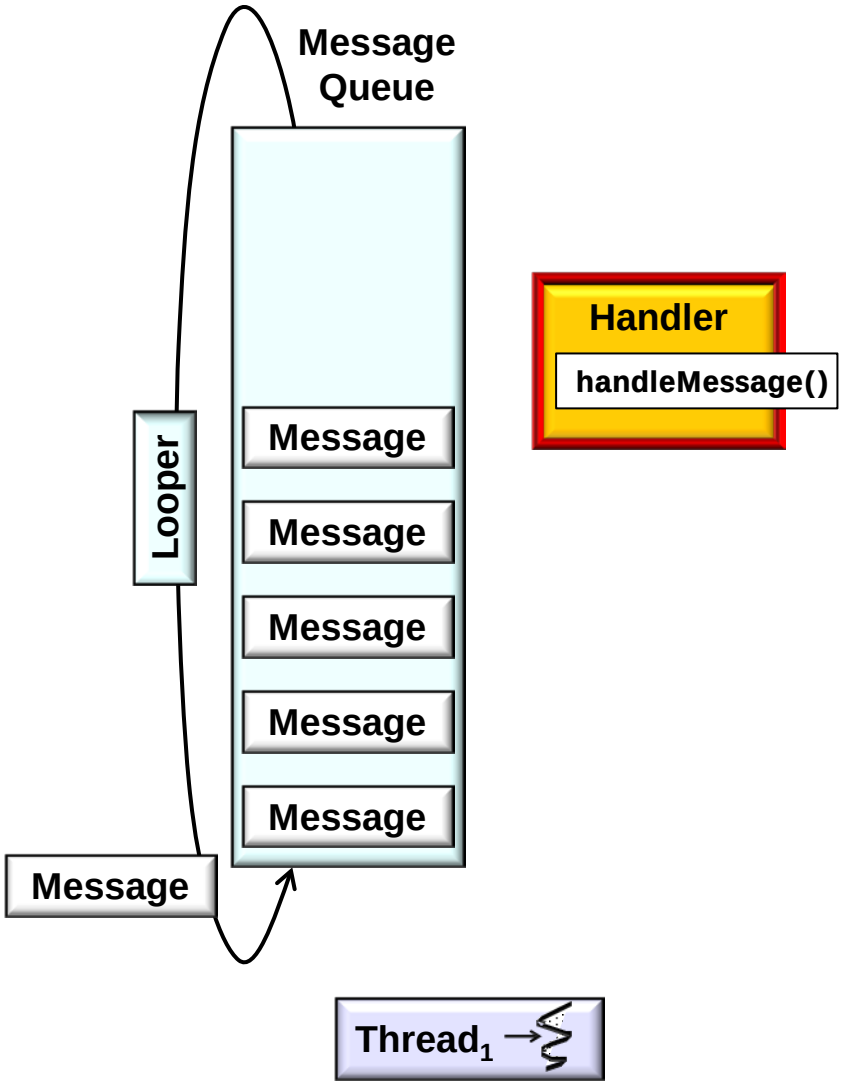
Summary

- Handler's sendMessage() methods form a key portion of the Android HaMeR framework



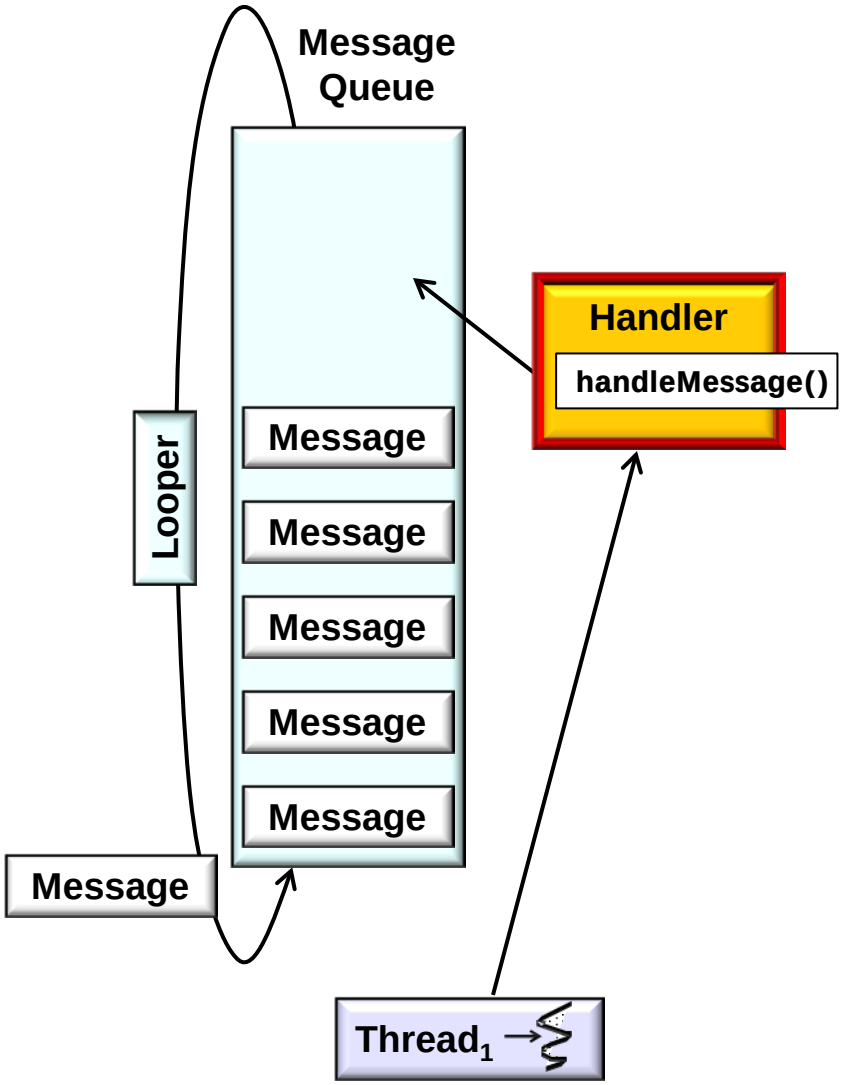
Summary

- Handler's sendMessage() methods form a key portion of the Android HaMeR framework
  - They can enqueue & later handle Messages posted from



