Threads and Multithreading

Multiprocessing

- Modern operating systems are multiprocessing
- Appear to do more than one thing at a time
- Three general approaches:
  - Cooperative multiprocessing
  - Preemptive multiprocessing
  - Really having multiple processors

Multithreading

- Multithreading programs *appear* to do more than one thing at a time
- Same ideas as multiprocessing, but within a single program
- More efficient than multiprocessing
- Java tries to hide the underlying multiprocessing implementation

Why multithreading?

- Allows you to do more than one thing at once
  - Play music on your computer's CD player,
  - Download several files in the background,
  - while you are writing a letter
- Multithreading is essential for animation
  - One thread does the animation
  - Another thread responds to user inputs

Threads

- A Thread is a single flow of control
  - When you step through a program, you are following a Thread
- Your previous programs all had one Thread
- A Thread is an Object you can create and control

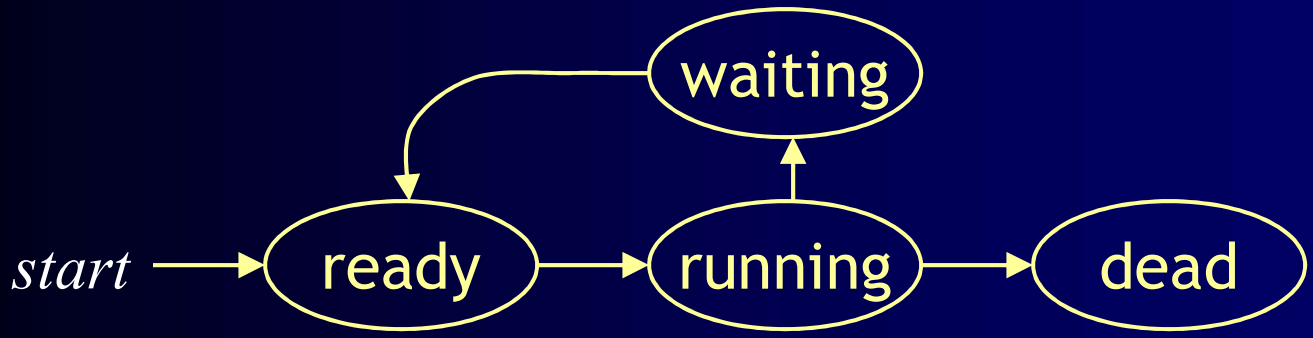
Sleeping

- Every program uses at least one Thread
- Thread.sleep(int milliseconds);
- try { Thread.sleep(1000); }
  catch (InterruptedException e) { }
- sleep only works for the current Thread

States of a Thread

- A Thread can be in one of four states:
  - **Ready:** all set to run
  - **Running:** actually doing something
  - **Waiting,** or **blocked:** needs something
  - **Dead:** will never do anything again
- State names vary across textbooks
- You have some control, but the Java scheduler has more

State transitions



Two ways of creating Threads

- You can extend the Thread class:
  - class Animation extends Thread {…}
  - Limiting, since you can only extend one class
- Or you can implement the Runnable interface:
  - class Animation implements Runnable {…}
  - requires public void run( )
- I recommend the second for most programs

Extending Thread

- class Animation extends Thread {
  public void run( ) { *code for this thread* }
  Anything else you want in this class
  }
- Animation anim = new Animation( );
  - A newly created Thread is in the **Ready** state
- To start the anim Thread running, call anim.start( );
- start( ) is a *request* to the scheduler to run the Thread --it may not happen right away
- The Thread should eventually enter the **Running** state

Implementing Runnable

- class Animation implements Runnable {…}
- The Runnable interface requires run( )
  - This is the "main" method of your new Thread
- Animation anim = new Animation( );
- Thread myThread = new Thread(anim);
- To start the Thread running, call myThread.start( );
  - You do not write the start() method—it's provided by Java
- As always, start( ) is a *request* to the scheduler to run the Thread--it may not happen right away

Starting a Thread

- Every Thread has a start( ) method
- *Do not* write or override start( )
- You *call* start( ) to request a Thread to run
- The scheduler then (eventually) calls run( )
- You must supply public void run( )
  - This is where you put the code that the Thread is going to run

Extending Thread: summary

```
class Animation extends Thread {
    public void run( ) {
        while (okToRun) { ... }
    }
}

Animation anim = new Animation( );
anim.start( );
```

