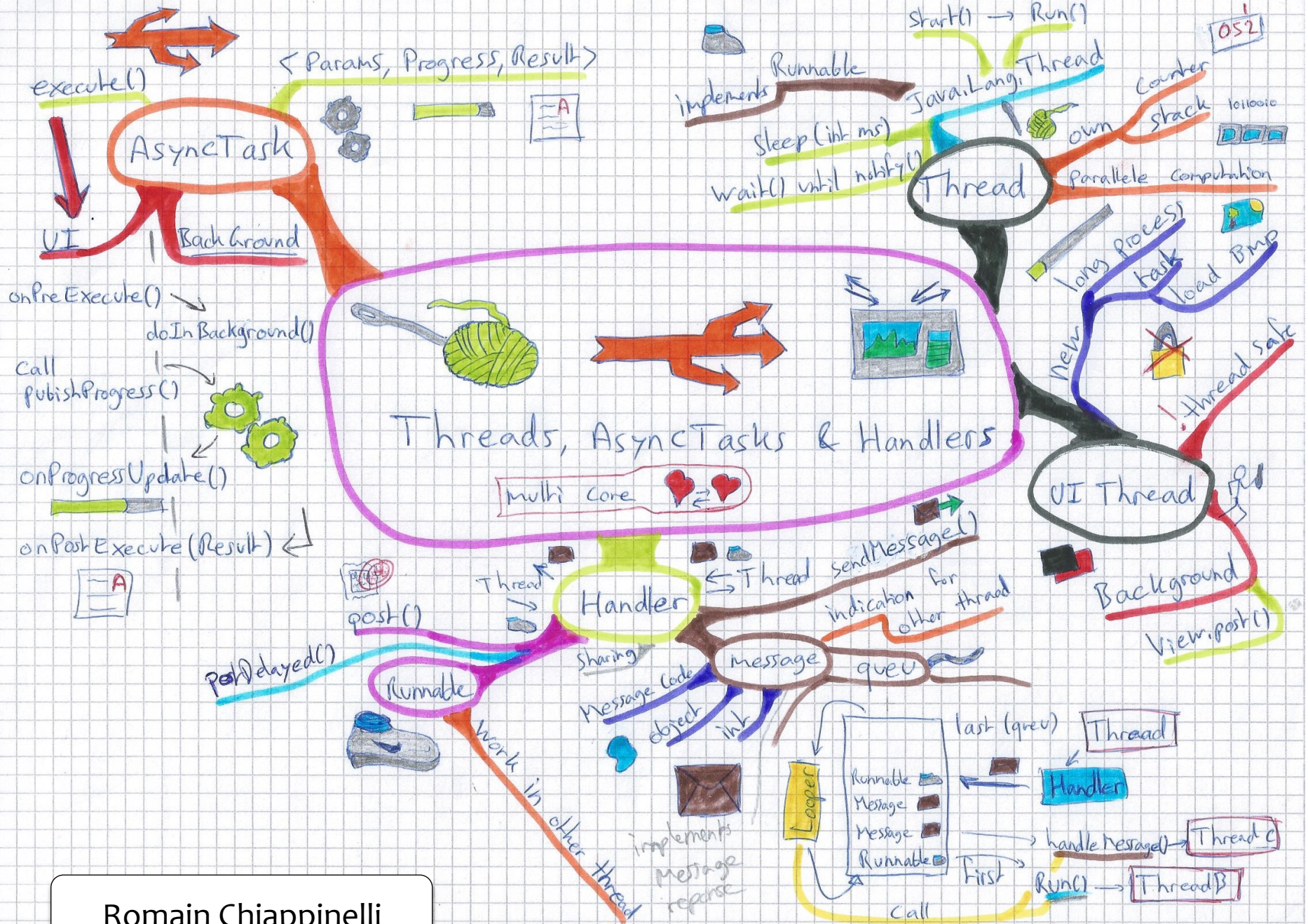


Design

Android

2010214



Romain Chiappinelli

PROGRAMMING HANDHELD SYSTEMS

ADAM PORTER

THREADS, ASYNCTASKS & HANDLERS

TODAY'S TOPICS

THREADING OVERVIEW

ANDROID'S UI THREAD

THE ASYNCTASK CLASS

THE HANDLER CLASS

WHAT IS A THREAD?

