Android XI

Threads, Handlers, Loopers
**Threads**

- Processes versus threads
- UI (main) thread versus worker threads
- Basic creation of a thread:
  - Pass a runnable into a Thread constructor
  - Extend Thread and override run()

```java
new Thread(new Runnable() {
    public void run() {
        // a potentially // time consuming task
    }
}).start();

class SampleThread extends Thread {
    @Override
    public void run()
    {
        // logic to execute in a thread
    }
}
```
Background Threads and the UI

- Background threads can’t touch UI elements. Methods to help:
  - `Activity.runOnUiThread(Runnable)`
  - `View.post(Runnable)`
    - see code to left
  - `View.postDelayed(Runnable)`

```java
new Thread(new Runnable() {
    public void run() {
        // Need to update UI from this thread
        final Bitmap bitmap = processBitMap("image.png");
        mImageView.post(new Runnable() {
            public void run() {
                mImageView.setImageBitmap(bitmap);
            }
        });
    }
}).start();
```
**Handlers And Loopers**

- **Handler**: A class that allows you to send and process Messages and Runnables associated with a thread’s MessageQueue.
  - It has two main uses:
    - Schedule messages and runnables to be executed in the future.
    - Enqueue an action to be performed on another thread.
- **Looper**: A class that dispatches messages and runnables on a message queue.
- *A handler will have the looper for the thread in which it was created.*
When a runnable is posted to the queue, it runs when it is dispatched by the looper by calling the runnable’s run() method.
When a message is posted to the queue, the handler for the thread that has the queue has its `handleMessage(Message)` called by the looper.
Some handler methods for runnables

- From the sending thread, using a Handler from the receiving thread:
  - boolean post(Runnable r)
  - boolean postAtTime(Runnable r, long uptimeMillis)
  - boolean postDelayed(Runnable r, long delayMillis)

- You can also post from the UI thread to its own looper
  - The preferred way of having a delayed action in the UI
Posting Messages

- Implement `handleMessage(Message m)` in the handler for the receiving thread to respond to the message
  - Get the info from the handler and use it in your code
- In your other thread
  - Obtain the handler for the thread you want to send a message to:
    (e.g., make a constructor that takes a handler)
  - Put the info into the message and `handler.sendMessage(message)`