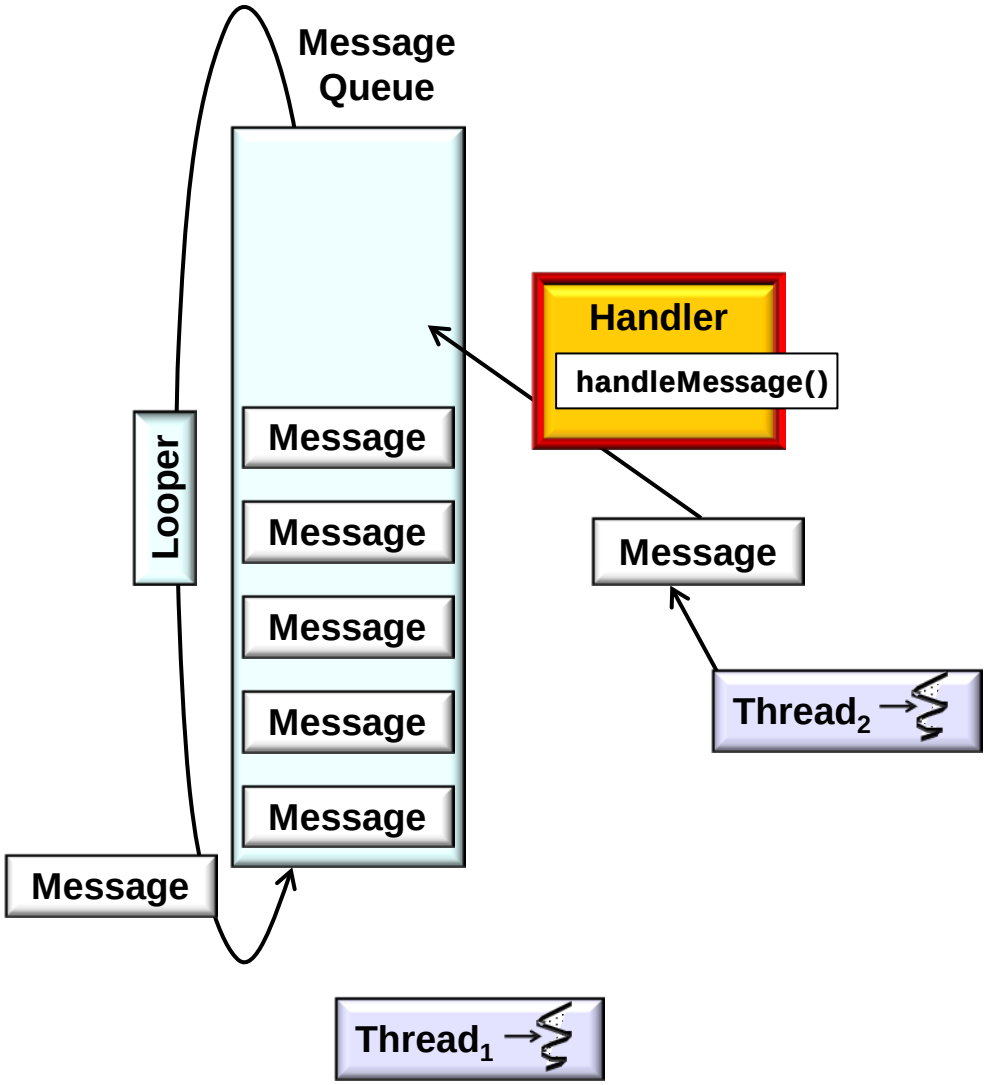
Summary

- Handler's sendMessage() methods form a key portion of the Android HaMeR framework
  - They can enqueue & later handle Messages posted from
    - within a single Thread to itself or