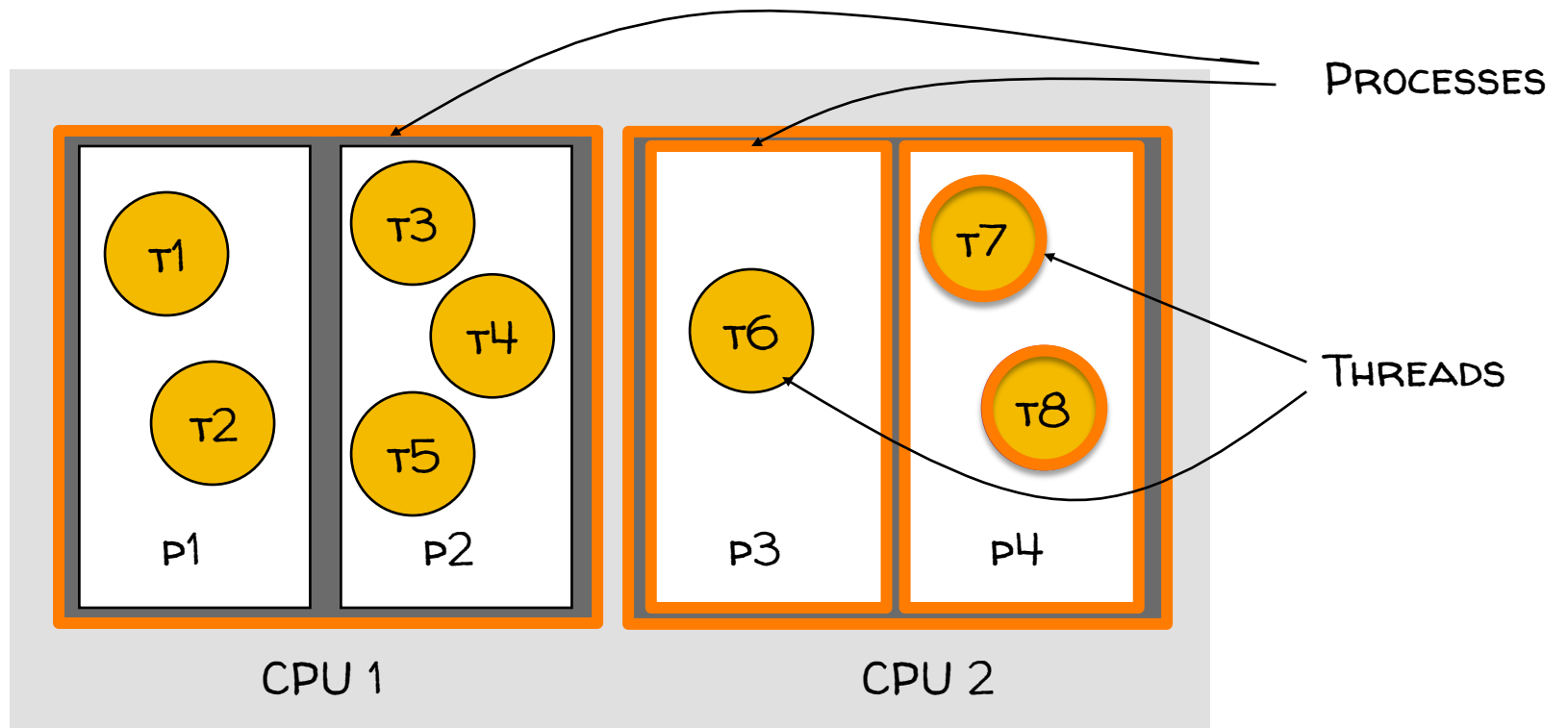
CONCEPTUAL VIEW

PARALLEL COMPUTATION RUNNING IN A PROCESS

IMPLEMENTATION VIEW

A PROGRAM COUNTER AND A STACK

WITH HEAP AND STATIC AREAS THAT ARE
SHARED WITH OTHER THREADS



JAVA THREADS

REPRESENTED BY AN OBJECT OF TYPE
JAVA.LANG.THREAD

THREADS IMPLEMENT THE RUNNABLE
INTERFACE

```
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

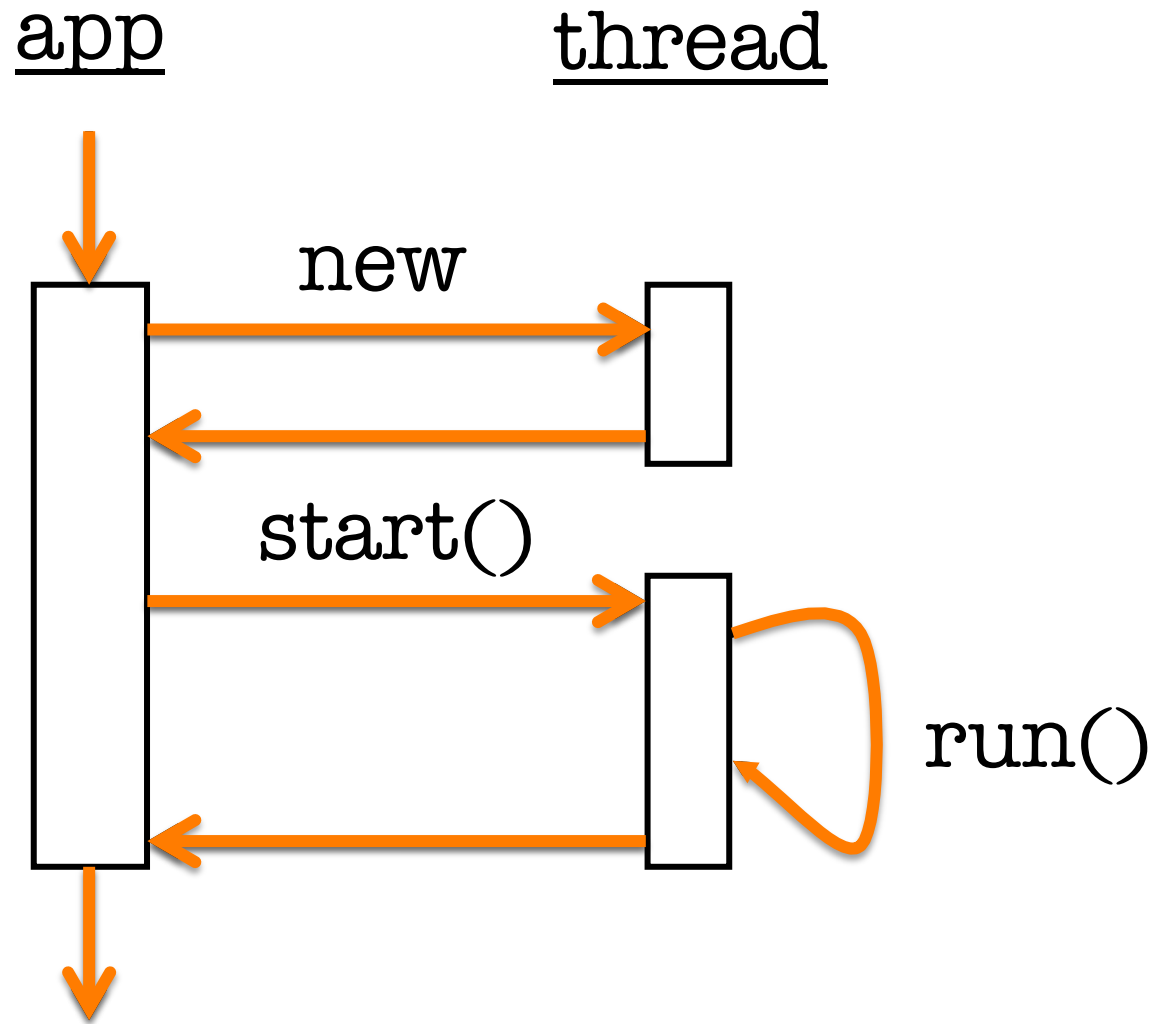
INSTANTIATE A THREAD OBJECT

INVOKE THE THREAD'S `start()` METHOD

THREAD'S `run()` METHOD GET CALLED

THREAD TERMINATES WHEN `run()` RETURNS

BASIC THREAD USE CASE



THREADING NO THREADING

APPLICATION DISPLAYS TWO BUTTONS

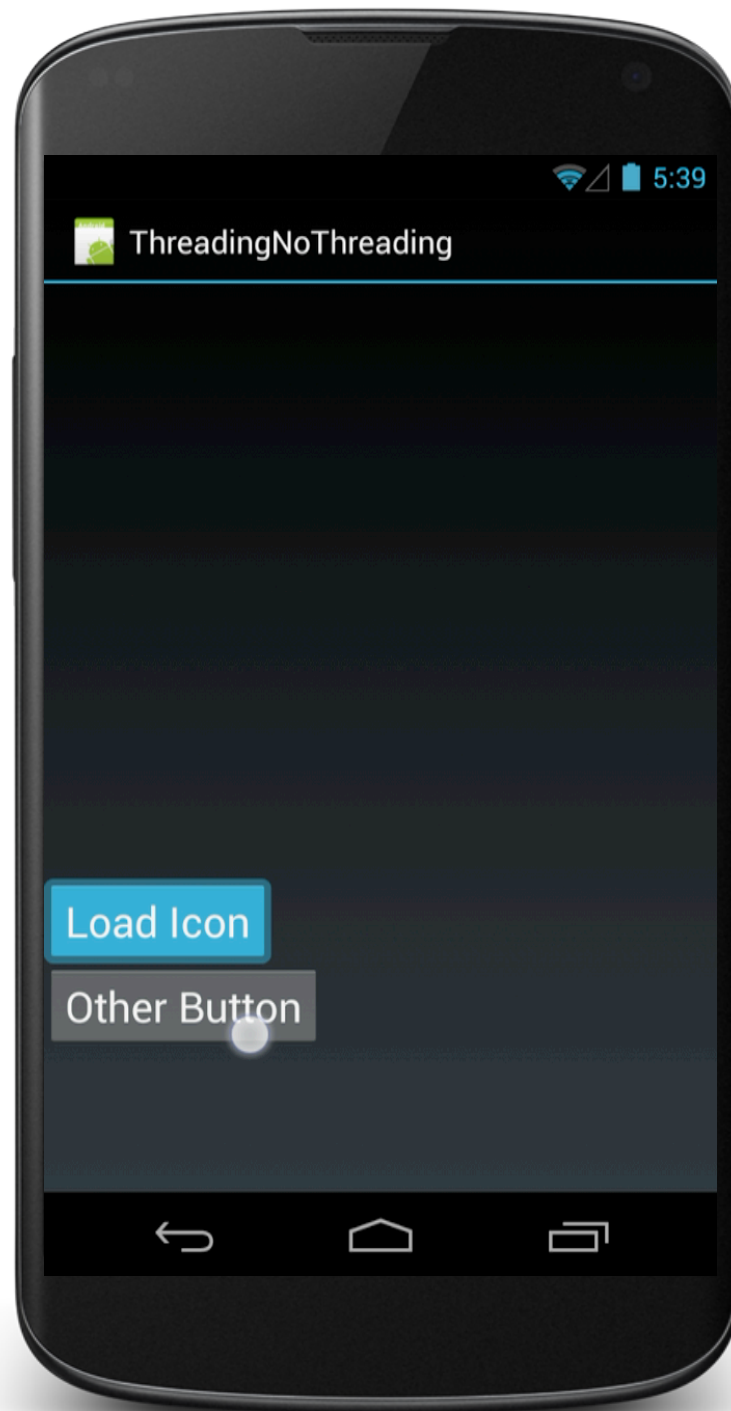
LOADICON

LOAD A BITMAP FROM A RESOURCE FILE &
DISPLAY

SHOW LOADED BITMAP

OTHER BUTTON

DISPLAY SOME TEXT



THREADINGSIMPLE

SEEMINGLY OBVIOUS, BUT INCORRECT, SOLUTION:

BUTTON LISTENER SPAWNS A SEPARATE
THREAD TO LOAD BITMAP & DISPLAY IT

THREADINGSIMPLE

```
public class SimpleThreadingExample extends Activity {

    private static final String TAG = "SimpleThreadingExample";

    private Bitmap mBitmap;
    private ImageView mIView;
    private int mDelay = 5000;

