CMSC436: Programming Handheld Systems

Fall 2017

Threads, AsyncTasks & Handlers

Today’s Topics

Threading overview

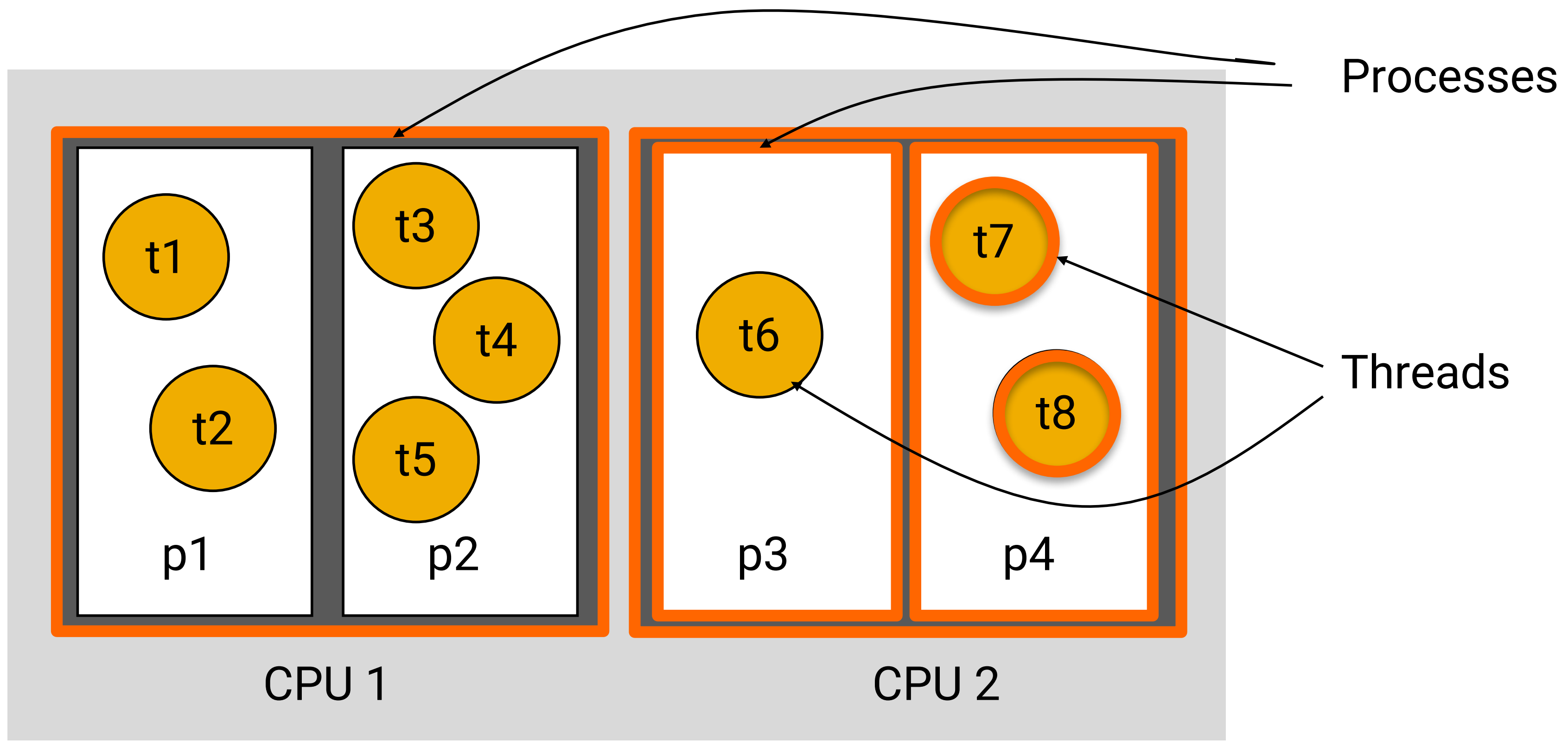
Android’s UI Thread

The AsyncTask class

The Handler class

What is a Thread?

- Conceptual view
  - Parallel computation running in a process
- Implementation view
  - A program counter and a stack
  - Heap and static areas shared with other threads



Computing Device

Java Threads

Represented by an Object of type Java.lang.Thread

Threads implement the Runnable interface

    public void run()

See:

http://docs.oracle.com/javase/tutorial/essential/concurrency/threads.html

Some Thread Methods

void start()

Starts the Thread

void sleep(long time)

Sleeps for the given period

Some Object Methods

void wait()

Current thread waits until another thread invokes notify() on this object

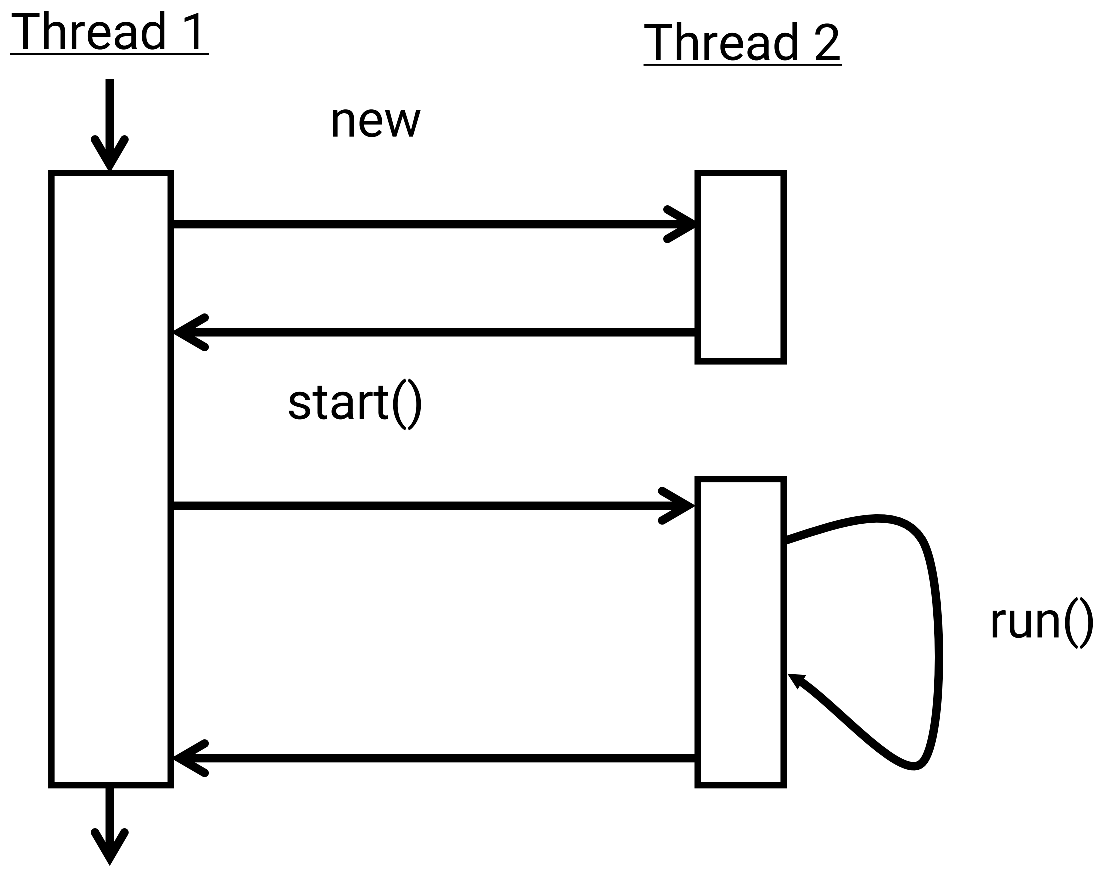
void notify()

Wakes up a single thread that is waiting on this object

Basic Thread Use Case

- Instantiate a Thread object
- Invoke the Thread’s start() method
  - Thread’s run() method get called
- Thread terminates when run() returns

