

Section 15: Threads

1) Multi-threading

doing multiple things at once

2) Thread

one activity

3) threads:

- a) making a class respond to a thread: class name extends Thread {
- b) putting a thread to sleep

```
try {
    Thread.sleep(milliseconds);
}
catch (InterruptedException e) {
    System.err.println("sleep exception");
}
```

- c) to start a treaded object: name.start();
- d) treaded objects need a user specified public void run() function, run() sleeps with a call to Thread.sleep
- e) when a threaded object is allsleep, the class that uses that object can respond to actions, including those that call functions in the threaded object (for example, those that would put change the conditions of that object)
- f) a thread dies when run() is exited
- g) a threaded object can be made to wait until another thread has died by executed join() [example: threadedObject.join() makes the thread that called threadedObject.join() wait until the thread in treadedObject has died
- h) to find out the state of a thread in a threaded object: threadedObject.isAlive() returns t/f

example:

```
import java.awt.*;
import java.applet.Applet;
import java.awt.event.*;

public class chp25 extends Applet implements ActionListener {

    private Button start, stop;
    private Roller ball;

    public void init() {
        start = new Button("Start");
        add(start);
        start.addActionListener(this);

        stop = new Button("Stop");
        add(stop);
        stop.addActionListener(this);
    }
}
```

```
        public void paint(Graphics g) {
            setBackground(Color.white);
        }

        public void actionPerformed(ActionEvent event) {
            if (event.getSource() == start) {
                Graphics g = getGraphics();
                ball = new Roller(g);
                ball.start();
            }
            if (event.getSource() == stop)
                ball.StopRolling();
        }
    }

import java.awt.*;

public class Roller extends Thread {

    private boolean rollOn;
    private Graphics g;
    private int x=100, step=5;
    private int size = 20;
    private final int MAX = 150;
    private final int MIN = 50;
    private final int Y = 100;
    private boolean right = false;

    public Roller(Graphics graphics) {
        g = graphics;
        rollOn = true;
    }

    public void StopRolling() {
        rollOn = false;
    }

    public void run() {
        g.setColor(Color.black);
        g.drawLine(MIN,Y-25,MIN,Y+25);
        g.drawLine(MAX+size,Y-25,MAX+size,Y+25);

        while (rollOn) {

            g.setColor(Color.white);
            g.fillOval(x,Y,size,size);

            if (right) {
                if (x+step < MAX)
                    x += step;
            }
            else
                right = false;
        }
    }
}
```

```
    }
    else
    {
        if (x-step > MIN)
            x -= step;
        else
            right = true;
    }

    g.setColor(Color.red);
    g.fillOval(x,Y,size,size);

    try {
        Thread.sleep(50);
    }
    catch (InterruptedException e) {
        System.err.println("sleep exception");
    }
}
}
```

4) priority of threads:

- a) all threads by default have same priority
- b) a thread may let other threads run by invoking `yield()`:

5) mutual exclusion and synchronizing

- a) when multithreading and two or more threads use same resource, declare methods to be shared evently with “synchronized”

example: `public synchronized void MyMethod();`

6) thread interaction:

- a) `wait()` - when called, puts thread to sleep
- b) `notify()` - when called, releases another thread that is `wait()`; if none, does nothing

example: `wait()`

```
try {
    wait();
}
catch (InterruptedException e) {
    System.err.println("Exception");
}
```

example: `notify()`

```
notify();
```

7) invoking interrupt:

- a) one thread accept an interrupt from another: `threadObject.interrupt()`
 - i) works only if another object is in wait or sleep
- b) a tread can check to see if it has been interrupted: example:

```
while (!interrupted()) {
}
```