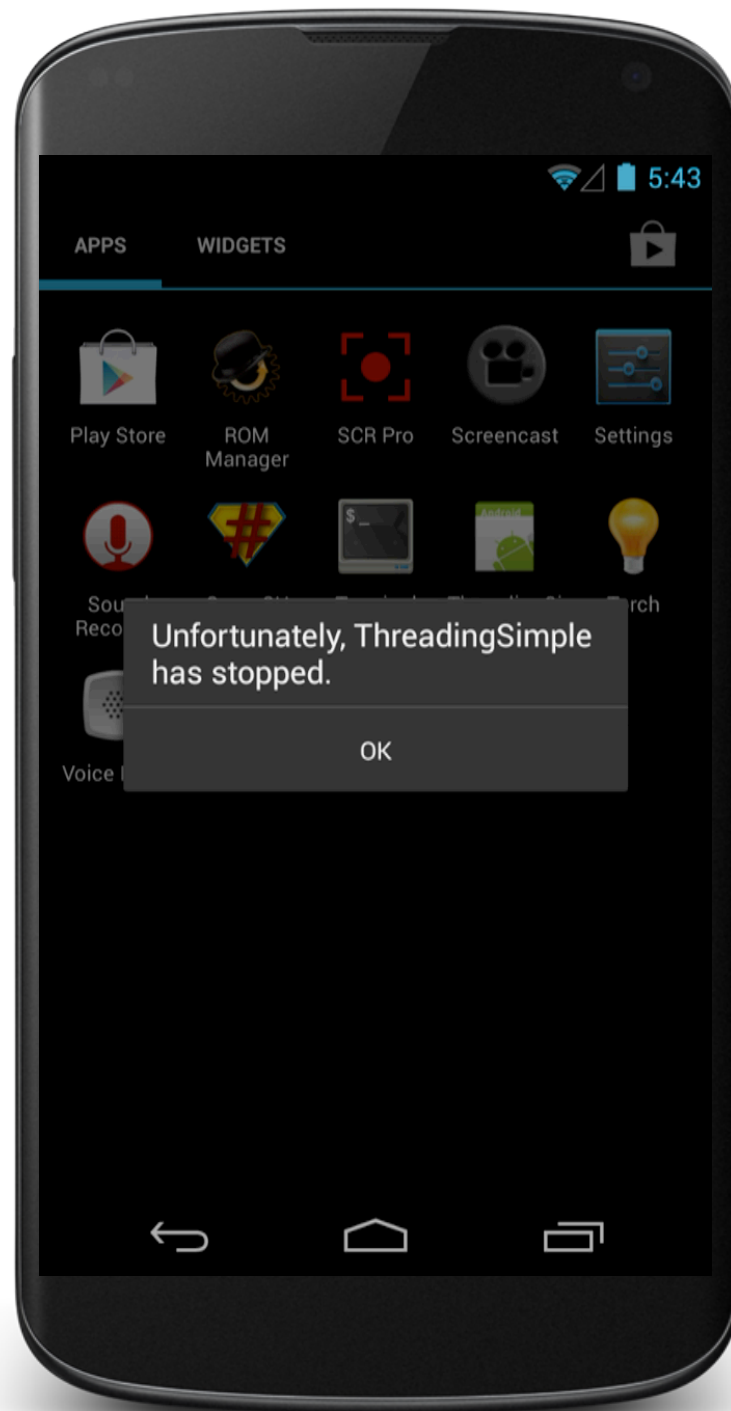
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        mIView = (ImageView) findViewById(R.id.imageView);

        final Button loadButton = (Button) findViewById(R.id.loadButton);
        loadButton.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                loadIcon();
            }
        });

        final Button otherButton = (Button) findViewById(R.id.otherButton);
        otherButton.setOnClickListener(new OnClickListener() {
            @Override
            public void onClick(View v) {
                Toast.makeText(SimpleThreadingExample.this, "I'm Working",
                    Toast.LENGTH_SHORT).show();
            }
        });
    }
}
```


THREADINGSIMPLE

```
private void loadIcon() {  
    new Thread(new Runnable() {  
        @Override  
        public void run() {  
            try {  
                Thread.sleep(mDelay);  
            } catch (InterruptedException e) {  
                Log.e(TAG, e.toString());  
            }  
            mBitmap = BitmapFactory.decodeResource(getResources(),  
                R.drawable.painter);  
  
            // This doesn't work in Android  
            mImageView.setImageBitmap(mBitmap);  
        }  
    }).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 & LIFECYCLE METHODS HANDLED IN THE UI THREAD