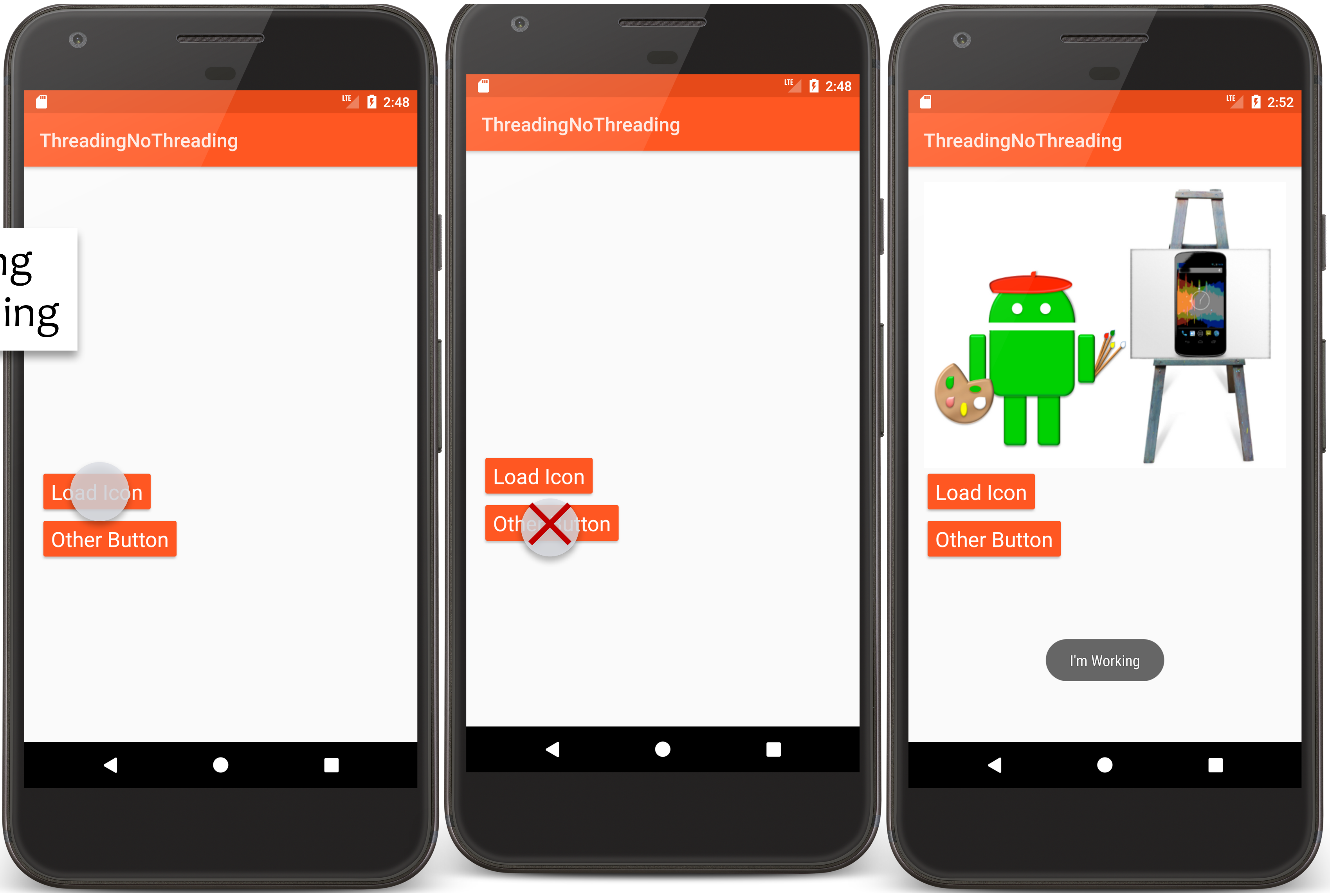
Basic Thread Use Case



ThreadingNoThreading

- Application displays two buttons
- LoadIcon
  - Load a bitmap from a resource file & display
  - Show loaded bitmap
- Other Button
  - Display some text

Threading NoThreading



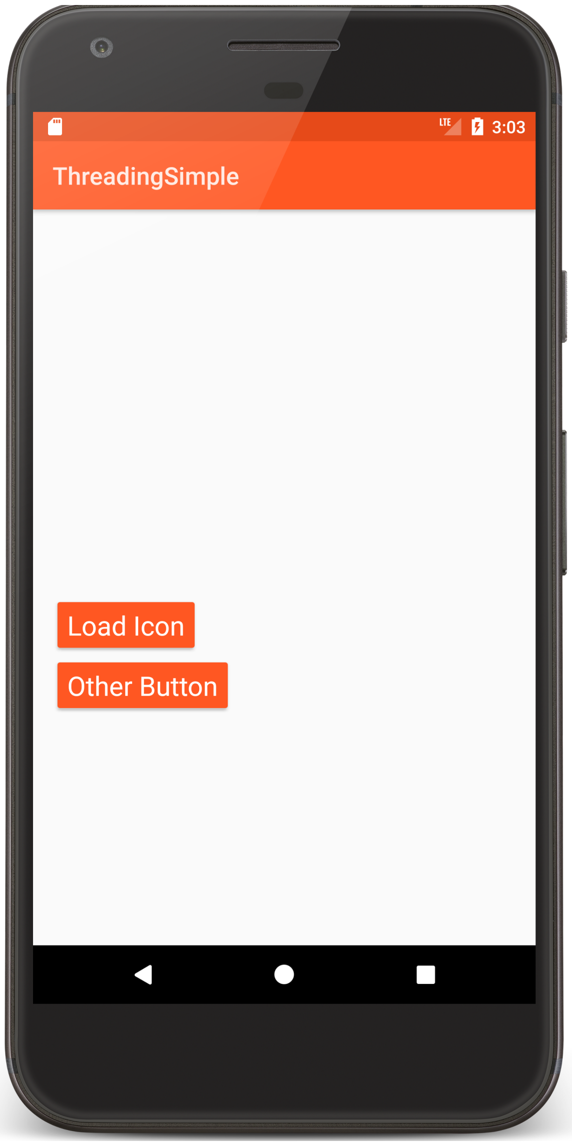
```
public void onClickOtherButton(View v) {
   Toast.makeText(NoThreadingExample.this, "I'm Working",
         Toast.LENGTH_SHORT).show();
}


public void onClickLoadButton(View view) {
   try {
      // Accentuates slow operation
      Thread.sleep(5000);
   } catch (InterruptedException e) { e.printStackTrace(); }
   mIView.setImageBitmap(BitmapFactory.decodeResource(
                                                        getResources(), R.drawable.painter));
}
```

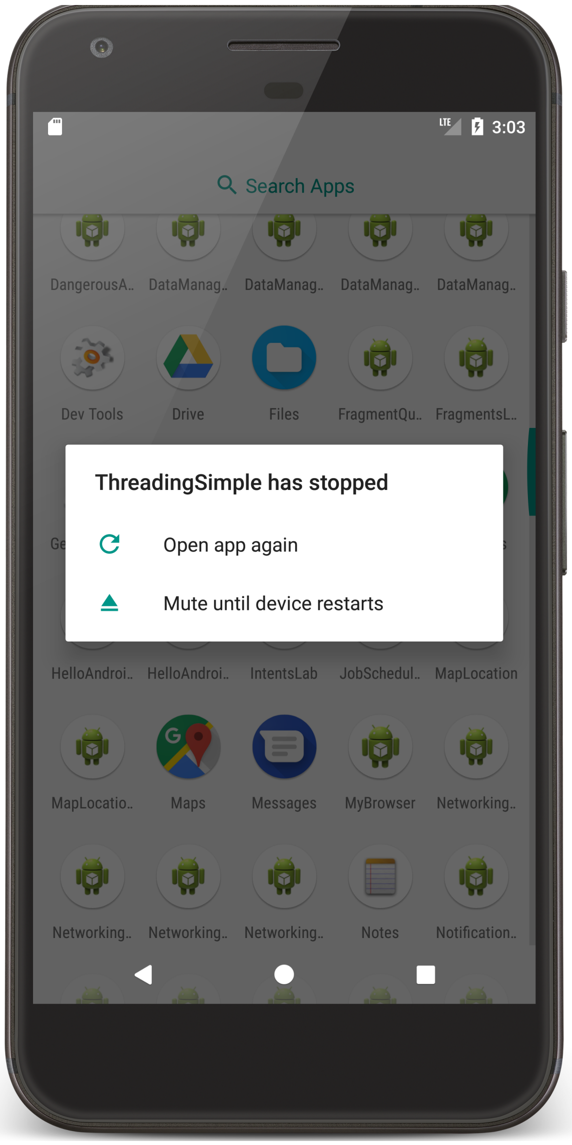
ThreadingSimple

Seemingly obvious, but incorrect, solution:

