Section 1 Multithreading

Creating Threads

Threads allow a program to accomplish many tasks at the same time. This creates increased time efficiencies at the cost of added complexity and CPU usage. Being an object oriented language, java uses the *Thread* class to create and modify threads. In Java, each thread must be it's own class or subclass either implementing Runnable or extending Thread. The difference between using Runnable or Thread to create a thread is that Runnable allows more flexibility as you can only extend one class. Here are two examples that start a new thread to print the same message. The first implements Runnable and the second extends Thread.

```
public class HelloRunnable implements Runnable {
   public void run() {
      System.out.println("Hello from a thread!");
   }
   public static void main(String args[]) {
      (new Thread(new HelloRunnable())).start();
   }
}
public class HelloThread extends Thread {
   public void run() {
      System.out.println("Hello from a thread!");
   }
   public static void main(String args[]) {
      (new HelloThread()).start();
   }
}
```

Code courtesy of Oracle

Ending Threads

When threads finish or are no longer needed, they are either interrupted or joined. To end a thread using interrupts, the interrupt() method is used. To properly handle this exception on the end of the thread, the thread must either have a try-catch block containing the sleep method or an if statement with the parameter checking the state of *Thread.interupted()*. Regardless of the method used to interrupt, the thread must return, either nothing or something.

```
for (int i = 0; i < importantInfo.length; i++) {
   // Pause for 4 seconds
   try {
      Thread.sleep(4000);
   } catch (InterruptedException e) {
      // We've been interrupted: no more messages.
      return;
   }
   // Print a message
   System.out.println(importantInfo[i]);
}
```

or

```
for (int i = 0; i < inputs.length; i++) {
   heavyCrunch(inputs[i]);
   if (Thread.interrupted()) {
      // We've been interrupted: no more crunching.
      return;
   }
}
```

Code courtesy of Oracle

To end a thread by joining two threads, the thread calls the *join() method*. The method calling join waits until the method that it called join on finishes, then continues normal operation. By ending a thread using interrupts or join, the thread is ended as gracefully as possible, instead of just killing the thread while it is still running which can result in errors.

Synchronization and Threads:
Threads operate very efficiently due to their ability to work independently and utilize more system resources at a time. However, there are problems that can arise from independently working threads. Thread interference is when Threads attempt to use fields at the same time, resulting in unexpected operations. To prevent this, the synchronized keyword is used when writing methods that will be used by various threads. By declaring multiple methods of a class synchronized, only one thread is allowed to work with the methods at a time. This prevents thread interference due to multiple threads accessing the same method.

Resources: Oracle Tutorial on Threads
Java API