IN ADDITION, UI TOOLKIT IS NOT THREAD-SAFE

IMPLICATIONS

BLOCKING THE UI THREAD HURTS
APPLICATION RESPONSIVENESS

LONG-RUNNING OPERATIONS SHOULD RUN IN
BACKGROUND THREADS

DON'T ACCESS THE UI TOOLKIT FROM A
NON-UI THREAD

IMPROVED SOLUTION

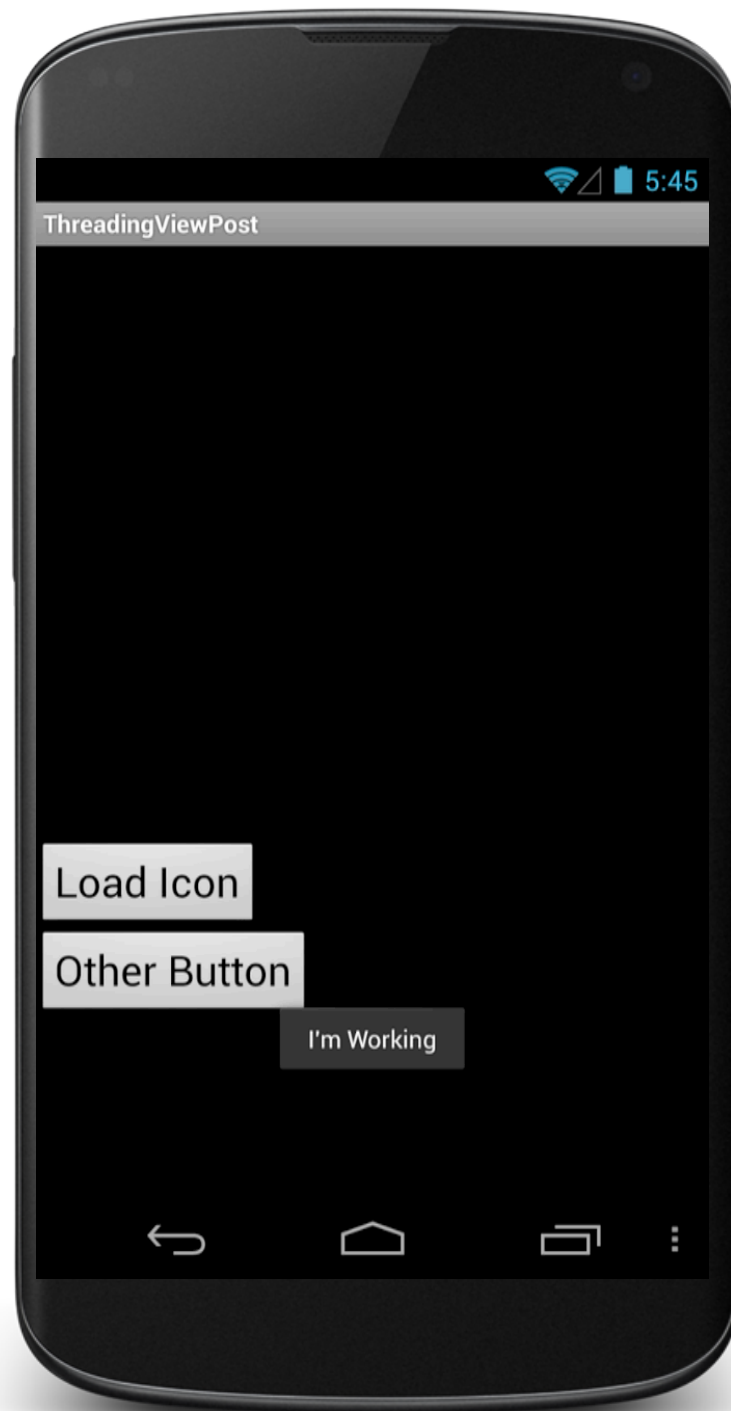
NEED TO DO WORK IN A BACKGROUND
THREAD, BUT UPDATE THE UI IN THE UI
THREAD

ANDROID PROVIDES SEVERAL METHODS
THAT ARE GUARANTEED TO RUN IN THE
UI THREAD

`boolean View.post (Runnable action)`

`void Activity.`

`runOnUiThread (Runnable action)`



THREADINGVIEWPOST

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    mImageView = (ImageView) findViewById(R.id.imageView);

    final Button button = (Button) findViewById(R.id.loadButton);
    button.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View v) {
            loadIcon();
        }
    });

    final Button otherButton = (Button) findViewById(R.id.otherButton);
    otherButton.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View v) {
            Toast.makeText(SimpleThreadingViewPostActivity.this, "I'm Working",
                Toast.LENGTH_SHORT).show();
        }
    });
}
```


THREADINGVIEWPOST

```
private void loadIcon() {
    new Thread(new Runnable() {
        @Override
        public void run() {
            try {
                Thread.sleep(mDelay);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }

            mBitmap = BitmapFactory.decodeResource(getResources(),
                R.drawable.painter);
            mImageView.post(new Runnable() {
                @Override
                public void run() {

                    mImageView.setImageBitmap(mBitmap);
                }
            });
        }
    }).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 IN UI THREAD BEFORE  doInBackground()

Result

doInBackground (Params...params)

PERFORMS WORK IN BACKGROUND THREAD

MAY CALL

void publishProgress(Progress... values)

ASYNCTASK

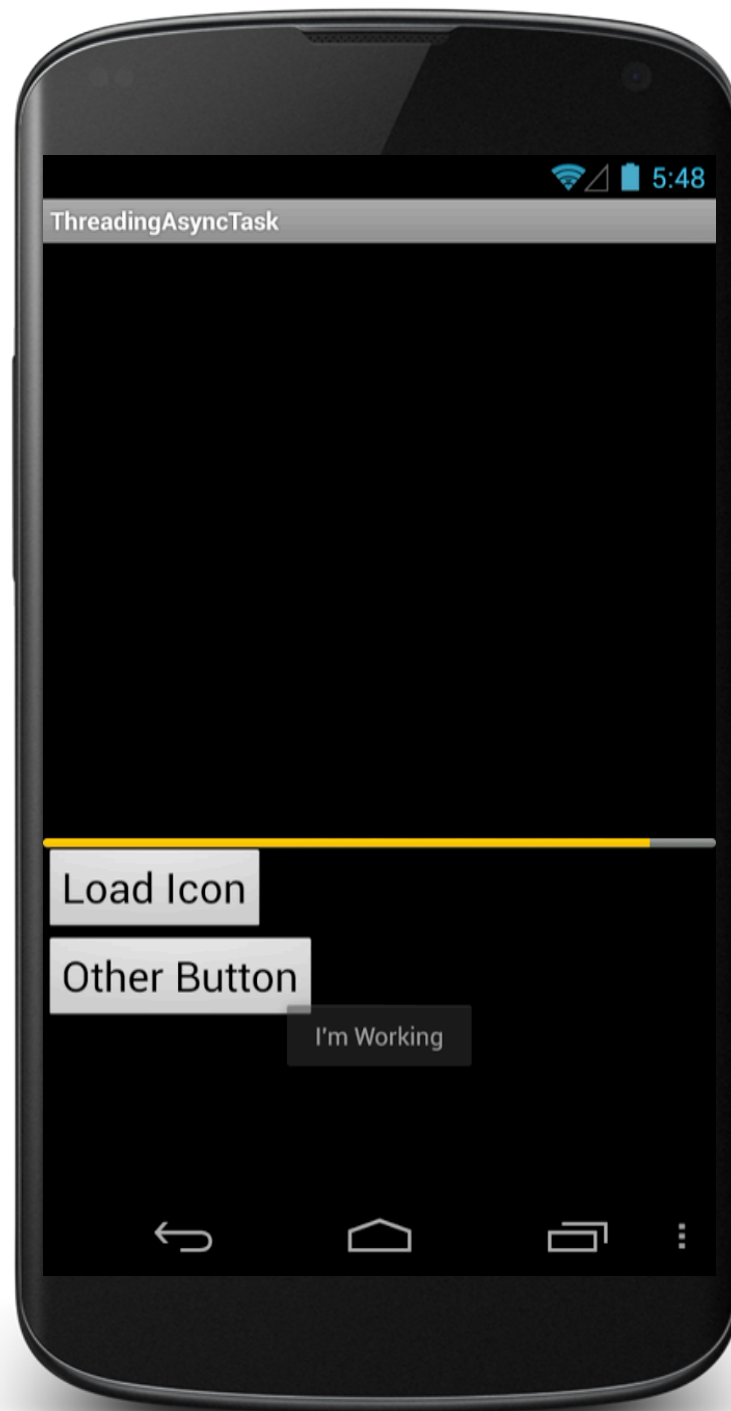
void

onProgressUpdate (Progress... values)

INVOKED IN RESPONSE TO publishProgress()

void onPostExecute (Result result)

RUNS AFTER doInBackground()



