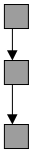
J.55

THREADS

Most programming languages are designed for single processor, sequential execution.

In Java, this corresponds to a single thread.



But Java can also handle multiple threads executing concurrently (sharing the processor).

J.56

- You create a Thread using new

  Thread myThread = new Thread();

- After creation, you can configure it using methods such as

  setName(String name)
  setPriority(int Priority)

- To run it, you invoke its start method, which spawns a new thread based on the data in the Thread object.

- Once it is started, the Java virtual machine invokes its run method, which you write.

- A Thread can be explicitly stopped with its stop method.

Example: J.57

```
class PingPong extends Thread {
    String word;
    int delay;

    PingPong(String whatToSay, int delayTime) {
      word = whatToSay;
      delay = delayTime;
    }

    public void run() {
      try {
        for (;;) {
          System.out.print(word + " ");
          sleep(delay);  // sleep CAN throw an exception
        }
      }  catch (InterruptedException e) {
          return;
      }
    }
    public static void main(String[] args) {
      new PingPong("ping", 33).start();       // 1/30 sec
      new PingPong("PONG", 100).start(); // 1/10 sec
      new PingPong("slam", 200).start();    // 2/10 sec
    }}
```

```
ping PONG slam ping ping ping PONG ping ping
ping PONG slam ping ping ping PONG ping ping
ping PONG slam . . .
```

J.58

Things you can do with Threads

1. Create them, run them, stop them.
2. Assign them priorities for preference in running.
3. Suspend them and resume them.
4. Use them to execute synchronized methods or synchronized blocks of code for working with shared data.
5. Use the explicit methods wait and notify to have some threads wait for results to be produced by others.

J.59

RUNNABLES

- The Runnable interface abstracts the concept of something that will execute code while it is active.

- The Runnable interface declares a single method:

  public abstract void run()

- The Thread class implements the Runnable interface, but it has a lot of extra overhead, so it is often easier to just implement Runnable directly.

- If an object implementing Runnable is used to create a thread, then starting the thread will lead to the invocation of the object's run method in that separately executing thread.

J.60

EXAMPLE: Graphics / Animation

```
import java.awt.Graphics;
import java.awt.Color;

public class check3 extends java.applet.Applet
  implements Runnable {

  /* This applet uses the concept of a thread to enable
     animation on its own without interfering with other
     system operations.

     It can be started and stopped and can run in parallel
     with other applets. */

  Thread runner;
  int xpos;
```

J.61

```
/* The start method creates a new thread and starts
   it running */

 public void start() {
   if (runner == null) {
      runner = new Thread(this);
      runner.start();
   }
 }

 /* The stop method suspends the execution of the
   thread when the reader leaves the page. */

 public void stop() {
   if (runner != null) {
      runner.stop();
      runner = null;
   }
 }
```

J.62

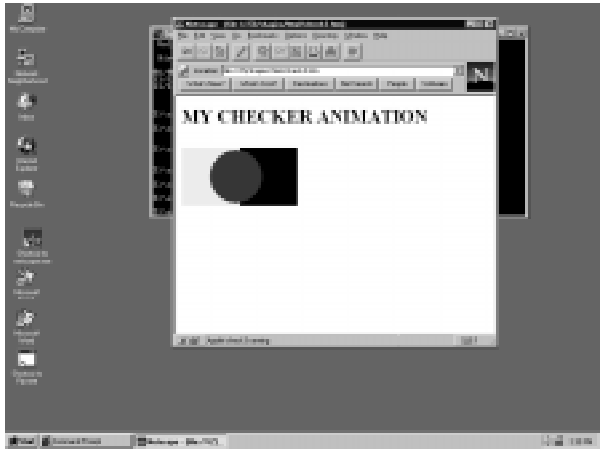
```
/* The run method changes the value of xpos AND
   repaints the screen after a pause of 100 milliseconds */

 public void run() {

 /* Moving xpos from left to right */

   while (true) {
     for (xpos = 5; xpos <=105; xpos +=4)
       {
       repaint();
       try {Thread.sleep(100); }
       catch (InterruptedException e) {}
       }
 /* Moving xpos from right to left */

     for (xpos = 105; xpos > 5; xpos -=4)
       {
       repaint();
       try {Thread.sleep(100); }
       catch (InterruptedException e) {}
     }
   }
 }
```

Why does run have to go to sleep ?

J.63

```
/* The paint method specifies what to draw
      on the screen */

   public void paint(Graphics g) {

      /* Draw board squares */

      g.setColor(Color.yellow);
      g.fillRect(0,0,100,100);
      g.setColor(Color.black);
      g.fillRect(100,0,100,100);


      /* Draw checker */

      g.setColor(Color.red);
      g.fillOval(xpos,5,90,90);

   }
}
```

J.64



This will have some flicker. There are methods for dealing with it such as

1. Repainting only the changed part of the screen.

2. Double buffering

J.65

Making the figure and motion more complex

```
/*
TITLE:     MOREMAN.JAVA
AUTHOR:    LINDA SHAPIRO
DATE:      NOVEMBER 17, 1997
PURPOSE:   TO DEMONSTRATE THE USE OF SINE
        AND COSINE IN AN ANIMATION
*/



import java.awt.Graphics;
import java.awt.Color;

public class moreman extends java.applet.Applet
  implements Runnable {

  Thread runner;
  int xpos,ypos,ypos2;
  double dx, dy, dy2;
```