Button listener spawns a separate thread to load bitmap & display it

Threading Simple

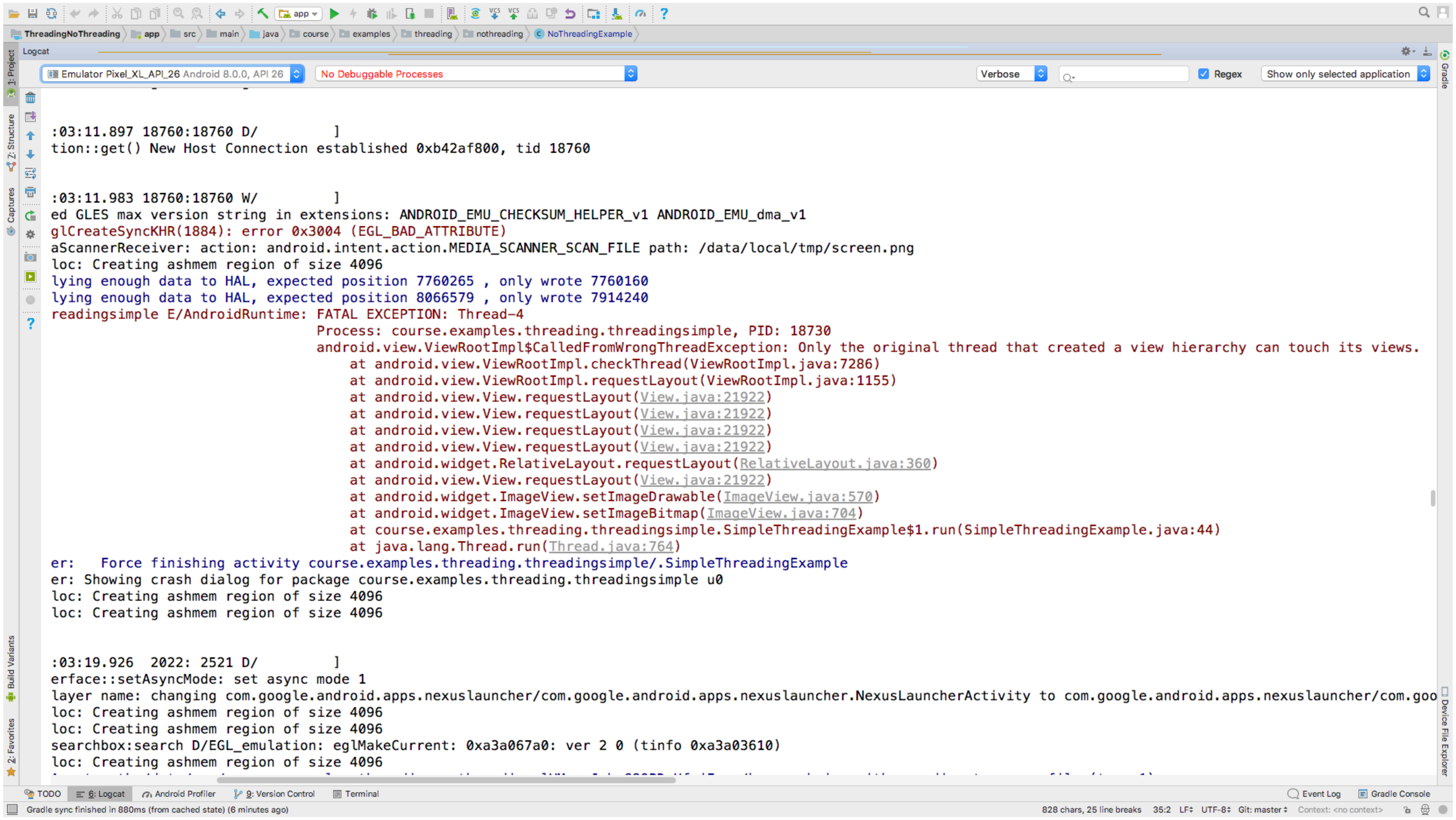




ThreadingSimple has stopped

Open app again

Mute until device restarts



```
:03:11.897 18760:18760 D/         ]
tion::get() New Host Connection established 0xb42af800, tid 18760


:03:11.983 18760:18760 W/         ]
ed GLES max version string in extensions: ANDROID_EMU_CHECKSUM_HELPER_v1 ANDROID_EMU_dma_v1
glCreateSyncKHR(1884): error 0x3004 (EGL_BAD_ATTRIBUTE)
aScannerReceiver: action: android.intent.action.MEDIA_SCANNER_SCAN_FILE path: /data/local/tmp/screen.png
loc: Creating ashmem region of size 4096
lying enough data to HAL, expected position 7760265 , only wrote 7760160
lying enough data to HAL, expected position 8066579 , only wrote 7914240
readingsimple E/AndroidRuntime: FATAL EXCEPTION: Thread-4
                                Process: course.examples.threading.threadingsimple, PID: 18730
                                android.view.ViewRootImpl$CalledFromWrongThreadException: Only the original thread that created a view hierarchy can touch its views.
                                    at android.view.ViewRootImpl.checkThread(ViewRootImpl.java:7286)
                                    at android.view.ViewRootImpl.requestLayout(ViewRootImpl.java:1155)
                                    at android.view.View.requestLayout(View.java:21922)
                                    at android.view.View.requestLayout(View.java:21922)
                                    at android.view.View.requestLayout(View.java:21922)
                                    at android.view.View.requestLayout(View.java:21922)
                                    at android.widget.RelativeLayout.requestLayout(RelativeLayout.java:360)
                                    at android.view.View.requestLayout(View.java:21922)
                                    at android.widget.ImageView.setImageDrawable(ImageView.java:570)
                                    at android.widget.ImageView.setImageBitmap(ImageView.java:704)
                                    at course.examples.threading.threadingsimple.SimpleThreadingExample$1.run(SimpleThreadingExample.java:44)
                                    at java.lang.Thread.run(Thread.java:764)
er:   Force finishing activity course.examples.threading.threadingsimple/.SimpleThreadingExample
er: Showing crash dialog for package course.examples.threading.threadingsimple u0
loc: Creating ashmem region of size 4096
loc: Creating ashmem region of size 4096


:03:19.926  2022: 2521 D/         ]
erface::setAsyncMode: set async mode 1
layer name: changing com.google.android.apps.nexuslauncher/com.google.android.apps.nexuslauncher.NexusLauncherActivity to com.google.android.apps.nexuslauncher/com.goo
loc: Creating ashmem region of size 4096
loc: Creating ashmem region of size 4096
searchbox:search D/EGL_emulation: eglMakeCurrent: 0xa3a067a0: ver 2 0 (tinfo 0xa3a03610)
loc: Creating ashmem region of size 4096
```

Gradle sync finished in 880ms (from cached state) (6 minutes ago)   828 chars, 25 line breaks 35:2 LF UTF-8 Git: master Context: <no context>

```
public void onClickLoadButton(View v) {
    new Thread(new Runnable() {
        public void run() {
            try {
                Thread.sleep(mDelay);
            } catch (InterruptedException e) {
                Log.e(TAG, e.toString());
            }
            // This doesn't work in Android
            mIView.setImageBitmap(BitmapFactory.decodeResource(getResources(),
                    R.drawable.painter));
        }
    }).start();
}
```

The UI Thread

Applications have a main thread (the UI thread)

Application components in the same process use the same UI thread

User interaction, system callbacks, and lifecycle methods handled on the UI thread

In addition, UI toolkit is not thread-safe

Implications

- Blocking the UI thread hurts application responsiveness
  - Long-running ops should run in background threads
- Don’t access the UI toolkit from a non-UI thread