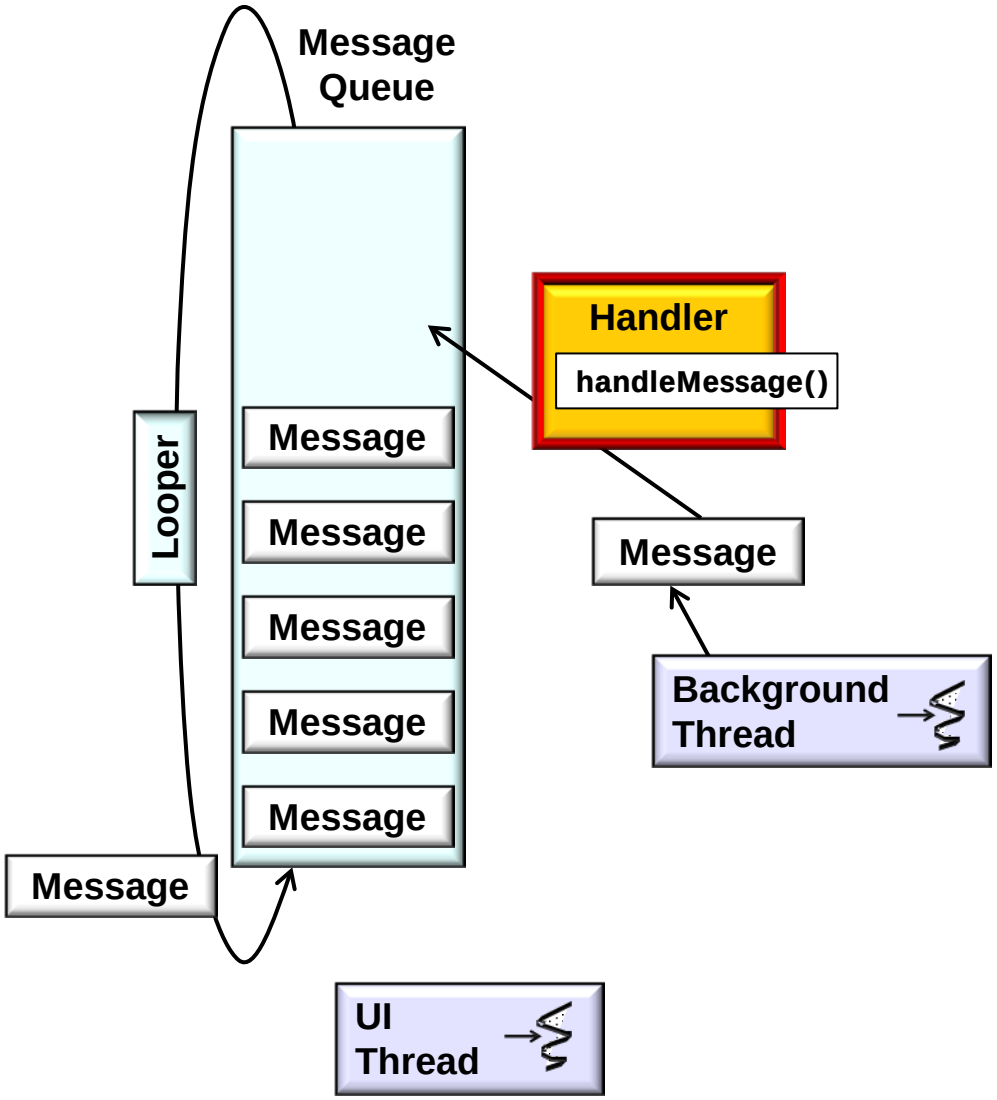
Summary

- Handler's sendMessage() methods form a key portion of the Android HaMeR framework
  - They can enqueue & later handle Messages posted from
    - within a single Thread to itself or
    - one Thread to another



Summary

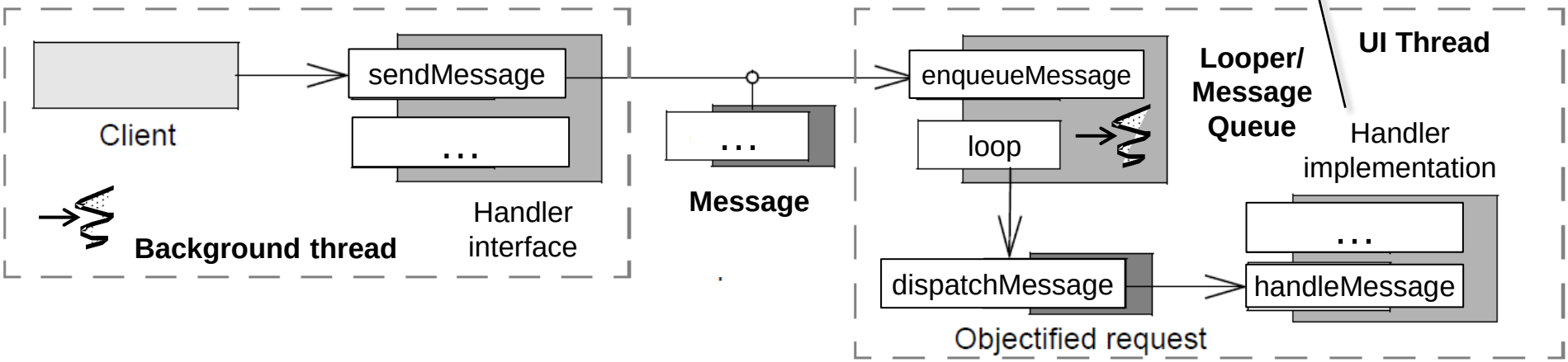
- Handler's sendMessage() methods form a key portion of the Android HaMeR framework
  - They can enqueue & later handle Messages posted from
    - within a single Thread to itself or
    - one Thread to another
  - They are often used to send Messages from background Threads to the UI Thread



Summary

- Handler's sendMessage() methods form a key portion of the Android HaMeR framework
- These methods implement a message passing variant of the *Active Object* pattern

```
private Handler mHandler =
  new Handler() {
    public void handleMessage
      (Message msg) {
        ...
        onTick(...);
        ...
        onFinished();
        ...
```

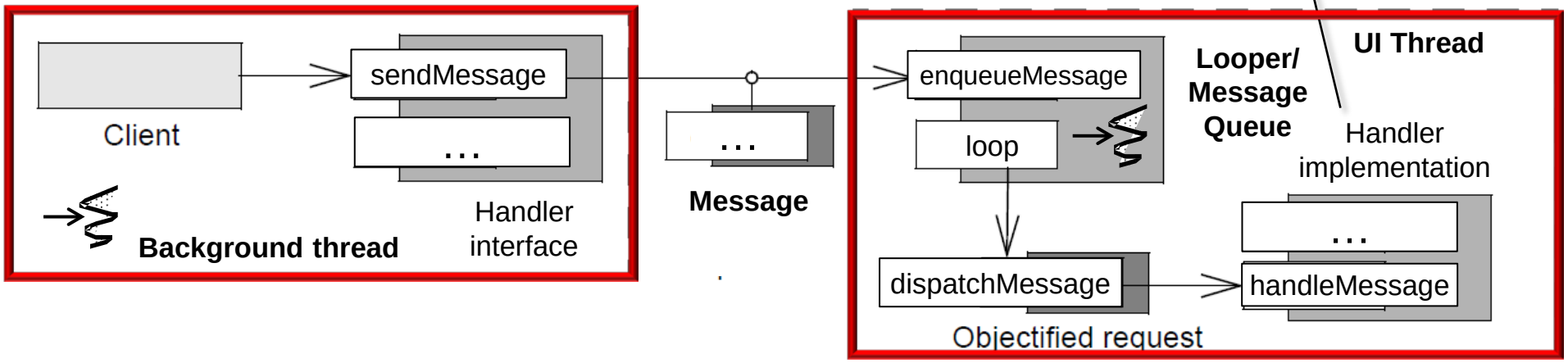


See upcoming parts on "the *Active Object* pattern"

Summary

- Handler's sendMessage() methods form a key portion of the Android HaMeR framework
- These methods implement a message passing variant of the *Active Object* pattern
- Messages decouple client senders from Handler receivers