Implementing Runnable: summary

```
class Animation extends Applet
                    implements Runnable {
    public void run( ) {
        while (okToRun) { ... }
    }
}

Animation anim = new Animation( );
Thread myThread = new Thread(anim);
myThread.start( );
```

Demo

Example I

```
class MyThread extends Thread {
        private String name, msg;
        public MyThread(String name, String msg) {
        this.name = name;
        this.msg = msg;
}
public void run() {
        System.out.println(name + " starts its execution");
        for (int i = 0; i < 5; i++) {
                System.out.println(name + " says: " + msg);
                try {
                        Thread.sleep(5000);
                } catch (InterruptedException ie) {}
        }
        System.out.println(name + " finished execution");
        }
}
```

Example I

```
class MyThread extends Thread {
        private String name, msg;
        public MyThread(String name, String msg) {
        this.name = name;
        this.msg = msg;
}
public void run() {
        System.out.println(name + " starts its execution");
        for (int i = 0; i < 5; i++) {
                System.out.println(name + " says: " + msg);
                try {
                        Thread.sleep(5000);
                } catch (InterruptedException ie) {}
        }
        System.out.println(name + " finished execution");
        }
}
```

Example I

```
class MyThread extends Thread {
        private String name, msg;
        public MyThread(String name, String msg) {
        this.name = name;
        this.msg = msg;
}
public void run() {
        System.out.println(name + " starts its execution");
        for (int i = 0; i < 5; i++) {
                System.out.println(name + " says: " + msg);
                try {
                        Thread.sleep(5000);
                } catch (InterruptedException ie) {}
        }
        System.out.println(name + " finished execution");
        }
}
```

Example I

```
public class test {
    public static void main(String[] args) {
        MyThread mt1 = new MyThread("thread1", "ping");
        MyThread mt2 = new MyThread("thread2", "pong");
        mt1.start();
        mt2.start();
    }
}
```

Example II

```
class MyThread implements Runnable {
        private String name, msg;
        public MyThread(String name, String msg) {
        this.name = name;
        this.msg = msg;
}
public void run() {
        System.out.println(name + " starts its execution");
        for (int i = 0; i < 5; i++) {
                System.out.println(name + " says: " + msg);
                try {
                        Thread.sleep(5000);
                } catch (InterruptedException ie) {}
        }
        System.out.println(name + " finished execution");
        }
}
```

Example II

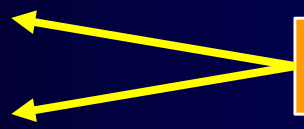
```
class MyThread implements Runnable {
        private String name, msg;
        public MyThread(String name, String msg) {
        this.name = name;
        this.msg = msg;
}
public void run() {
        System.out.println(name + " starts its execution");
        for (int i = 0; i < 5; i++) {
                System.out.println(name + " says: " + msg);
                try {
                        Thread.sleep(5000);
                } catch (InterruptedException ie) {}
        }
        System.out.println(name + " finished execution");
        }
}
```

Example II

```
class MyThread implements Runnable {
        private String name, msg;
        public MyThread(String name, String msg) {
        this.name = name;
        this.msg = msg;
}
public void run() {
        System.out.println(name + " starts its execution");
        for (int i = 0; i < 5; i++) {
                System.out.println(name + " says: " + msg);
                try {
                        Thread.sleep(5000);
                } catch (InterruptedException ie) {}
        }
        System.out.println(name + " finished execution");
        }
}
```

Example II

```
public class test {
    public static void main(String[] args) {
        MyThread mt1 = new MyThread("thread1", "ping");
        MyThread mt2 = new MyThread("thread2", "pong");
        Thread T1 = new Thread(mt1);
        Thread T2 = new Thread(mt2);
        T1.start();
        T2.start();
    }
}
```



These Two Threads will run in parallel

Typical output of the previous examples:

```
thread1 starts its execution
thread1 says: ping
thread2 starts its execution
thread2 says: pong
thread1 says: ping
thread2 says: pong
thread1 says: ping
thread2 says: pong
thread1 says: ping
thread2 says: pong
thread1 says: ping
thread2 says: pong
thread1 says: ping
thread2 says: pong
thread1 finished execution
thread2 finished execution
```