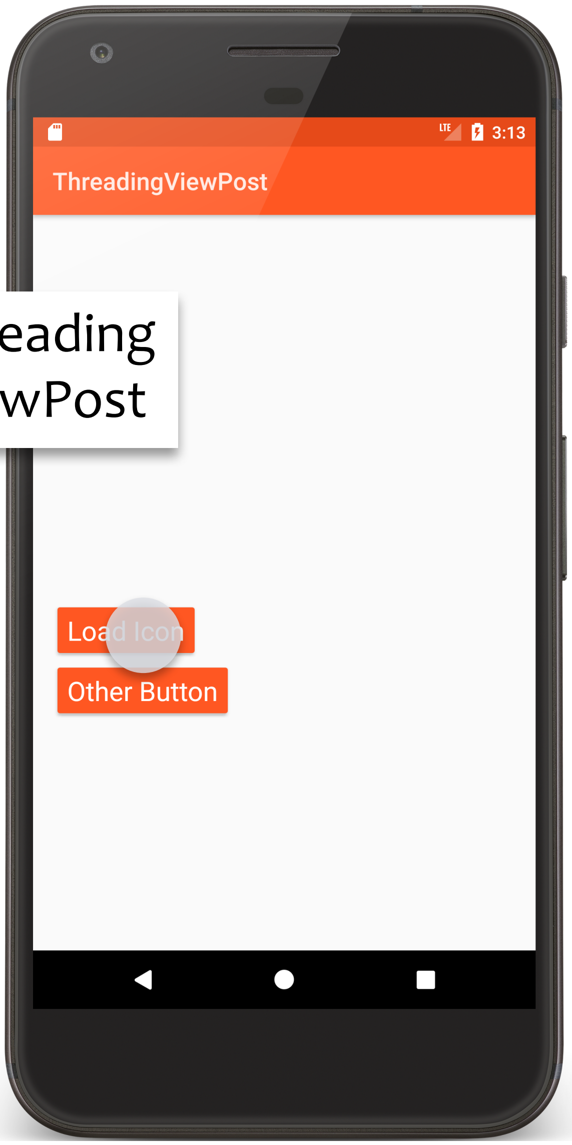
Improved Solution

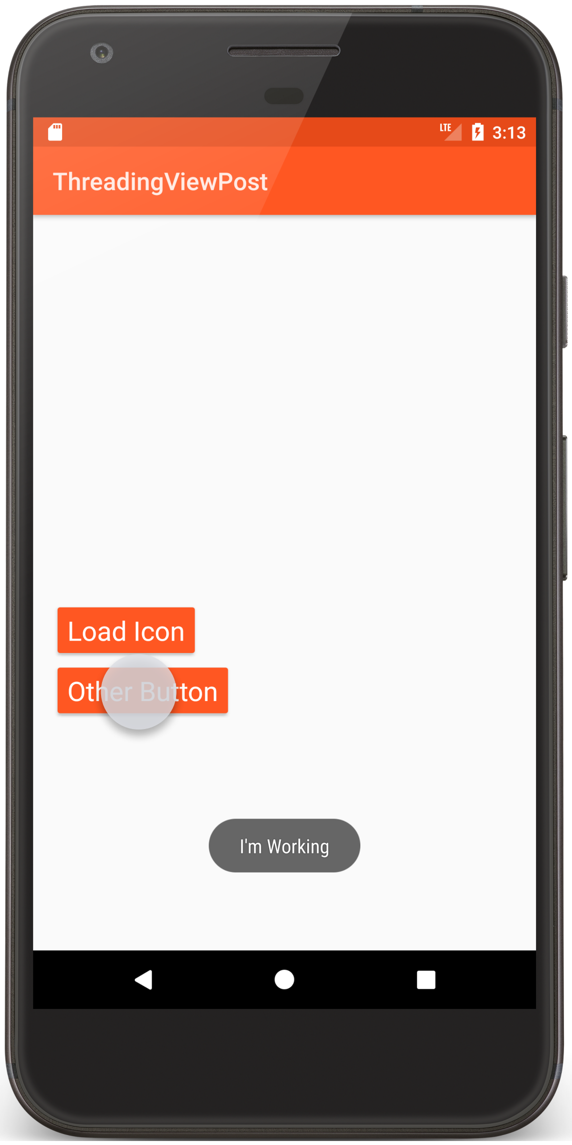
Do work on a background thread, but update the UI on the UI Thread

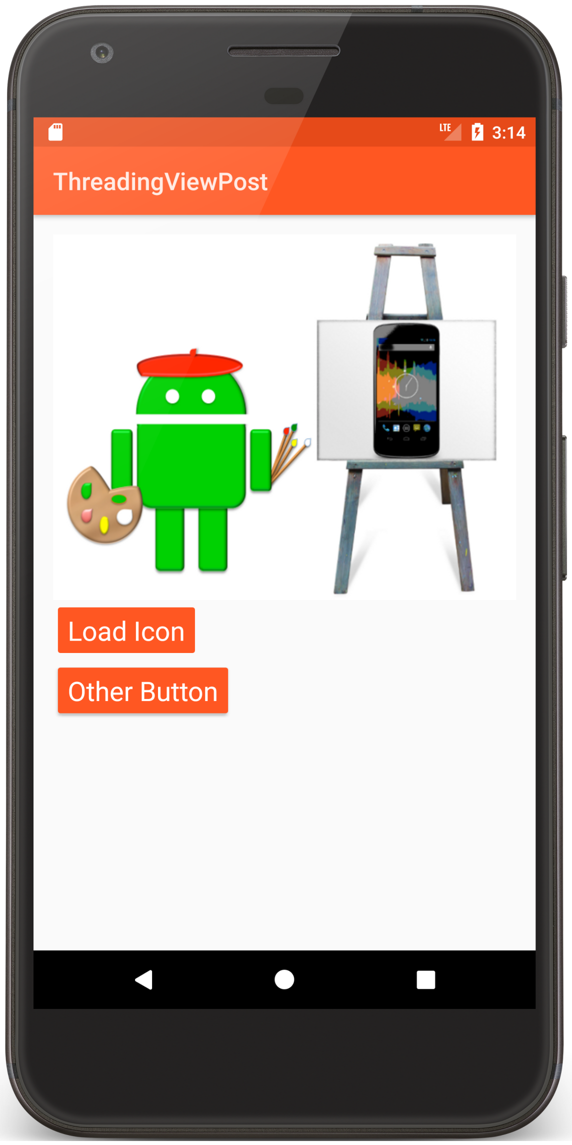
Android provides several methods that are guaranteed to run in the UI Thread

- boolean View.post (Runnable action)
- void Activity.runOnUiThread (Runnable action)

Threading ViewPost







```
public void onClickLoadButton(final View view) {
   view.setEnabled(false);
    …
   mImageView.post(new Runnable() {
            public void run() {
               mImageView.setImageBitmap(
                        BitmapFactory.decodeResource(getResources(), R.drawable.painter));
            }
         });
      }
   }).start();
}
```

AsyncTask

Provides a structured way to manage work involving background & UI Threads

AsyncTask

- Background Thread
  - Performs work
  - Indicates progress
- UI Thread
  - Does setup
  - Publishes intermediate progress
  - Uses results

AsyncTask

- Generic class
  ```
  class AsyncTask<Params, Progress, Result> {
     ...
  }
  ```
- Generic type parameters
  - Params – Type used in background work
  - Progress – Type used when indicating progress
  - Result – Type of result

AsyncTask

void onPreExecute()

Runs on UI Thread

Result doInBackground (Params…params)

Runs on background Thread

void publishProgress(Progress... values)