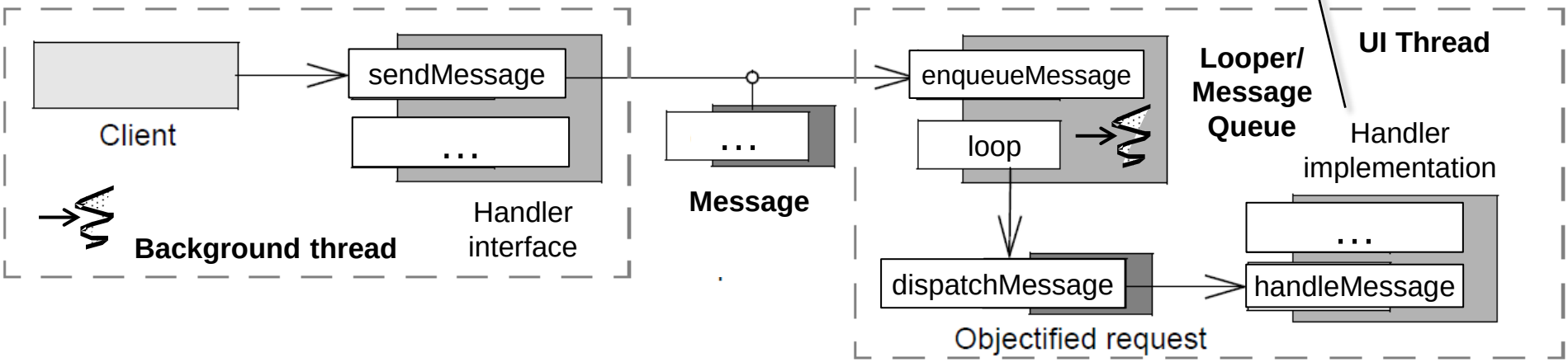
```
private Handler mHandler =
  new Handler() {
    public void handleMessage
      (Message msg) {
        ...
        onTick(...);
        ...
        onFinished();
        ...
```



Summary

- Handler's sendMessage() methods form a key portion of the Android HaMeR framework
- These methods implement a message passing variant of the *Active Object* pattern
- Messages decouple client senders from Handler receivers

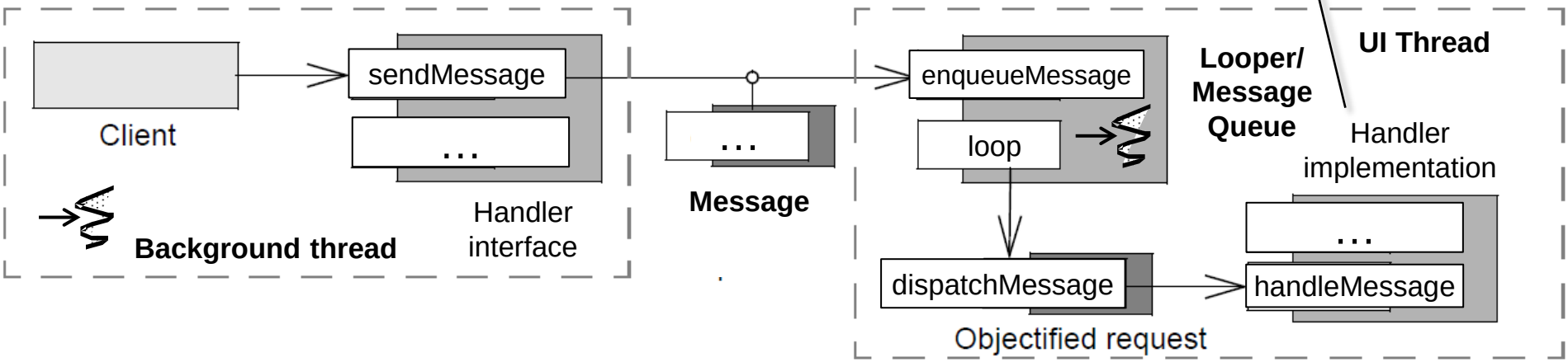
```
private Handler mHandler =
  new Handler() {
    public void handleMessage
      (Message msg) {
        ...
        onTick(...);
        ...
        onFinished();
        ...
```



Summary

- Handler's sendMessage() methods form a key portion of the Android HaMeR framework
- These methods implement a message passing variant of the *Active Object* pattern
- Messages decouple client senders from Handler receivers

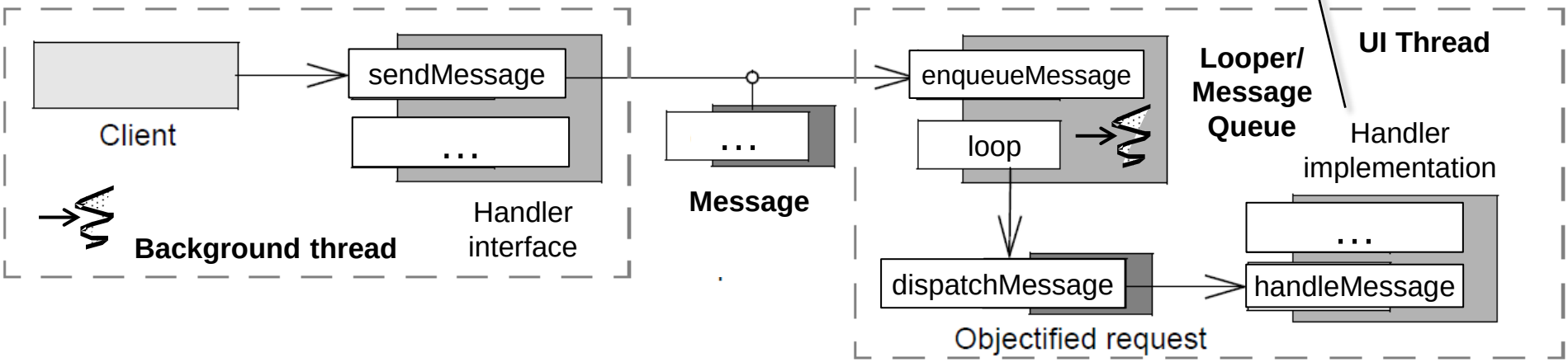
```
private Handler mHandler =
  new Handler() {
    public void handleMessage
      (Message msg) {
        ...
        onTick(...);
        ...
        onFinished();
        ...
```



Summary

- Handler's sendMessage() methods form a key portion of the Android HaMeR framework
- These methods implement a message passing variant of the *Active Object* pattern
- Messages decouple client senders from Handler receivers