THREADINGASYNC TASK

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    mImageView = (ImageView) findViewById(R.id.imageView);
    mProgressBar = (ProgressBar) findViewById(R.id.progressBar);

    final Button button = (Button) findViewById(R.id.loadButton);
    button.setOnClickListener(new OnClickListener() {
        public void onClick(View v) {
            new LoadIconTask().execute(R.drawable.painter);
        }
    });

    final Button otherButton = (Button) findViewById(R.id.otherButton);
    otherButton.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View v) {
            Toast.makeText(AsyncTaskActivity.this, "I'm Working",
                Toast.LENGTH_SHORT).show();
        }
    });
}
```

THREADINGASYNC TASK

```
class LoadIconTask extends AsyncTask<Integer, Integer, Bitmap> {

    @Override
    protected void onPreExecute() {
        mProgressBar.setVisibility(ProgressBar.VISIBLE);
    }


    @Override
    protected Bitmap doInBackground(Integer... resId) {
        Bitmap tmp = BitmapFactory.decodeResource(getResources(), resId[0]);
        // simulating long-running operation
        for (int i = 1; i < 11; i++) {
            sleep();
            publishProgress(i * 10);
        }
        return tmp;
    }
}
```

THREADINGASYNC TASK

```
.  
  
@Override  
protected void onProgressUpdate(Integer... values) {  
    mProgressBar.setProgress(values[0]);  
}  
  
@Override  
protected void onPostExecute(Bitmap result) {  
    mProgressBar.setVisibility(ProgressBar.INVISIBLE);  
    mImageView.setImageBitmap(result);  
}  
  