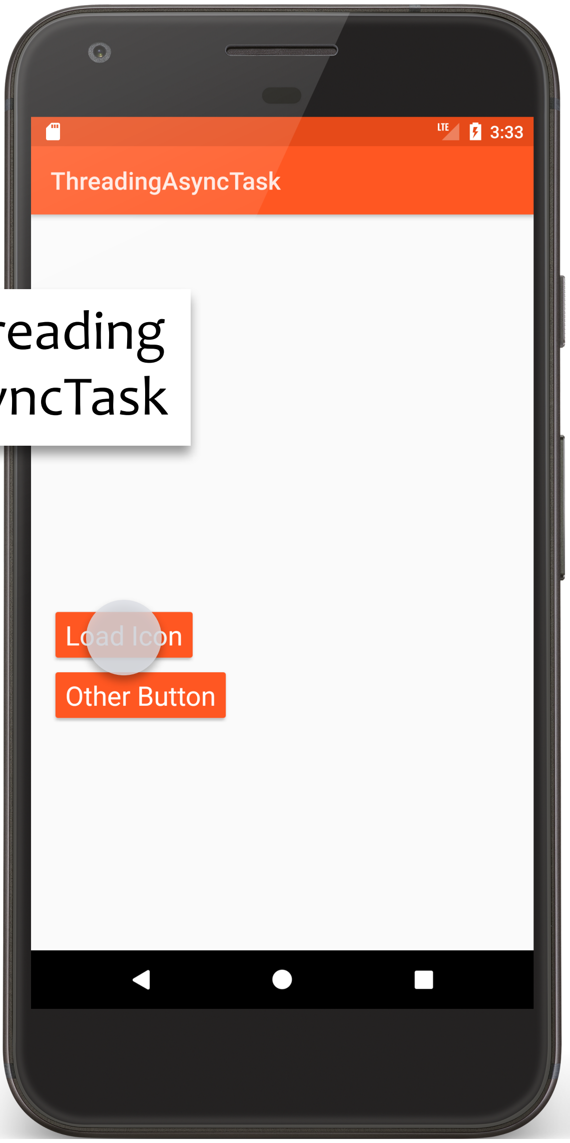
Can be called by doInBackground

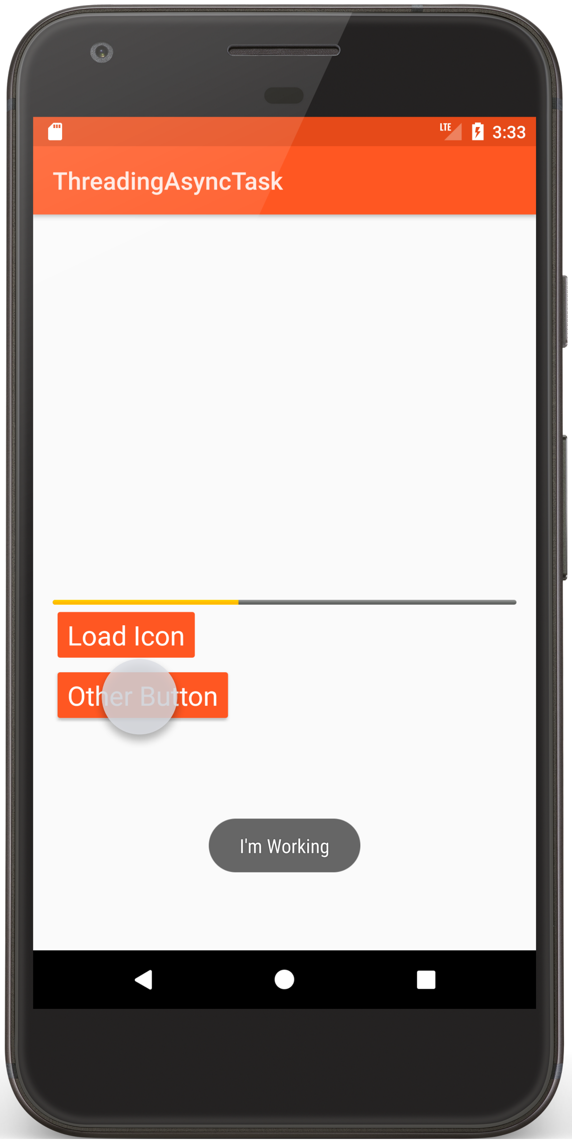
Runs on background Thread

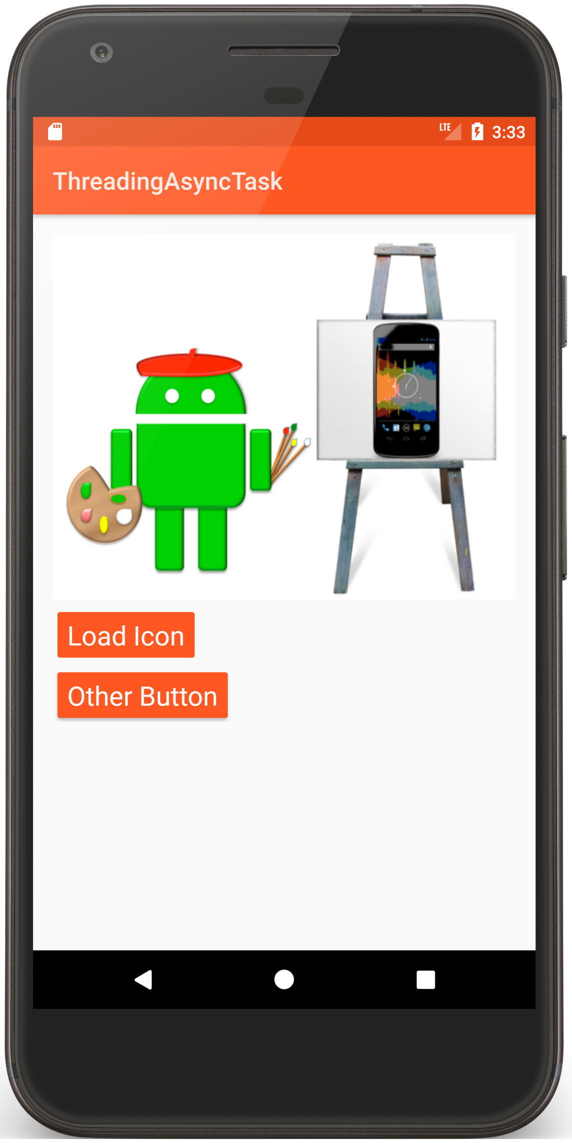
AsyncTask

- void onProgressUpdate (Progress... values)
  - Invoked in response to publishProgress()
  - Runs on UI Thread
- void onPostExecute (Result result)
  - Runs after doInBackground()
  - Runs in UI Thread

Threading AsyncTask







```java
// In AsyncTaskActivity.java
public void onClickLoadButton(View v) {
   mLoadButton.setEnabled(false);
   mAsyncTaskFragment.onButtonPressed();
}

// In AsyncTaskFragment.java
void onButtonPressed() {
   new LoadIconTask(this).execute(PAINTER);
}
```

```java
static class LoadIconTask extends AsyncTask<Integer, Integer, Bitmap> {
   // GC can reclaim weakly referenced variables.
   private final WeakReference<AsyncTaskFragment> mAsyncTaskFragment;
   LoadIconTask(AsyncTaskFragment fragment) {
      mAsyncTaskFragment = new WeakReference<>(fragment);
   }
   protected void onPreExecute() {
      mAsyncTaskFragment.get().setProgressBarVisibility(ProgressBar.VISIBLE);
   }
   protected Bitmap doInBackground(Integer... resId) {
      // simulating long-running operation
      for (int i = 1; i < 11; i++) {
         sleep();   publishProgress(i * 10);
      }
      return BitmapFactory.decodeResource(
                                      mAsyncTaskFragment.get().getResources(), resId[0]);
   }
```

```
protected void onProgressUpdate(Integer... values) {
   mAsyncTaskFragment.get().setProgress(values[0]);
}

protected void onPostExecute(Bitmap result) {
   mAsyncTaskFragment.get().setProgressBarVisibility(ProgressBar.INVISIBLE);
   mAsyncTaskFragment.get().setImageBitmap(result);
}
```

AsyncTask Threading Rules

The AsyncTask class must be loaded on the UI thread

The task instance must be created on the UI thread

execute(Params...) must be invoked on the UI thread

Do not call onPreExecute(), onPostExecute(Result), doInBackground(Params...), onProgressUpdate(Progress...) manually

The task can be executed only once. An exception will be thrown on violation

Dealing with Reconfiguration

Remember that Android kills and restarts Activities on reconfiguration

ThreadAsync gracefully handles reconfiguration

- Runs AsyncTask in Headless Fragment
- Saves and restores other Activity state

Examine ThreadingAsyncTask

Handler

Handler let you send and process Messages and Runnables to a Thread’s Message queue

Each Handler is bound to the Thread in which it was created

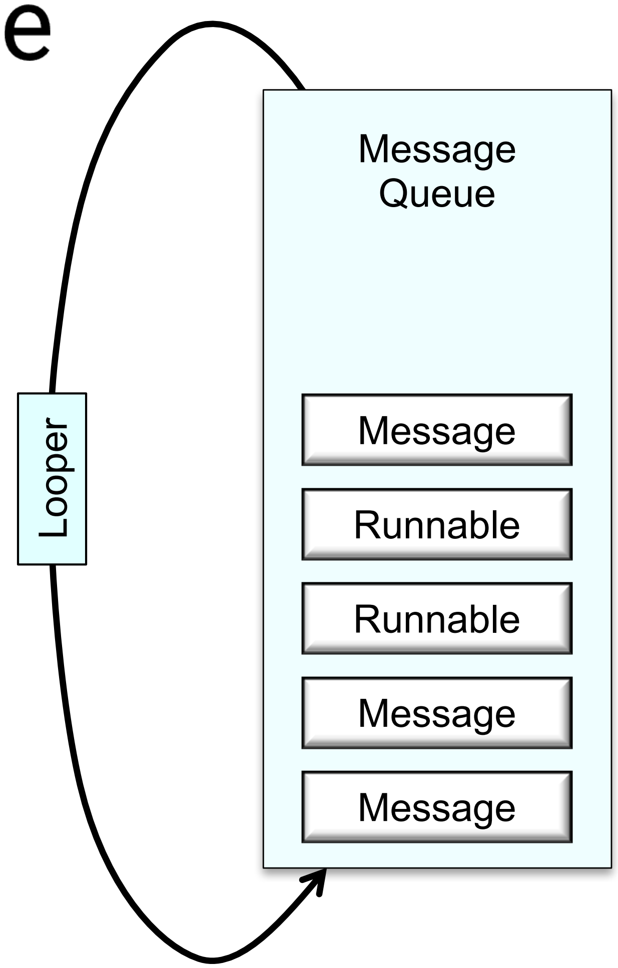
Handler Message Types

- Runnable
  - Contains an instance of the Runnable interface
  - Sender implements response
- Message
  - Can contain a message code, an object & integer arguments
  - Handler implements response

