 Threads

- What if our Java program must do a lot of computation on the sensor inputs?
  - And listen for sensor input changes when they occur, possibly from many sensors
  - And respond to user requests through the Swing GUI, such as button clicks or other input
- How can it do multiple things “at the same time”?
  - If it’s computing, will it delay processing or lose sensor events, or Swing events?
- Java (and other) programs run in their own processes, fully isolated from each other
  - Two programs running on your laptop don’t interact
- We need to run “mini-programs” within the same program, but they must share data and logic
  - Threads are mini-processes within a process

Reading for next time: Big Java 20.4
Threads

- Most computers have multiple cores or processors that execute more than one piece of code or "thread" at a time
  - Even if they only have a single core/processor, they can implement threads by rapidly switching between tasks, giving each thread a small execution slice before switching to another task. All threads appear to proceed in parallel.

- Threads are different from processes
  - Processes are expensive but safe. Processes are so well insulated from each other that it is both complex and expensive to communicate between them.
  - Threads are cheap, but different threads running in the same process are not well-insulated from each other.

Processes

Process –

- Each program on your computer runs in its own process.
- Operating system’s way of ensuring one program uses its own memory, and each program has a reasonable amount of processing time.
- Safe - the separation of memory spaces allow one program to continue running even if another program writes garbage into its own memory space (and crashes). It also provides a level of security.

|------|------------------|-------------------|------------------|-------------------|

(Single processor example. If you have two, two processes can run simultaneously)

Image by MIT OpenCourseWare.
Threads

Threads –

- If a process is an operating system’s way of interleaving programs...
- A thread is a program’s way of interleaving sections of code.

Writing a Thread, Option 1

- Inherit from the Thread class and override its method `public void run();`:

  ```java
  public class SensorManager extends Thread {
      public void run() {
          // Code executed in the Thread goes here
          // run() is just like main(): Thread starts here
      }
  }
  ```

- In `main()` or other method, create a Thread instance and start it:

  ```java
  Thread t = new SensorManager();
  t.start();
  ```
Writing a Thread, Option 2

• If the object you want to place in a thread already inherits from a superclass, use an interface

```
public interface Runnable {
    public void run();
}
```

• Modify the class to implement Runnable, e.g.,

```
public class SensorManager extends JFrame {
    implements Runnable {
        // constructors and other methods go here
        public void run() {
            // code executed in the Thread goes here
        }
    }
}
```

• In main() or other method:

```
Thread t = new Thread(new SensorManager());
t.start();
```

Find Status of Thread

• In main() or other method:

```
// Ask if thread is executing
if (t.isAlive())
    // code (can't use t's results yet)
else
    // code (assumes t has completed)
```

• You can also wait for a thread to finish:

```
// In a method of some thread other than t
t.join(); // waits until t completes
// code (assumes t has completed)
```

• To stop a thread, use:

```
t.interrupt();
// Thread t catches InterruptedException, returns
// We don't cover this in 1.00.
```
Computing with threads

• Program finds sum of set of numerical integrals of cylinder volumes
  – We cover numerical integration in lecture 32
  – In this lecture, we treat it as just an expensive computation
  – We scale the output, so we always get approximately π

• Reads radius of each cylinder
  – Must respond to input in timely fashion, even if integral takes a long time to compute
  – If program doesn’t respond quickly, either the user will find the application unusable, or input events may be dropped
    • Maximum event queues are often short
  – We use Scanner and System.in for keyboard input
    • JOptionPane is a Swing object in its own thread, which would complicate the example
  – We use floats, not doubles, to simplify one ugly issue
    • We discuss it (synchronization) in the next lecture

Cylinder integral without threads

```java
import java.util.*;

public class Cylinder {
    private static final int ITER = 20000000;
    private float radius;

    public Cylinder(float r) { radius = r; }

    public float circularIntegral() {
        float sum = 0.0F;
        for (int i = 0; i < ITER; i++) {
            // Math.random() returns double d: 0 <= d <= 1
            float x = 2*radius*(float)Math.random() - radius;
            float y = 2*radius*(float)Math.random() - radius;
            float f = 1.0F;  // f(x,y)—constant here
            if ((x*x + y*y) < radius*radius) // If in region
                sum += f;   // Increment integral sum
        }
        System.out.println("r " + radius + " i " + 4.0F*sum/ITER);
        return 4.0F*sum/ITER; // Integral value * 4 (pi)
    }
}
```
public static void main(String[] args) {
    Scanner in= new Scanner(System.in);    // Keyboard input
    float integral= 0.0;
    for (int i= 0; i < 6; i++) {
        System.out.println("Enter radius ");
        float radius= in.nextFloat(); // Keyboard input
        Cylinder t = new Cylinder(radius);
        integral += t.circularIntegral();
    }
    System.out.println("integral " + integral);
    System.out.println("Done");
}

