CS 470/570 Fall 2017

Introduction to Java Threads
Java Threads

• Compute Pi with Monte Carlo Method
• Compute Pi with Numerical Integration
• Producer/Consumer pattern
Monte Carlo Methods

- A broad category of methods that can be used to simulate a process or estimate some value
- Monte Carlo simulations use a random number generator and some probability distribution
Approximate the Value of Pi

Unit Circle (circle with radius 1) Inside a Square

The Area inside the Circle is

\[ \pi \cdot r^2 = \pi \cdot 1^2 = \pi \]
Approximate the value of Pi

• The area of the circle is $\pi$
• The area of the square is 4
• Area of the Circle / Area of the Square = $\pi / 4$
• $\pi = 4 \times (\text{Area of the Circle} / \text{Area of the Square})$
Approximate the Pi

• Assume random is a function that returns a random number between 0 and 1

count = 0
repeat n times
  x = random()
  y = random()
  if x^2+y^2 <= 1 then count = count + 1 //i.e. the point is in the
  //unit circle

pi = 4.0*(count/n)
public class Pi { 
    int iterations;

    public Pi(int iter) {
        iterations = iter;
    }

    public double calculate() {
        int count = 0;
        for (int i = 0; i < iterations; i++) {
            double x = Math.random();
            double y = Math.random();
            if (x*x + y*y <= 1)
                count++;
        }
        return (4.0 *count)/iterations;
    }
}
public static void main(String args[]) {
    int iter = new Integer(args[0]);
    Pi p = new Pi(iter);
    System.out.println("Pi is approximately "+p.calculate());
}


import java.util.concurrent.*;

public class P_Pi extends Thread {
    int iterations;
    int count;

    public P_Pi(int iter) {
        iterations = iter;
    }

    public int getCount() {
        return count;
    }
}

Compute PI (Monte Carlo)
public void run() {
    count = 0;
    for (int i = 0; i < iterations; i++) {
        double x = ThreadLocalRandom.current().nextDouble(1.0);
        double y = ThreadLocalRandom.current().nextDouble(1.0);
        if (x*x + y*y <= 1)
            count++;
    }
}
public static void main(String args[]) throws InterruptedException {
    int iterPerThread = new Integer(args[0]);
    int numThreads = new Integer(args[1]);
    P_Pi counts[] = new P_Pi[numThreads];
    int count = 0;

    for (int i = 0; i < numThreads; i++) {
        counts[i] = new P_Pi(iterPerThread);
        counts[i].start();
    }

    for (int i = 0; i < numThreads; i++) {
        counts[i].join();
        count = count + counts[i].getCount();
    }

    System.out.println("Pi is approximately " + (4.0*count)/(numThreads*iterPerThread));
}
Compute Pi (Numerical Integration)

• Find the area under the curve for the function \( \frac{4}{1 + x^2} \) between 0 and 1
• An example of numerical integration
• The algorithm does the following:
  – Break the area under the curve into rectangles
  – Calculate the area of each rectangle
    • Area = width\( \times \left( \frac{4}{1 + x^2} \right) \)
  – Sum the areas of the rectangles
Compute Pi
public class ComputePi extends Thread {
    int start;
    int stop;
    double width;
    double sum;
    double result;

    public ComputePi(int start, int stop, double width) {
        this.start = start;
        this.stop = stop;
        this.width = width;
    }
}
public void run() {
    double x;
    sum = 0.0;
    for (int i = start; i < stop; i++) {
        x = (i+0.5) * width;
        sum = sum + 4.0/(1.0+x*x);
    }
    result = sum * width;
}

public double result() {
    return result;
}
public static void main(String args[]) {
    int numThreads = Integer.parseInt(args[0]);
    int numIntervals = Integer.parseInt(args[1]);
    int localIntervals = numIntervals/numThreads;
    double width = 1.0/numIntervals;
    double result = 0.0;

    ComputePi results[];
    results = new ComputePi[numThreads];
for (int i = 0; i < numThreads; i++) {
    int start = i*localIntervals;
    int stop = start+localIntervals;
    results[i] = new ComputePi(start, stop, width);
}

for (int i = 0; i < numThreads; i++)
    results[i].start();
ComputePi

for (int i = 0; i < numThreads; i++)
    try {
        results[i].join();
    }
    catch (InterruptedException e) {}

for (int i = 0; i < numThreads; i++)
    result = result + results[i].result();

    System.out.println("pi = " + result);
}
Producer/Consumer Thread Example

```java
public class ProducerConsumer {
    public static void main(String[] args) {
        Buffer sharedVal = new Buffer();
        int start = new Integer(args[0]);
        int finish = new Integer(args[1]);
        int quit = new Integer(args[2]);
        Producer prod = new Producer(sharedVal, start, finish, quit);
        Consumer con = new Consumer(sharedVal, quit);
    }
}
```
Producer/Consumer Thread Example (part 2)

prod.start();
con.start();

try {
    prod.join();
    con.join();
} catch(InterruptedException e) {} 

System.out.println(con.sum());
}
class Consumer extends Thread {
    private Buffer sharedVal;
    int quit;
    int sum;

    public Consumer(Buffer b, int q) {
        sharedVal = b;
        quit = q;
    }
}
Producer/Consumer Thread Example (part 4)

```java
class ProducerConsumerThreadExample {
    public void run() {
        sum = 0;
        int i = sharedVal.delete();
        while (i != quit) {
            sum = sum + i;
            i = sharedVal.delete();
        }
    }

    public int sum() {
        return sum;
    }
}
```
class Producer extends Thread {
    private Buffer sharedVal;
    private int start;
    private int finish;
    private int quit;

    public Producer(Buffer b, int s, int f, int q) {
        sharedVal = b;
        start = s;
        finish = f;
        quit = q;
    }
}
public void run() {
    int i;
    for (i = start; i <= finish; i++)
        sharedVal.insert(i);
    sharedVal.insert(quit);
}
}
Monitor

• Synchronized
• Object methods
  – Wait
  – Notify
  – NotifyAll
• Limitations
class Buffer {
    private int contents;
    private boolean available = false;
    public synchronized int delete() {
        while (available == false) {
            try {
                wait();
            } catch (InterruptedException e) {
            }
        }
        available = false;
        notifyAll();
        return contents;
    }
}
public synchronized void insert(int value) {
    while (available == true) {
        try {
            wait();
        } catch (InterruptedException e) { }
    }
    contents = value;
    available = true;
    notifyAll();
}
}
Java Runnable Interface

• A class can implement the interface Runnable
• Run
• Why?
  – Single inheritance
Producer/Consumer Runnable Example (part 1)

public class ProducerConsumer {
    public static void main(String[] args) {
        Buffer sharedVal = new Buffer();
        int start = new Integer(args[0]);
        int finish = new Integer(args[1]);
        int quit = new Integer(args[2]);
        Producer prod = new Producer(sharedVal, start, finish, quit);
        Consumer con = new Consumer(sharedVal, quit);

        Thread t1 = new Thread(prod);
        Thread t2 = new Thread(con);
    }
}
Producer/Consumer Runnable Example (part 2)

t1.start();
t2.start();

try {
    t1.join();
    t2.join();
} catch(InterruptedException e) {}

System.out.println(con.sum());
Producer/Consumer Runnable Example (part 3)

class Consumer implements Runnable {
    //CONTENTS SAME AS THREAD EXAMPLE
}

class Producer implements Runnable {
    //CONTENTS SAME AS THREAD EXAMPLE
}

class Buffer {
    //CONTENTS SAME AS THREAD EXAMPLE
}