Handler Architecture

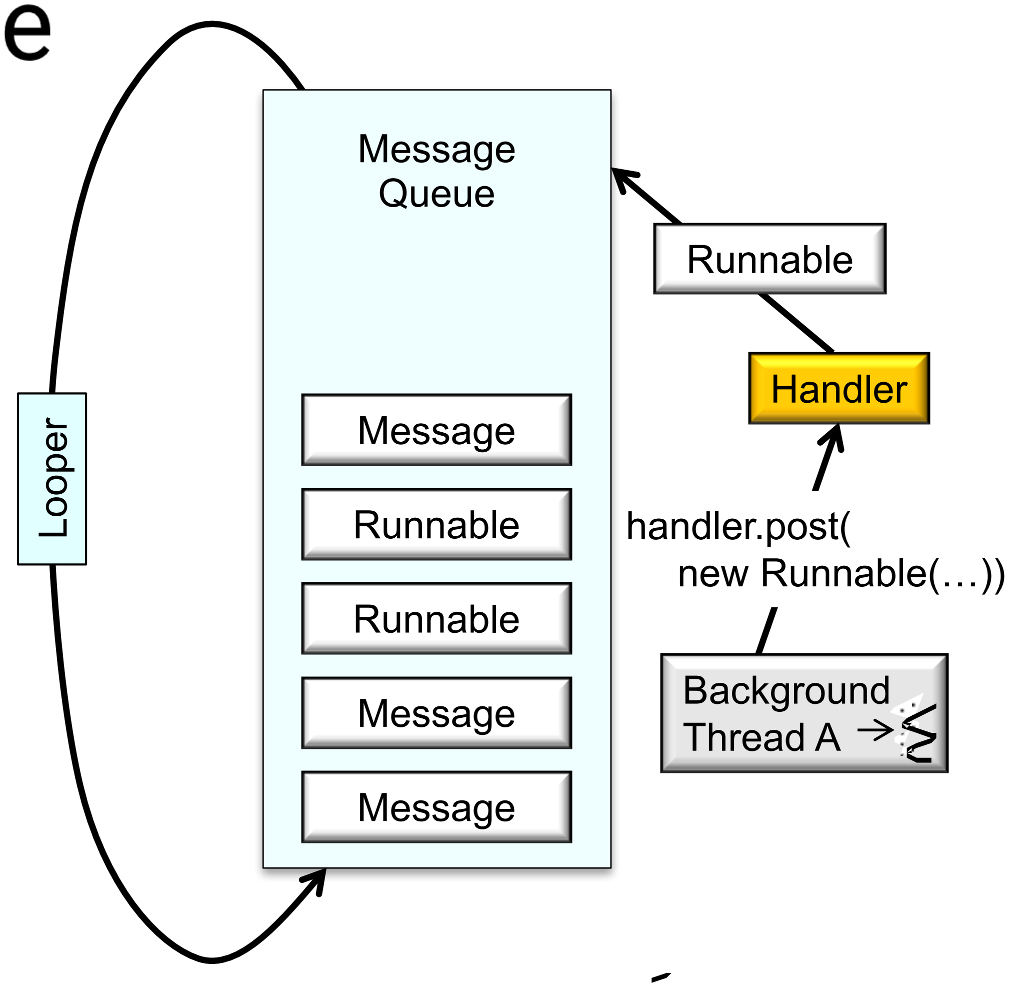
Each Android Thread is associated with a messageQueue & a Looper