// Run this to see that it responds slowly. Enter r=1 each time
// Eclipse is flaky in not putting the cursor at end of prompt.

Exercise 1: Cylinder with threads

• CylinderThread uses threads
  – It splits the integration work across 6 threads
  – The threads each contribute their part of the integral to a static variable in CylinderThread
  – In main()
    • Each thread is created with a new keyword
    • Each thread is then started with start()
    • Each thread starts by executing its run() method
    • Which calls method circularIntegral()
    • Which increments a static variable that holds the sum
  – We enter the radius for each of the 6 parts
    • This could also come from a sensor or Swing event
  – We will complete CylinderThread
Exercise 1a: Cylinder with threads

```java
import java.util.*;

public class CylinderThread { // Complete this line
    private static final int ITER = 20000000;
    private float radius;
    private static float integral; // New; holds sum
    public CylinderThread(float r) { super(); radius= r; } // Diff

    public float circularIntegral() {
        float sum= 0.0F;
        for (int i= 0; i < ITER; i++) {
            float x = 2*radius*(float)Math.random() - radius;
            float y = 2*radius*(float)Math.random() - radius;
            float f = 1.0F; // f(x,y)—constant here
            if ((x*x + y*y) < radius*radius) // If in region
                sum += f; // Increment integral sum
        }
        System.out.println("r " + radius + " i " + 4.0F*sum/ITER);
        integral += 4.0*sum/ITER; // New, adds to sum
        return 4.0F*sum/ITER; // Integral value * 4 (pi)
    }
}
```

Exercise 1b: Cylinder with threads

```java
// Write the run() method. It's like a main() method.

public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    for (int i = 0; i < 6; i++) {
        System.out.println("Enter radius ");
        float radius = in.nextFloat();
        CylinderThread t = new CylinderThread(radius); // New
        t.start(); // Uncomment this
    }
    System.out.println("integral: " + CylinderThread.integral);
    System.out.println("Done");
}
```

// When you are done, compile and run this.
// It prompts for keyboard input quickly, unlike Cylinder
// But... it does not give the correct answer.
What happens?

Exercise 2

- Correct CylinderThread, which has an error:
  - It splits the integration work across 6 threads
  - The threads each contribute their part of the integral to a static variable in CylinderThread
  - However, the threads may not have all completed when main() outputs the answer
- Use isAlive() or join() to wait until all parts of the integral have been computed
  - You will need to keep track of all the thread objects you created, perhaps in an array or ArrayList
  - join() and isAlive() throw InterruptedException
  - You must put calls to them in a try-catch block
- Your changes are all in main()
### Interface Runnable

- Suppose Cylinder extends another class
  - It still contains the integration method that takes a long time to compute
  - So we still want to put each computation in a separate thread
- **We use the following class definition:**
  ```java
  public class Cylinder2 extends EngrComponent implements Runnable {
    // Instead of: extends Thread
  }
  ```
- **We use the following constructor:**
  ```java
  Thread t = new Thread(new Cylinder2(radius));
  // Instead of:
  CylinderThread2 t = new CylinderThread2(radius);
  ```

### Exercise 3

- **Modify the solution to exercise 2 to implement Runnable instead of extending Thread. The changes are minor:**
  - Copy and rename the exercise 2 solution to a different class name (e.g., CylinderRunnable)
  - Inherit from EngrComponent and Runnable
    - Change class definition, as on previous slide
    - Modify constructor to also call superclass constructor
  - main(): change use of constructor, as on previous slide

```java
public class EngrComponent {
    // In download
    private int ID;
    private static int nextID= 0;
    public EngrComponent() {ID= nextID++; }
    public final int getID() { return ID;
    }
```