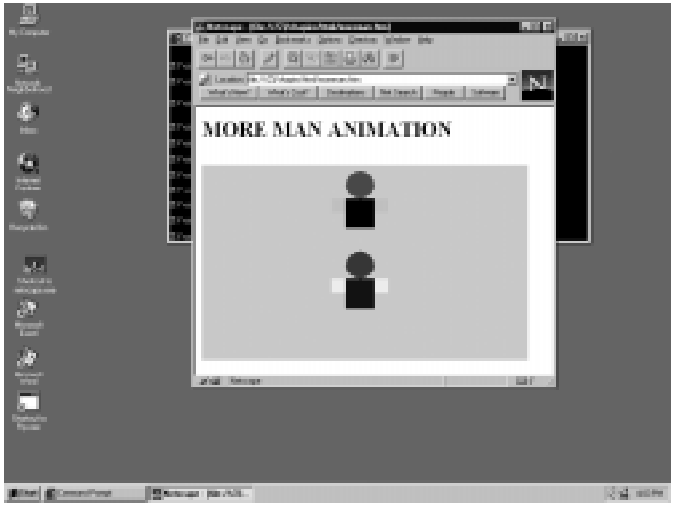
J.66

```
/* CODE TO CONTROL THE MOVEMENTS  */

  public void run() {
    setBackground(Color.cyan);
    while (true) {

      /* FORWARD LOOP. */

        for (xpos = 5; xpos <=405; xpos +=4) {
        dx = (double) (xpos * .0078);
        dy = Math.sin(dx)/ .0078;
        ypos = (int) dy;
        dy2 = Math.cos(dx)/ .0078;
        ypos2 = (int) Math.abs(dy2);
        repaint();
        try {Thread.sleep(100); }
        catch (InterruptedException e) {}
      }
      /* REVERSE LOOP. */

      for (xpos = 405; xpos > 5; xpos -=4) {
        dx = (double) (xpos * .0078);
        dy = Math.sin(dx)/ .0078;
        ypos = (int) dy;
        dy2 = Math.cos(dx)/ .0078;
        ypos2 = (int) Math.abs(dy2);
        repaint();
        try {Thread.sleep(100); }
        catch (InterruptedException e) {}
      }}}
```

J.67

```
public void paint(Graphics g) {

  /* THIS MAN USES THE SINE FUNCTION */

  g.setColor(Color.red);
  g.fillOval(xpos+23,ypos+5,45,45);
  g.setColor(Color.blue);
  g.fillRect(xpos+23,ypos+50,45,45);
  g.setColor(Color.yellow);
  g.fillRect(xpos+2,ypos+50,21,21);
  g.fillRect(xpos+68,ypos+50,21,21);

  /* THIS MAN USES THE COSINE FUNCTION */

  g.setColor(Color.magenta);
  g.fillOval(xpos+23,ypos2+5,45,45);
  g.setColor(Color.black);
  g.fillRect(xpos+23,ypos2+50,45,45);
  g.setColor(Color.pink);
  g.fillRect(xpos+2,ypos2+50,21,21);
  g.fillRect(xpos+68,ypos2+50,21,21);

}
}
```

J.68



The Moving Dog Image Example

J.69

```
import java.awt.Graphics;
  import java.awt.Image;
  import java.awt.Color;

  public class MoveDog extends java.applet.Applet
    implements Runnable {

    Image dogpics[] = new Image[4];
    Image currentimg;
    Thread runner;
    int xpos;
    int ypos = 50;

  public void init() {

  String dogsrc[] = {"right1.gif", "right2.gif", "stop.gif",
                     "yawn.gif"};

    for (int i=0; i < dogpics.length; i++) {
      dogpics[i] = getImage(getCodeBase(),"images/" +
                     dogsrc[i]);

    }
  }
```

J.70

```
public void run() {

    setBackground(Color.white);
    while (true) {

 /* run from one side of the screen to the middle */
   dogrun(0, this.size().width / 2);

 /* stop and pause  */
    currentimg = dogpics[2];
    repaint();
    pause(1000);

 /* yawn  */
    currentimg = dogpics[3];
    repaint();
    pause(1000);

 /* go back to plain old stop  */
    currentimg = dogpics[2];
    repaint();
    pause(1000);

 /* wake up and run off */

    dogrun(xpos, this.size().width + 10);  }  }
```

J.71

```
void dogrun(int start, int end) {
  for (int i = start; i < end; i+=10) {
      this.xpos = i;

  /*  swap images  */

   if (currentimg == dogpics[0]) currentimg = dogpics[1];
       else currentimg = dogpics[0];

   repaint();
   pause(150);
  }
}

void pause(int time) {
  try { Thread.sleep(time); }
  catch (InterruptedException e) { }
}

public void paint(Graphics g) {
  g.drawImage(currentimg, xpos, ypos, this);
  }
}
```

J.72