A MessageQueue holds Messages and Runnables to be dispatched by the Looper



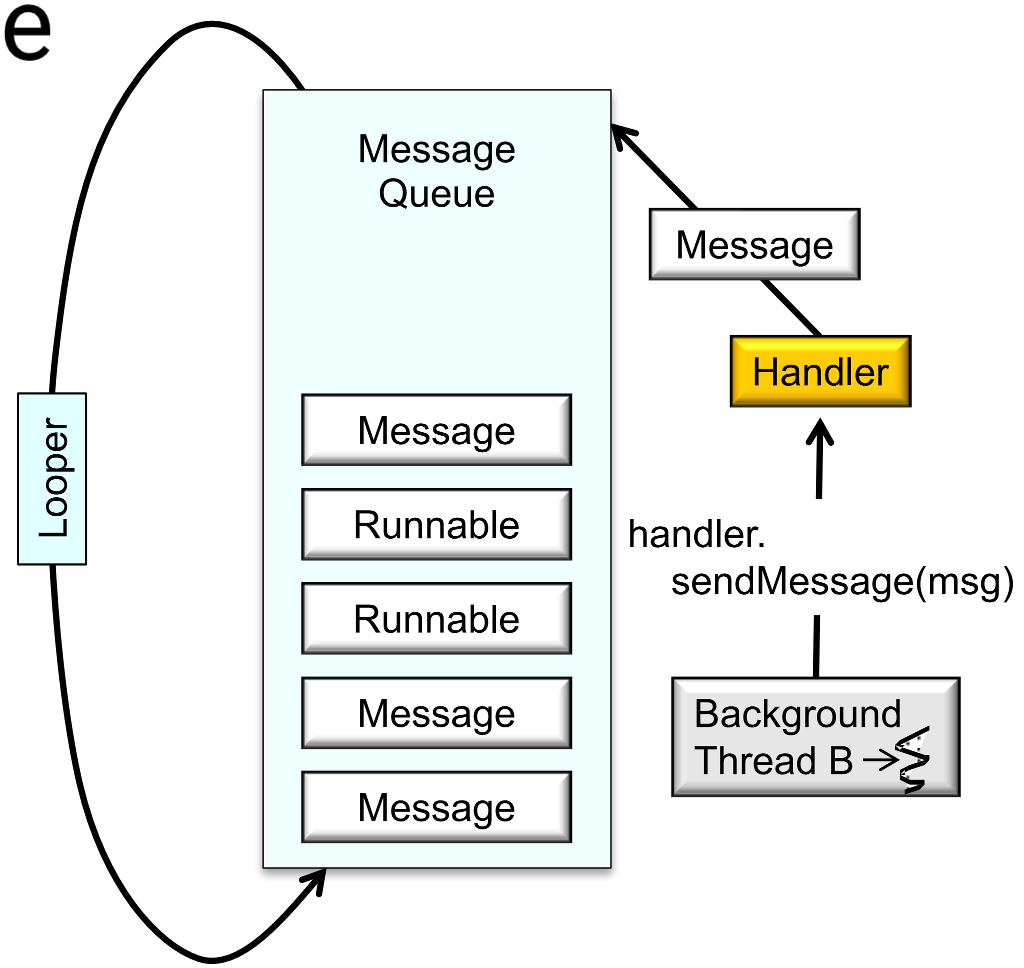
Handler Architecture

Add Runnables to MessageQueue by calling Handler's post() method



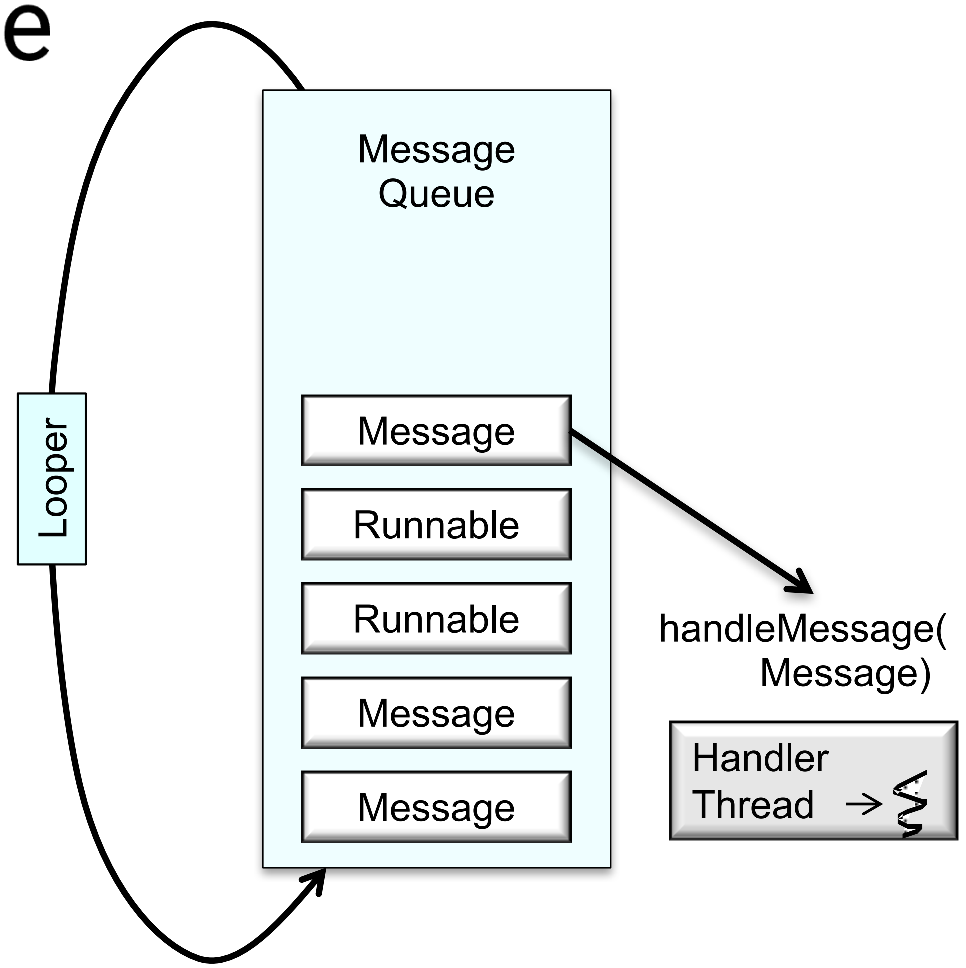
Handler Architecture

Add Messages to MessageQueue by calling Handler’s sendMessage() method



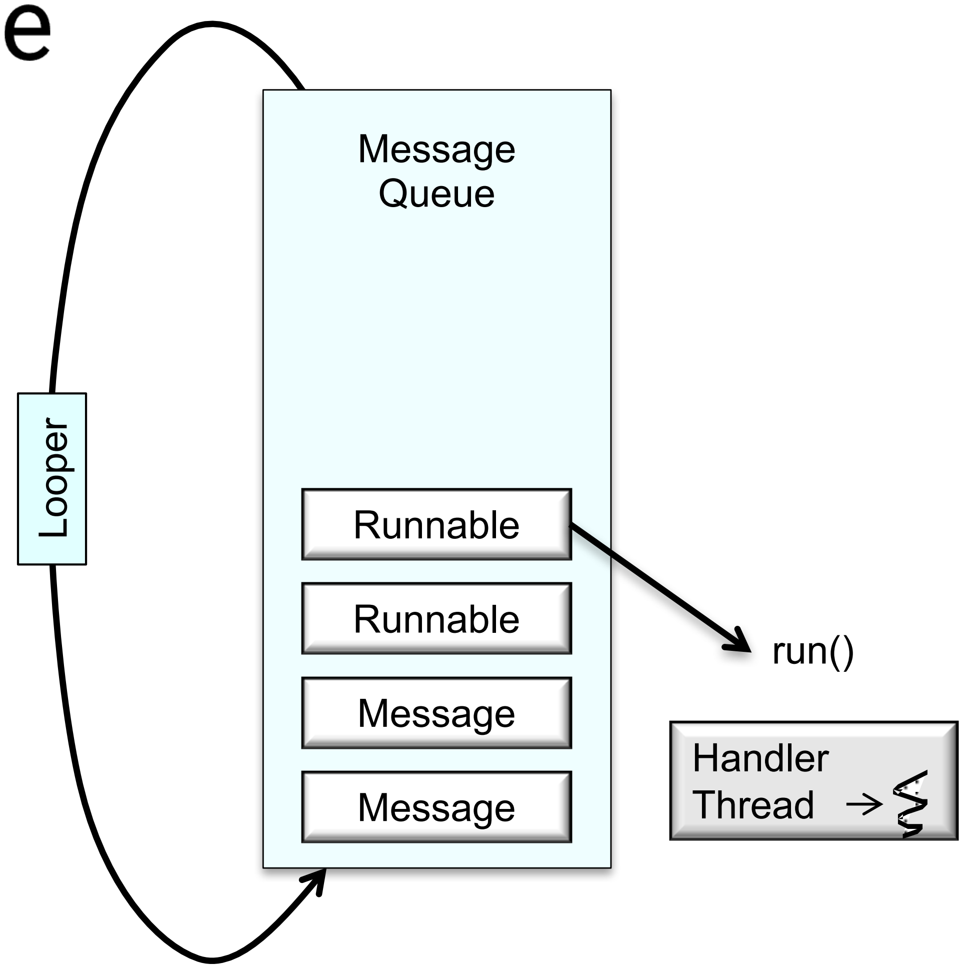
Handler Architecture

Looper dispatches Messages by calling the Handler's handleMessage() method on the Handler's Thread



Handler Architecture

Looper dispatches Runnables by calling their run() method in the Handler's Thread



Handler Methods for Runnables

boolean post(Runnable r)

Add Runnable to the MessageQueue

Boolean postAtTime(Runnable r, long uptimeMillis)

Add Runnable to the MessageQueue. Run at a specific time (based on SystemClock.upTimeMillis())

boolean postDelayed(Runnable r, long delayMillis)

Add Runnable to the message queue. Run after the specified amount of time elapses

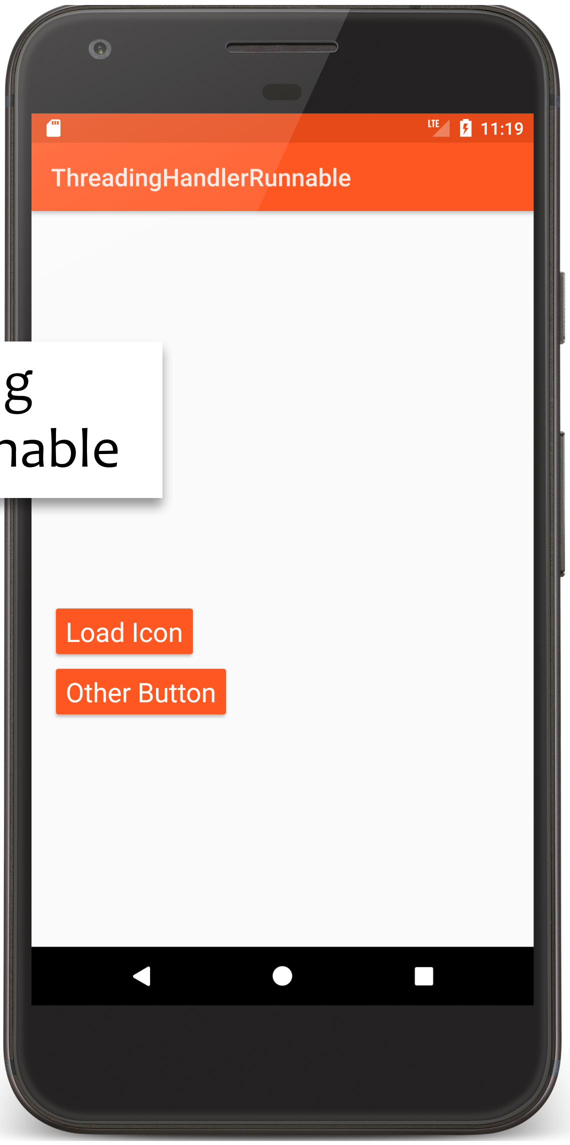
Handler Methods for Creating Messages

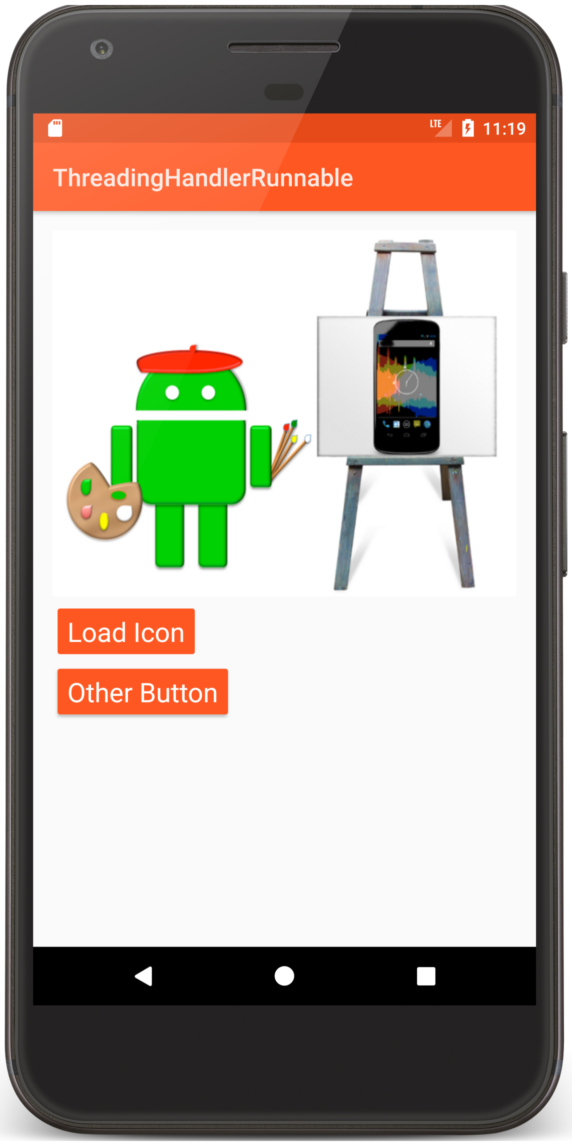
- Create Message & set Message content
  - Handler.obtainMessage()
  - Message.obtain()
- Message parameters include
  - int arg1, arg2, what
  - Object obj
  - Bundle data
- Many variants. See documentation

