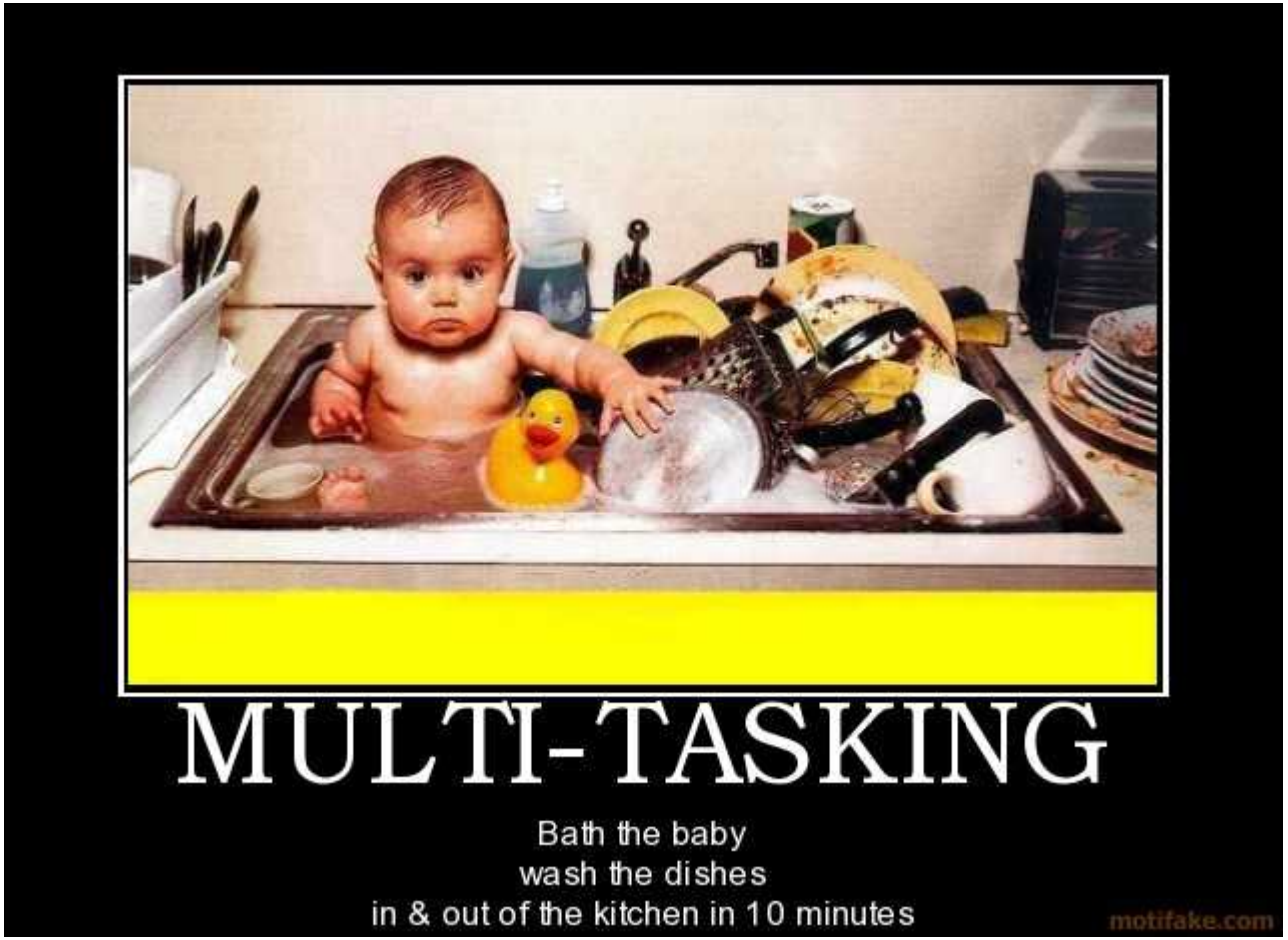
Multithreaded Programming



http://www.motifake.com/multi-tasking-baby-dishes-bath-wash-demotivational-posters-118837.html


take an umbrella they said
so you don't get wet they said

http://thechive.com/2014/01/15/champions-of-multitasking-35-photos/

Traditional Multi-tasking

- One CPU
- Many users, each wishing to use a computer
- Fast CPU (relatively speaking) so share it
  - Time slice
  - Works well for human interaction
  - Not so well for compute intensive applications
- Provided by the operating system

- Referred to as process based
- A program is the smallest unit of code that can be dispatched by the scheduler

Example of process based multitasking

- Running your word processor simultaneous with receiving email

- Printing a document from a word processor and simultaneously googling something

-

Multithreading

- a multithreaded program contains two or more parts that can run concurrently
- each part is called a thread
- each thread defines a separate path of execution

- Java handles threads

A thread can be in one of several states:

> running

> ready to run (as soon as it gets CPU time)

> suspended

- if suspended, can later be resumed

> blocked

- waiting for resources

> terminated

In Java, multithreading is built upon the Thread class
and
a companion interface: Runnable

Two ways to create a runnable object:

- implement the Runnable interface
- extend the Thread class

Both approaches use the Thread class to instantiate, access, and control the thread. The difference is how a thread-enabled class is created.

- The Runnable interface abstracts a unit of code

- A thread can be constructed on any object that implements the Runnable interface

- Runnable defines only one method called run( ) which is declared like this:

```
public void run( ) {
    // code that makes up the thread
    //   goes here
}
```

Here is what run( ) can do:

- call other methods
- use other classes
- declare variables

The difference, compared with a “regular” program is that run( ) establishes the entry point for a concurrent thread of execution within a program.