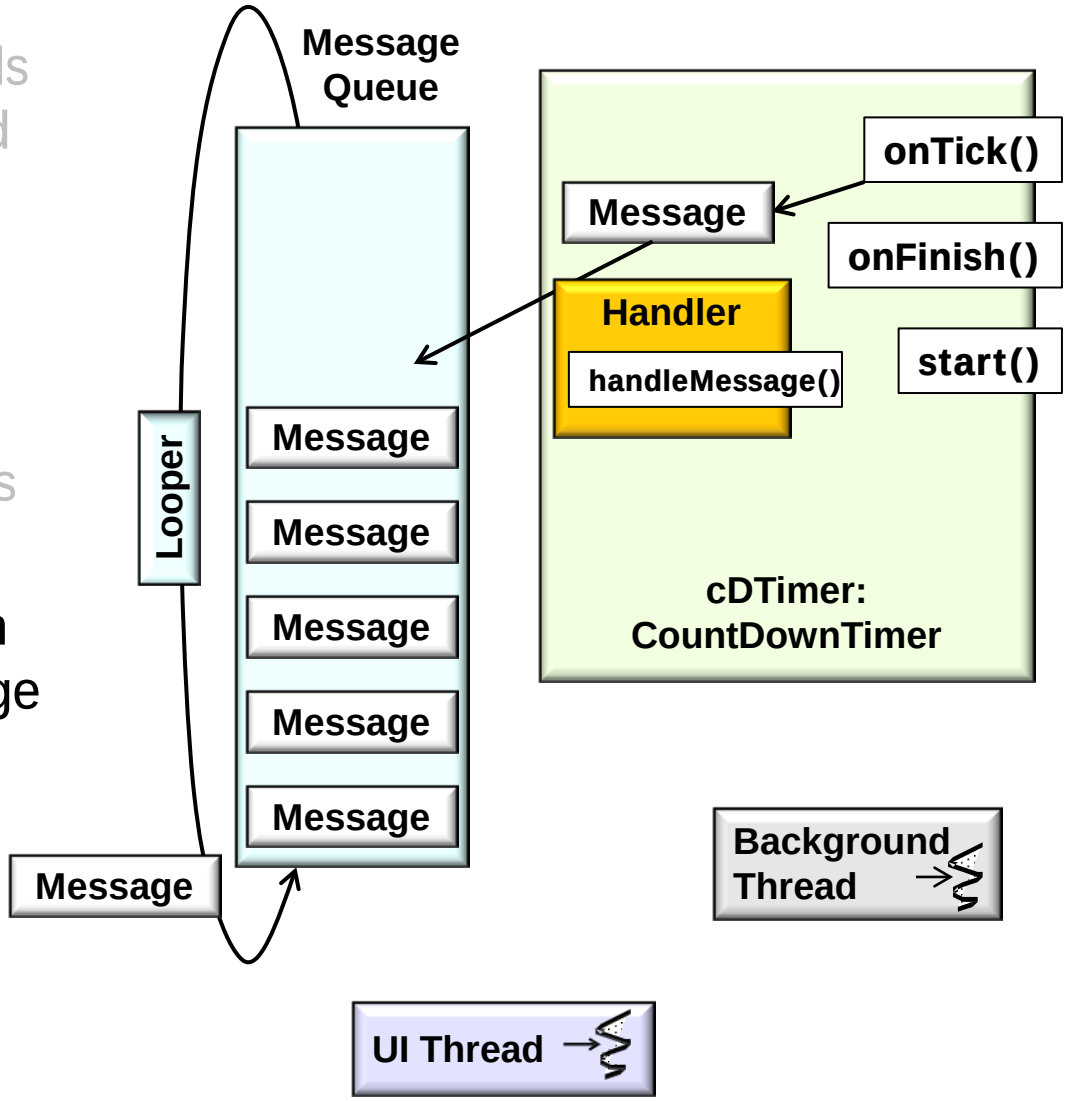
```
private Handler mHandler =
  new Handler() {
    public void handleMessage
      (Message msg) {
        ...
        onTick(...);
        ...
        onFinished();
        ...
```