Handler Methods for Sending Messages

- sendMessage()
  - Queue Message now
- sendMessageAtFrontOfQueue()
  - Insert Message now at front of queue
- sendMessageAtTime()
  - Queue Message at the stated time
- sendMessageDelayed()
  - Queue Message after delay

Threading HandlerRunnable





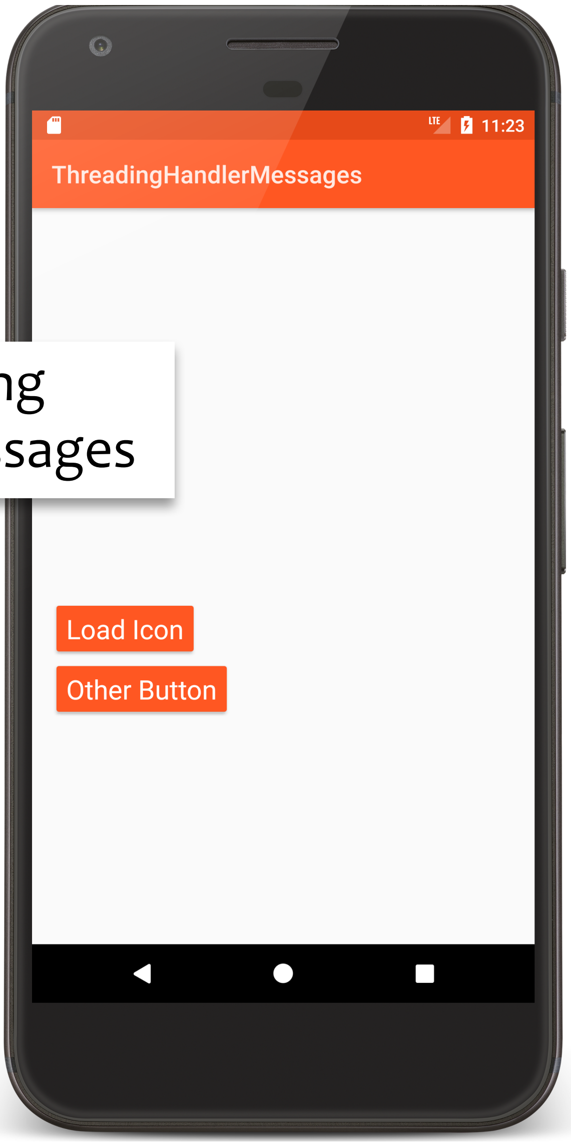
```
public class HandlerRunnableActivity extends Activity {
…
public void onClickLoadButton(View v) {
   v.setEnabled(false);
   mLoadIconTask = new LoadIconTask(getApplicationContext())
         .setmImageView(mImageView)
         .setmProgressBar(mProgressBar);
   mLoadIconTask.start();
}
```

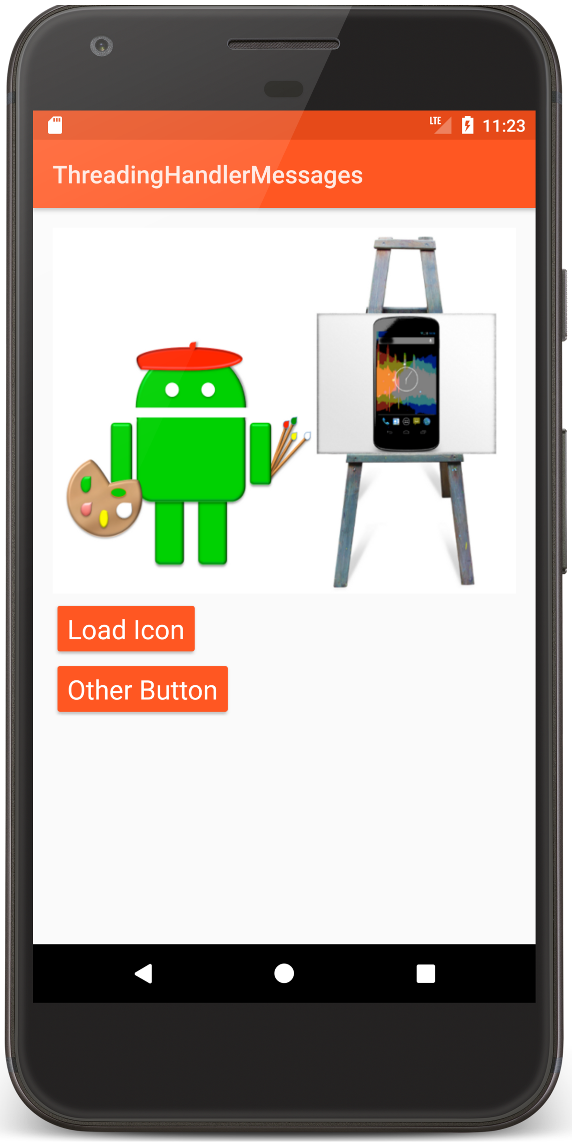
```
public class LoadIconTask extends Thread {
   …

   LoadIconTask(Context context) {
      mAppContext = context;
      mHandler = new Handler();
   }
```

```
public void run() {
   mHandler.post(new Runnable() {
      public void run() {  mProgressBar.setVisibility(ProgressBar.VISIBLE);  }});

   // Simulating long-running operation

   for (int i = 1; i < 11; i++) {
      sleep();
      final int step = i;
      mHandler.post(new Runnable() {
         public void run() { mProgressBar.setProgress(step * 10); }});
   }
…
```

```
mHandler.post(new Runnable() {
   public void run() { mImageView.setImageBitmap(
      BitmapFactory.decodeResource(mAppContext.getResources(),mBitmapResID)); }
});

mHandler.post(new Runnable() {
   public void run() { mProgressBar.setVisibility(ProgressBar.INVISIBLE); }
});
```

Threading
HandlerMessages





```
public class LoadIconTask extends Thread {
  …
  public void run() {
    Message msg = mHandler.obtainMessage(
       HandlerMessagesActivity.SET_PROGRESS_BAR_VISIBILITY,ProgressBar.VISIBLE);
    mHandler.sendMessage(msg);

    int mResId = R.drawable.painter;
    final Bitmap tmp = BitmapFactory.decodeResource(
                                                  mContext.getResources(), mResId);
    for (int i = 1; i < 11; i++) {
      sleep();
      msg = mHandler.obtainMessage(
                                HandlerMessagesActivity.PROGRESS_UPDATE, i * 10);
      mHandler.sendMessage(msg);
    }
   …
…
      msg = mHandler.obtainMessage(HandlerMessagesActivity.SET_BITMAP, tmp);
      mHandler.sendMessage(msg);

      msg = mHandler.obtainMessage(
          HandlerMessagesActivity.SET_PROGRESS_BAR_VISIBILITY, ProgressBar.INVISIBLE);
      mHandler.sendMessage(msg);
   }
```

```
static class UIHandler extends Handler {
  …
  public void handleMessage(Message msg) {
    switch (msg.what) {
      case HandlerMessagesActivity.SET_PROGRESS_BAR_VISIBILITY: {
        mProgressBar.setVisibility((Integer) msg.obj);
        break;
      }
      case HandlerMessagesActivity.PROGRESS_UPDATE: {
        mProgressBar.setProgress((Integer) msg.obj);
        break;
      }
      case HandlerMessagesActivity.SET_BITMAP: {
        mImageView.setImageBitmap((Bitmap) msg.obj);
        break;
      }
    }
  }
```

Next Time

- Alarms