A thread ends when run( ) returns

Methods defined by the Thread class

Method	Meaning
final String getName( )	Obtains a thread's name.
final int getPriority( )	Obtains a thread's priority.
final boolean isAlive( )	Determines whether a thread is still running.
final void join( )	Waits for a thread to terminate.
void run( )	Entry point for the thread.
static void sleep(long *milliseconds*)	Suspends a thread for a specified period of milliseconds.
void start( )	Starts a thread by calling its **run( )** method.

1) create a class that implements Runnable
2) instantiate an object of type Thread on an object of that class
3) start the thread

```
class MyThread implements Runnable {
  String thrdName;

  MyThread(String name) {
    thrdName = name;
  }

  // Entry point of thread.
  public void run() {
    System.out.println(thrdName + " starting.");
    try {
      for(int count=0; count < 10; count++) {
        Thread.sleep(400);
        System.out.println("In " + thrdName +
                           ", count is " + count);
      }
    }
    catch(InterruptedException exc) {
      System.out.println(thrdName + " interrupted.");
    }
    System.out.println(thrdName + " terminating.");
  }
}
```

```
class MyThread implements Runnable {
  String thrdName;

  MyThread(String name) {
    thrdName = name;
  }
```

```
  // Entry point of thread.
   public void run() {
      System.out.println(thrdName + " starting.");
      try {
         for(int count=0; count < 10; count++) {
           Thread.sleep(400);
           System.out.println("In " + thrdName +
                              ", count is " + count);
         }
      }
      catch(InterruptedException exc) {
         System.out.println(thrdName + " interrupted.");
      }
      System.out.println(thrdName + " terminating.");

   }       (end of run( ) method, i.e. the thread)

}      (end of MyThread class)
```

```
class UseThreads {
  public static void main(String args[]) {
    System.out.println("Main thread starting.");

    // First, construct a MyThread object.
    MyThread mt = new MyThread("Child #1");

    // Next, construct a thread from that object.
    Thread newThrd = new Thread(mt);

    // Finally, start execution of the thread.
    newThrd.start();

    for(int i=0; i<50; i++) {
      System.out.print(".");
      try {
        Thread.sleep(100);
      }
      catch(InterruptedException exc) {
        System.out.println("Main thread interrupted.");
      }
    }

    System.out.println("Main thread ending.");
  }
}
```

```
class UseThreads {
  public static void main(String args[]) {
    System.out.println("Main thread starting.");

        // First, construct a MyThread object.
    MyThread mt = new MyThread("Child #1");

         // Next, construct a thread from that object.
    Thread newThrd = new Thread(mt);

        // Finally, start execution of the thread.
    newThrd.start();
```

```
    for(int i=0; i<50; i++) {
      System.out.print(".");
      try {
        Thread.sleep(100);
      }
      catch(InterruptedException exc) {
        System.out.println("Main thread interrupted.");
      }
    }

    System.out.println("Main thread ending.");
  }
}
```

```
Main thread starting.
.Child #1 starting.
...In Child #1, count is 0
....In Child #1, count is 1
....In Child #1, count is 2
....In Child #1, count is 3
....In Child #1, count is 4
....In Child #1, count is 5
....In Child #1, count is 6
....In Child #1, count is 7
....In Child #1, count is 8
....In Child #1, count is 9
Child #1 terminating.
..........Main thread ending.
```

```
// Improved MyThread.

class MyThread implements Runnable {
  Thread thrd;

  // Construct a new thread.
  MyThread(String name) {
    thrd = new Thread(this, name);
    thrd.start(); // start the thread
  }
```

```
  // Begin execution of new thread.
  public void run() {
    System.out.println(thrd.getName() + " starting.");
    try {
      for(int count=0; count<10; count++) {
        Thread.sleep(400);
        System.out.println("In " + thrd.getName() +
                           ", count is " + count);
      }
    }
    catch(InterruptedException exc) {
      System.out.println(thrd.getName() + "
interrupted.");
    }
    System.out.println(thrd.getName() + " terminating.");
  }
}
```

```
class UseThreadsImproved {
  public static void main(String args[]) {
    System.out.println("Main thread starting.");

    MyThread mt = new MyThread("Child #1");

    for(int i=0; i < 50; i++) {
      System.out.print(".");
      try {
        Thread.sleep(100);
      }
      catch(InterruptedException exc) {
        System.out.println("Main thread interrupted.");
      }
    }

    System.out.println("Main thread ending.");
  }
}
```

A way to determine if a thread has completed:

method isAlive( )

Another way is method join( )

```
class UseThreadsImproved {
  public static void main(String args[]) {
    System.out.println("Main thread starting.");

    MyThread mt = new MyThread("Child #1");

    do {
      System.out.print(".");
      try {
            Thread.sleep(100);
      }
      catch(InterruptedException exc) {
            System.out.println("Main thread interrupted.");
      }
    } while (mt.thrd.isAlive());

    System.out.println("Main thread ending.");
  }
}
```

Output when using isAlive

```
.Child #1 starting.
....In Child #1, count is 0
....In Child #1, count is 1
....In Child #1, count is 2
....In Child #1, count is 3
....In Child #1, count is 4
....In Child #1, count is 5
....In Child #1, count is 6
....In Child #1, count is 7
....In Child #1, count is 8
....In Child #1, count is 9
Child #1 terminating.
Main thread ending
```