There's a trade-off between flexibility & simplicity

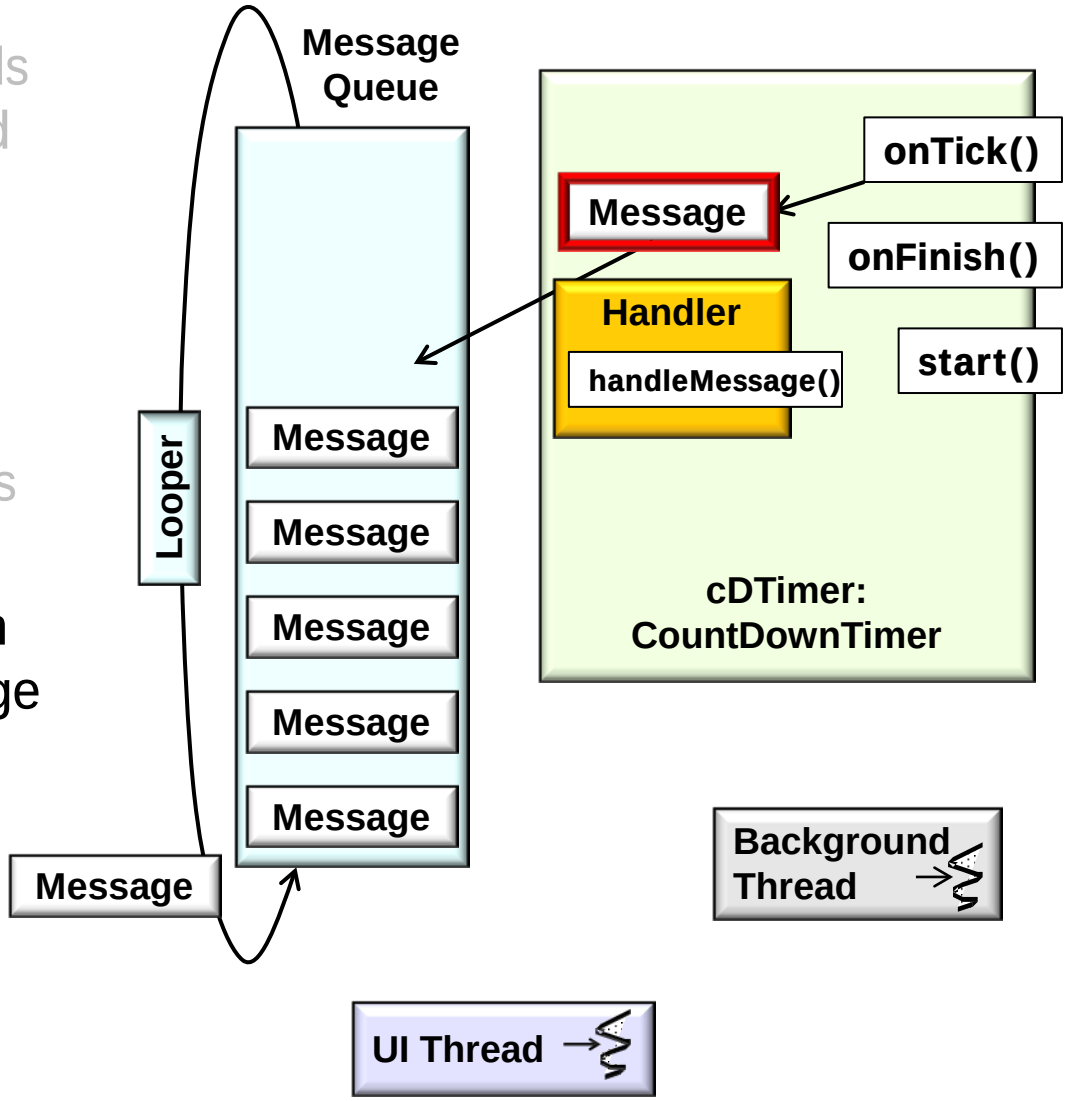
Summary

- Handler's sendMessage() methods form a key portion of the Android HaMeR framework
- These methods implement a message passing variant of the *Active Object* pattern
- Messages decouple client senders from Handler receivers
- CountDownTimer was used as an example to showcase the message passing features of Handler



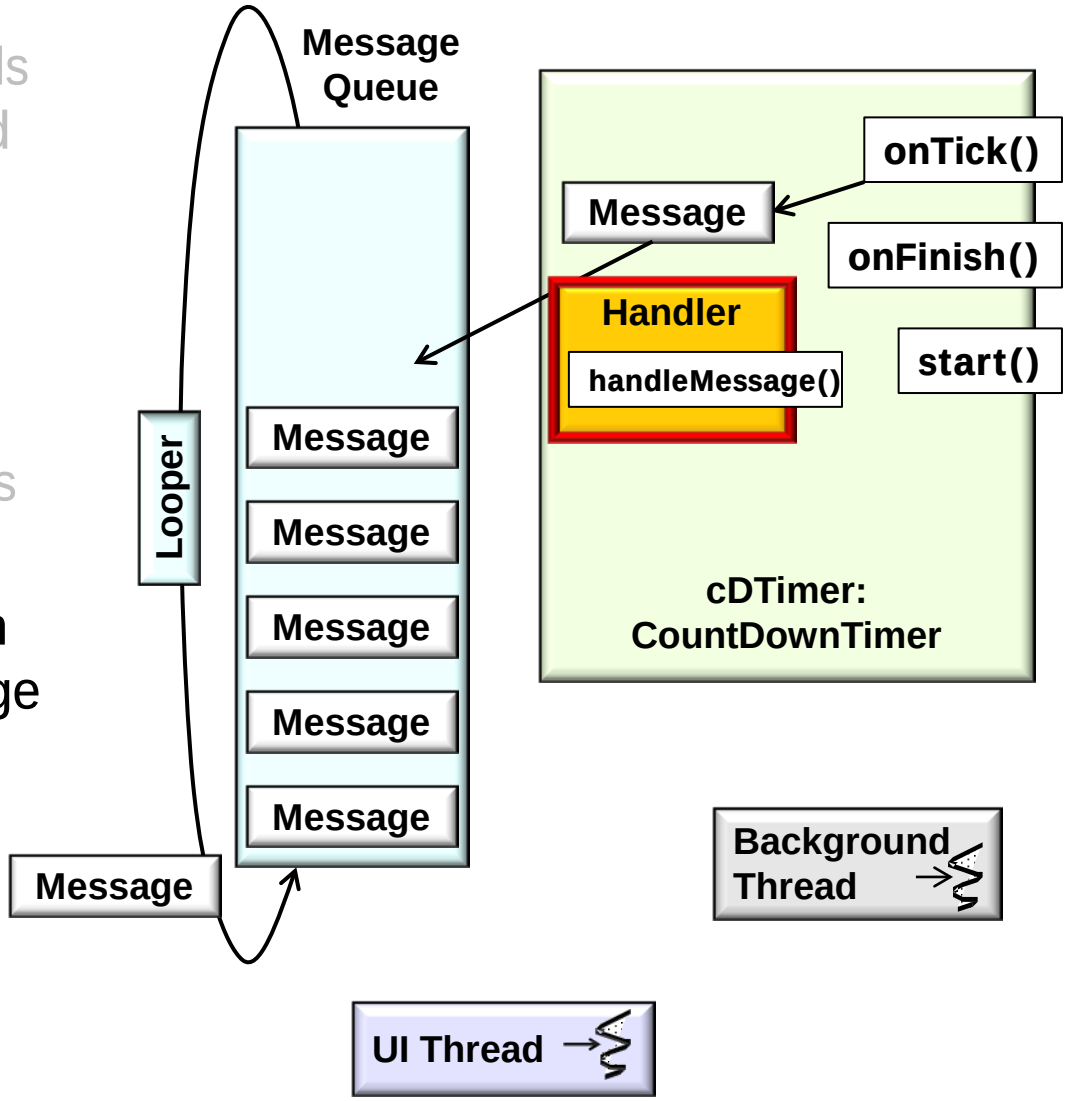
Summary

- Handler's sendMessage() methods form a key portion of the Android HaMeR framework
- These methods implement a message passing variant of the *Active Object* pattern
- Messages decouple client senders from Handler receivers
- CountDownTimer was used as an example to showcase the message passing features of Handler