private void sleep() {  
    try {  
        Thread.sleep(mDelay);  
    } catch (InterruptedException e) {  
        Log.e(TAG, e.toString());  
    }  
}  
}
```

HANDLER

EACH HANDLER IS ASSOCIATED WITH A
THREAD

ONE THREAD CAN HAND OFF WORK TO
ANOTHER THREAD BY SENDING
MESSAGES  POSTING RUNNABLES TO A
HANDLER ASSOCIATED WITH THE OTHER
THREAD

HANDLER

RUNNABLE

CONTAINS AN INSTANCE OF THE RUNNABLE
INTERFACE

SENDER IMPLEMENTS RESPONSE

MESSAGE

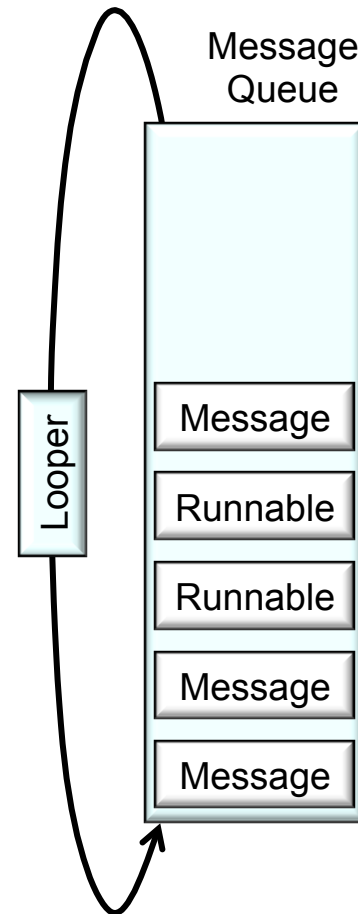
CAN CONTAIN A MESSAGE CODE, AN OBJECT &
INTEGER ARGUMENTS

HANDLER IMPLEMENTS RESPONSE

HANDLER ARCHITECTURE

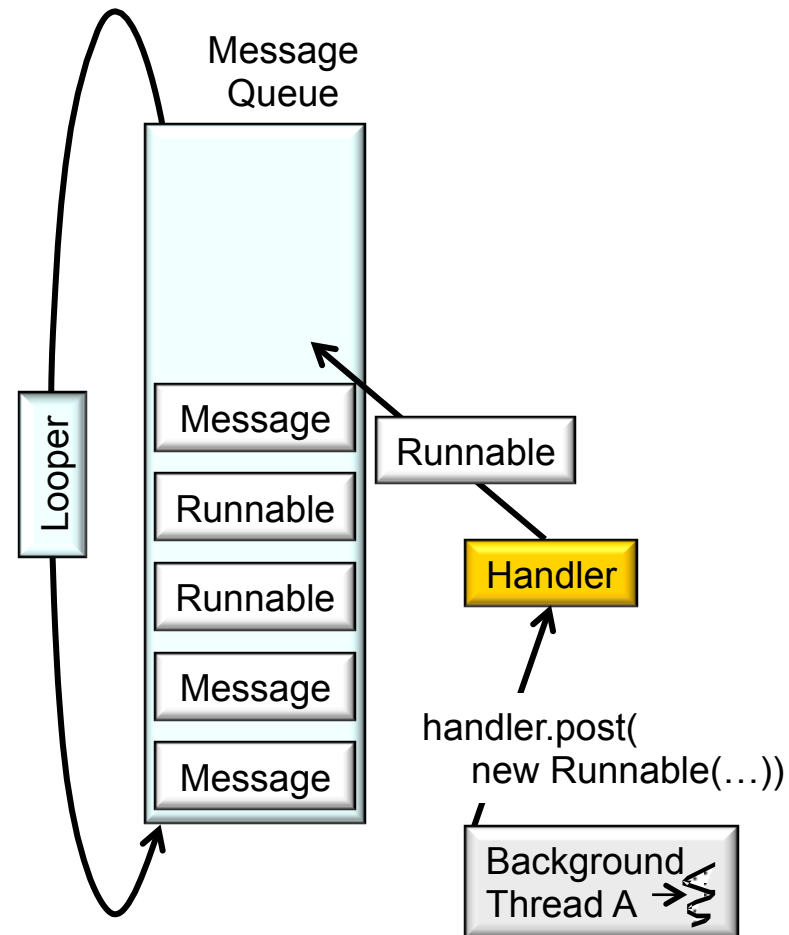
EACH ANDROID
THREAD IS
ASSOCIATED WITH A
MESSAGEQUEUE & A
LOOPER

A MESSAGEQUEUE
HOLDS MESSAGES
AND RUNNABLES TO
BE DISPATCHED BY
THE LOOPER



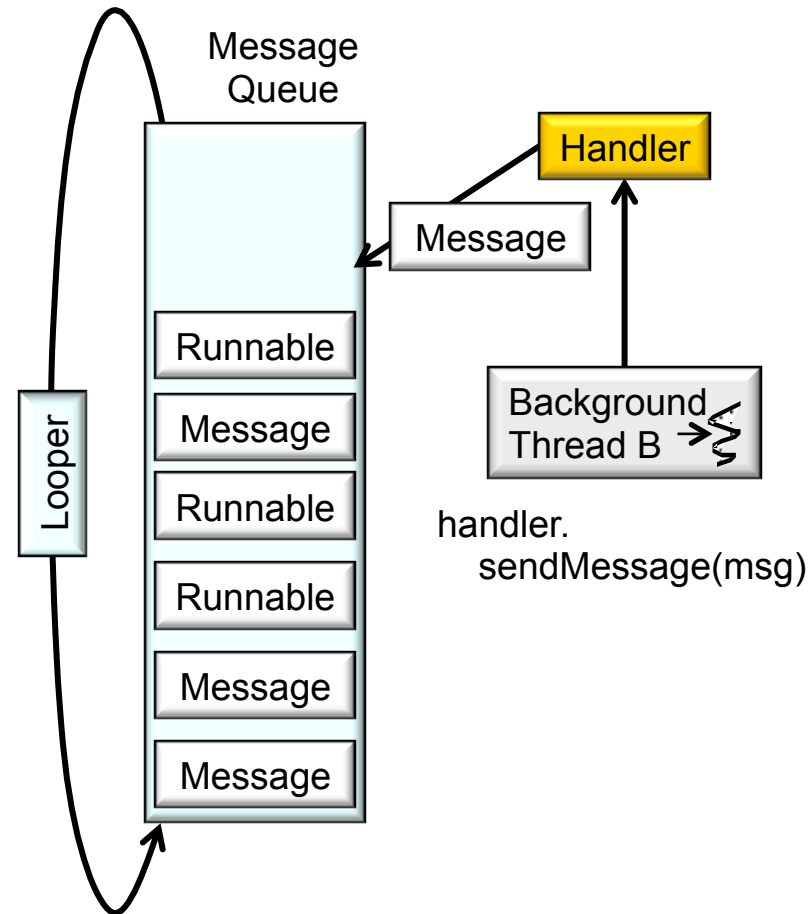
HANDLER ARCHITECTURE

ADD RUNNABLES TO
MESSAGEQUEUE BY
CALLING HANDLER'S
post() METHOD



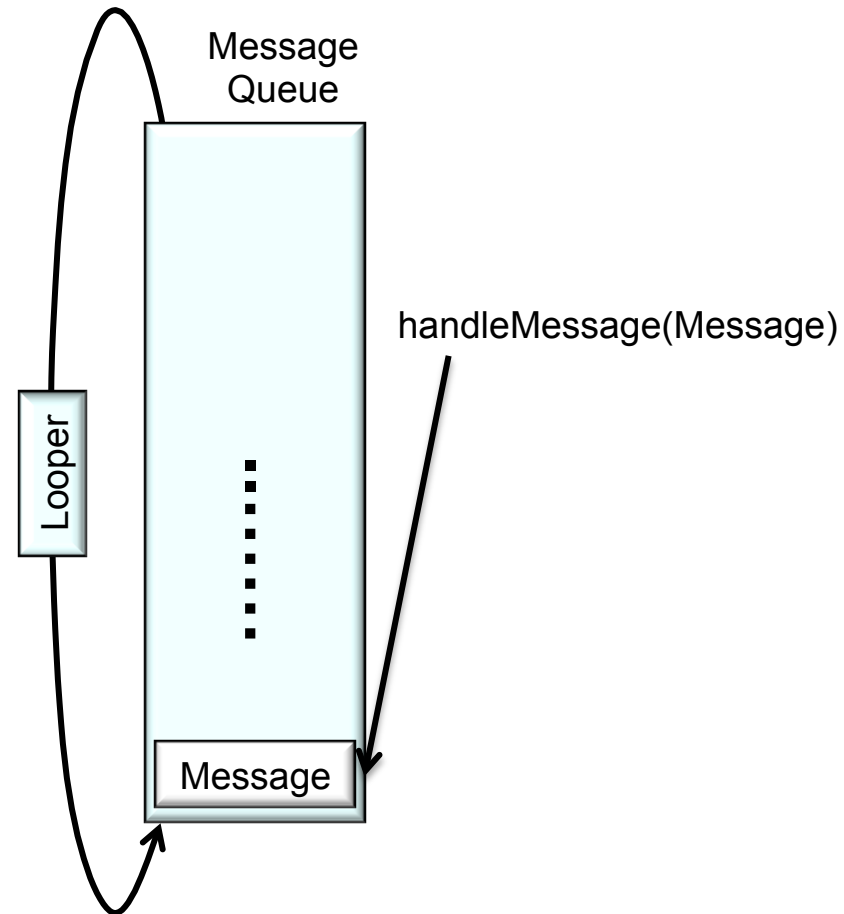
