Programming
Handheld Systems

Adam Porter
Threads, AsyncTask & 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

app

thread

new

start()

run()
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
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)
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

    Can call void publishProgress(Progress... values)
AsyncTask

void onProgressUpdate (Progress... values)
   Invoked in response to publishProgress()

void onPostExecute (Result result)
   Runs after doInBackground()
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
Each Android Thread is associated with a MessageQueue & a Looper

A MessageQueue holds Messages and Runnables to be dispatched by the Looper
Add Runnables to MessageQueue by calling Handler’s post() method
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.*
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 at current time + delay
Next Time

Alarms