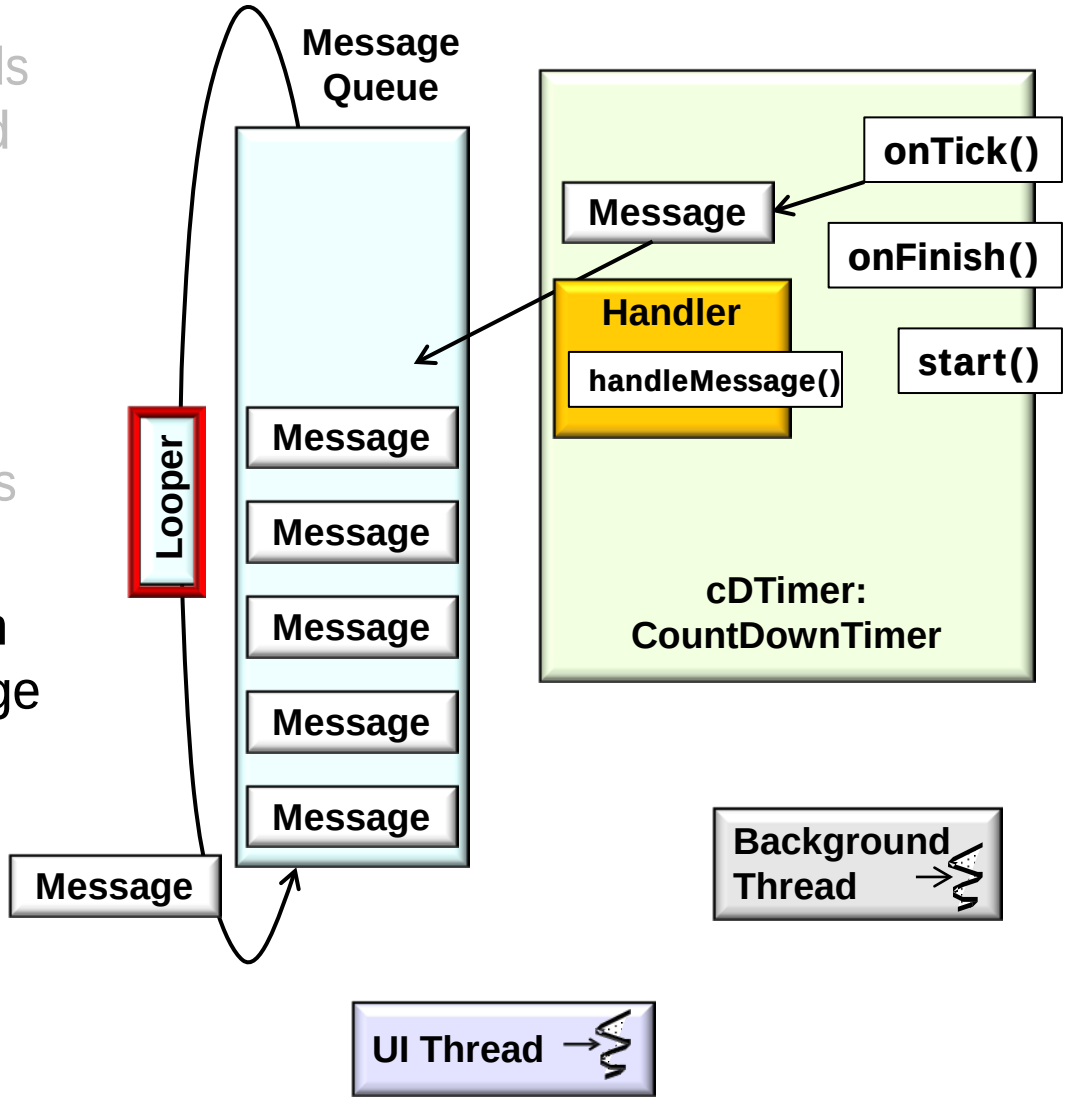
Summary

- Handler's sendMessage() methods form a key portion of the Android HaMeR framework
- These methods implement a message passing variant of the *Active Object* pattern
- Messages decouple client senders from Handler receivers
- CountDownTimer was used as an example to showcase the message passing features of Handler