HANDLER ARCHITECTURE

ADD MESSAGES TO
MESSAGEQUEUE BY
CALLING HANDLER'S
sendMessage()
METHOD



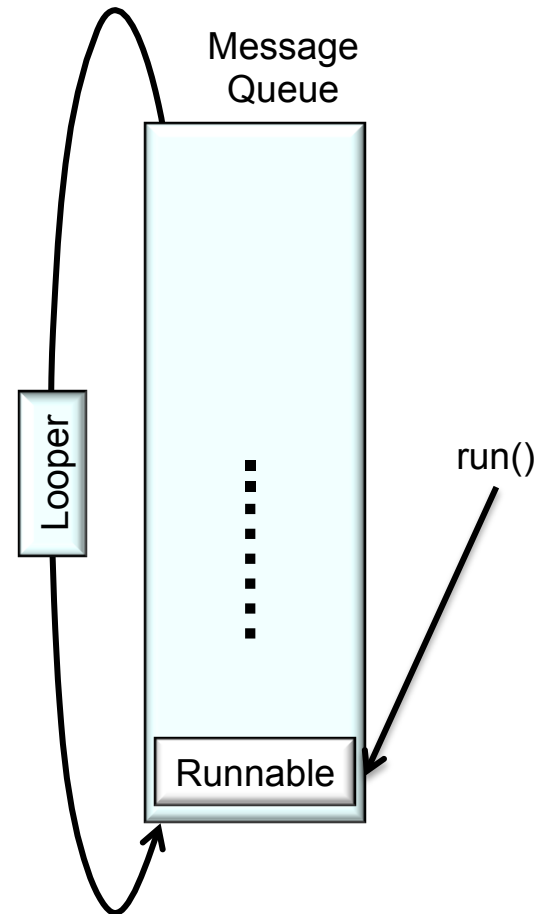
HANDLER ARCHITECTURE

LOOPER DISPATCHES
MESSAGES BY
CALLING THE
HANDLER'S
`handleMessage()`
METHOD IN THE
MESSAGEQUEUE'S
THREAD



HANDLER ARCHITECTURE

LOOPER DISPATCHES
RUNNABLES BY
CALLING THEIR `run()`
METHOD IN THE
MESSAGEQUEUE'S
THREAD



RUNNABLES & HANDLERS

`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

MESSAGES & HANDLERS

CREATE MESSAGE & SET MESSAGE CONTENT

HANDLER.OBTAINMESSAGE()

MESSAGE.OBTAIN() 

MESSAGE PARAMETERS INCLUDE

INT ARG1, ARG2, WHAT

OBJECT OBJ

BUNDLE DATA

MANY VARIANTS. SEE DOCUMENTATION

MESSAGES & HANDLERS

sendMessage()

QUEUE MESSAGE NOW

sendMessageAtFrontOfQueue()

INSERT MESSAGE NOW AT FRONT OF QUEUE

sendMessageAtTime()

QUEUE MESSAGE AT THE STATED TIME

sendMessageDelayed()

QUEUE MESSAGE AFTER DELAY

THREADINGHANDLERMESSAGES