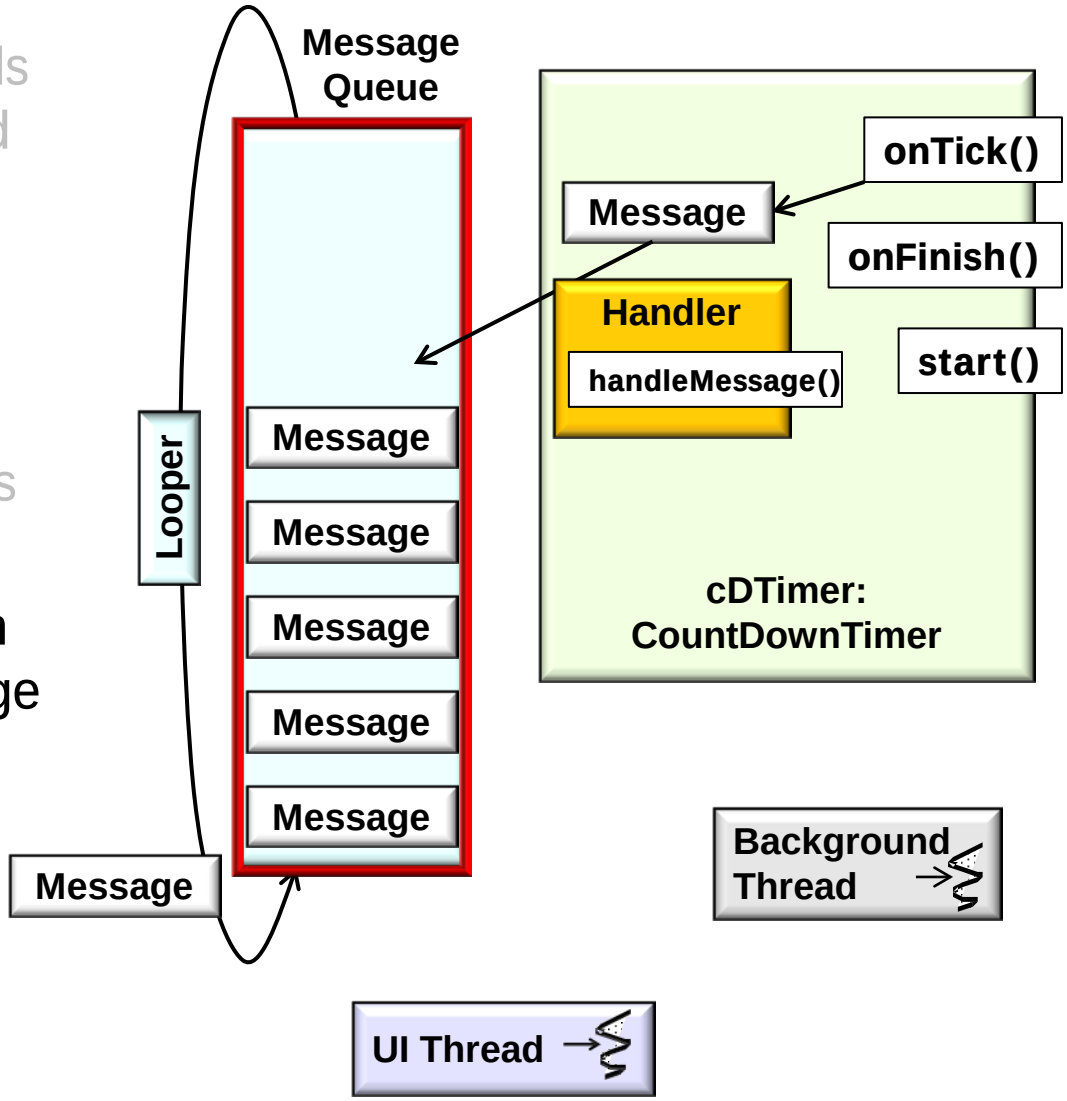
Summary

- Handler's sendMessage() methods form a key portion of the Android HaMeR framework
- These methods implement a message passing variant of the *Active Object* pattern
- Messages decouple client senders from Handler receivers
- CountDownTimer was used as an example to showcase the message passing features of Handler



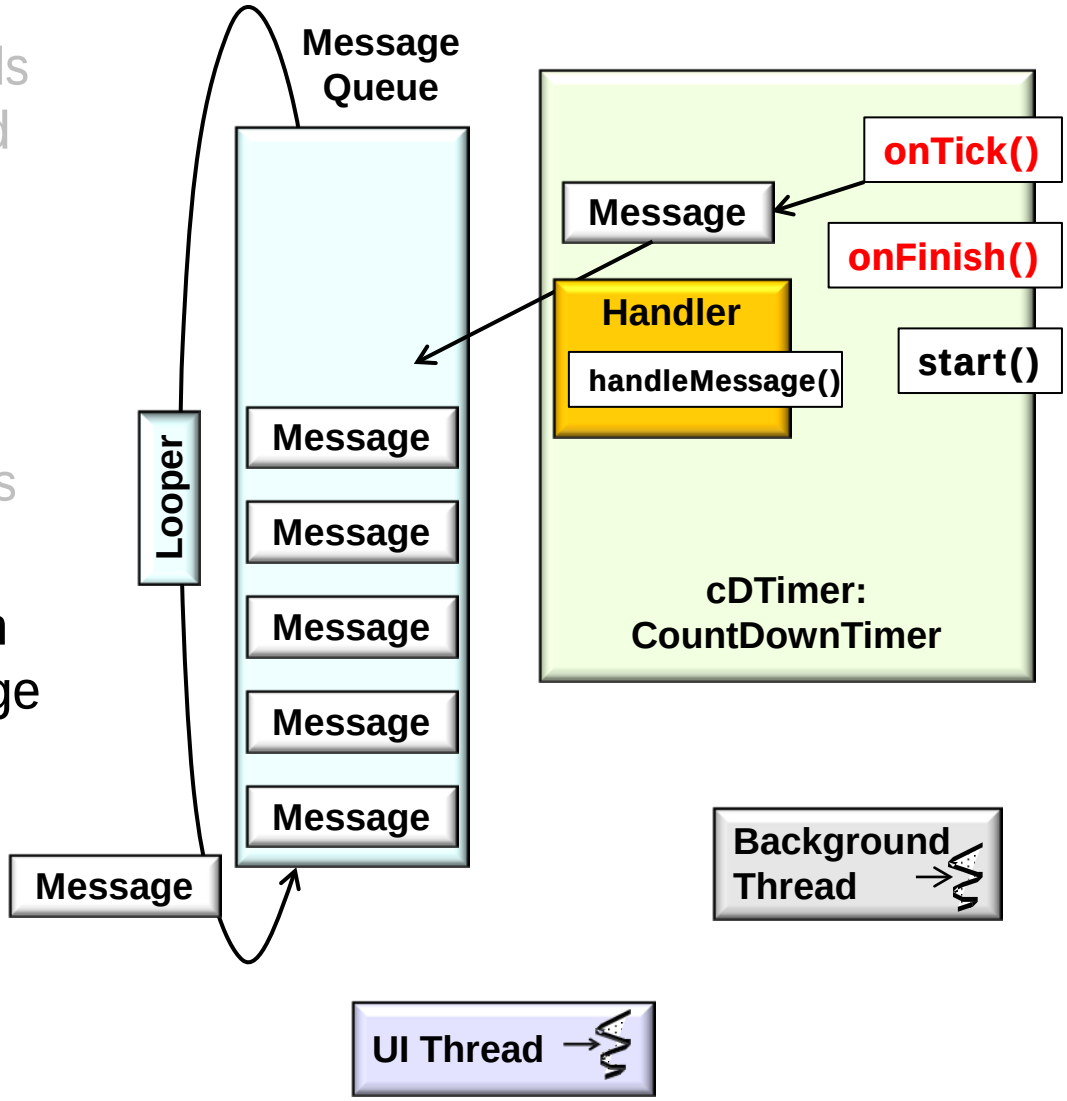
Summary

- Handler's sendMessage() methods form a key portion of the Android HaMeR framework
- These methods implement a message passing variant of the *Active Object* pattern
- Messages decouple client senders from Handler receivers
- CountDownTimer was used as an example to showcase the message passing features of Handler



Summary

- Handler's sendMessage() methods form a key portion of the Android HaMeR framework
- These methods implement a message passing variant of the *Active Object* pattern
- Messages decouple client senders from Handler receivers
- CountDownTimer was used as an example to showcase the message passing features of Handler



Summary

- Handler's sendMessage() methods form a key portion of the Android HaMeR framework
- These methods implement a message passing variant of the *Active Object* pattern
- Messages decouple client senders from Handler receivers
- CountDownTimer was used as an example to showcase the message passing features of Handler