```
static class UIHandler extends Handler {
    WeakReference<HandlerMessagesActivity> mParent;

    public UIHandler(WeakReference<HandlerMessagesActivity> parent) {
        mParent = parent;
    }

    @Override
    public void handleMessage(Message msg) {
        HandlerMessagesActivity parent = mParent.get();
        if (null != parent) {
            switch (msg.what) {
                case SET_PROGRESS_BAR_VISIBILITY: {
                    parent.getProgressBar().setVisibility((Integer) msg.obj);
                    break;
                }
                case PROGRESS_UPDATE: {
                    parent.getProgressBar().setProgress((Integer) msg.obj);
                    break;
                }
                case SET_BITMAP: {
                    parent.getImageView().setImageBitmap((Bitmap) msg.obj);
                    break;
                }
            }
        }
    }
}

Handler handler = new UIHandler(new WeakReference<HandlerMessagesActivity>(
    this));
```

THREADINGHANDLERMESSAGES

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    mImageView = (ImageView) findViewById(R.id.imageView);
    mProgressBar = (ProgressBar) findViewById(R.id.progressBar);

    final Button button = (Button) findViewById(R.id.loadButton);
    button.setOnClickListener(new OnClickListener() {
        public void onClick(View v) {
            new Thread(new LoadIconTask(R.drawable.painter, handler))
                .start();
        }
    });

    final Button otherButton = (Button) findViewById(R.id.otherButton);
    otherButton.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View v) {
            Toast.makeText(HandlerMessagesActivity.this, "I'm Working",
                Toast.LENGTH_SHORT).show();
        }
    });
}
```


THREADINGHANDLERMESSAGES

```
private class LoadIconTask implements Runnable {
    private final int resId;
    private final Handler handler;

    LoadIconTask(int resId, Handler handler) {
        this.resId = resId;
        this.handler = handler;
    }

    public void run() {

        Message msg = handler.obtainMessage(SET_PROGRESS_BAR_VISIBILITY,
            ProgressBar.VISIBLE);
        handler.sendMessage(msg);

        final Bitmap tmp = BitmapFactory.decodeResource(getResources(),
            resId);

        for (int i = 1; i < 11; i++) {
            sleep();
            msg = handler.obtainMessage(PROGRESS_UPDATE, i * 10);
            handler.sendMessage(msg);
        }

        msg = handler.obtainMessage(SET_BITMAP, tmp);
        handler.sendMessage(msg);

        msg = handler.obtainMessage(SET_PROGRESS_BAR_VISIBILITY,
            ProgressBar.INVISIBLE);
        handler.sendMessage(msg);
    }

    private void sleep() {
        try {
            Thread.sleep(mDelay);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

THREADINGHANDLERRUNNABLE

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    mImageView = (ImageView) findViewById(R.id.imageView);
    mProgressBar = (ProgressBar) findViewById(R.id.progressBar);

    final Button button = (Button) findViewById(R.id.loadButton);
    button.setOnClickListener(new OnClickListener() {
        public void onClick(View v) {
            new Thread(new LoadIconTask(R.drawable.painter)).start();
        }
    });

    final Button otherButton = (Button) findViewById(R.id.otherButton);
    otherButton.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View v) {
            Toast.makeText(HandlerRunnableActivity.this, "I'm Working",
                Toast.LENGTH_SHORT).show();
        }
    });
}
```

THREADINGHANDLERRUNNABLE

```
private class LoadIconTask implements Runnable {
    int resId;

    LoadIconTask(int resId) {
        this.resId = resId;
    }

    public void run() {

        handler.post(new Runnable() {
            @Override
            public void run() {
                mProgressBar.setVisibility(ProgressBar.VISIBLE);
            }
        });

        mBitmap = BitmapFactory.decodeResource(getResources(), resId);

        // Simulating long-running operation
        for (int i = 1; i < 11; i++) {
            sleep();
            final int step = i;
            handler.post(new Runnable() {
                @Override
                public void run() {
                    mProgressBar.setProgress(step * 10);
                }
            });
        }

        handler.post(new Runnable() {
            @Override
            public void run() {
                mImageView.setImageBitmap(mBitmap);
            }
        });

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

NEXT TIME

ALARMS