CS370 – HW5 Help Session

Spring 2016
Assignment Review

• Requirements
• Threads In Java
• Synchronized Block
• Do’s and Don’ts
• References
Assignment Review

• Bounded Buffer
  - FIFO Circular Buffer
  - Buffer Should hold maximum of 1000 items at a time
• Producer
  - The item added by producer to the buffer is a random Double
  - Producer must wait if buffer is full
• Consumer
  - Consumer is responsible for consuming elements from the buffer
  - If buffer is empty, Consumer must wait
Assignment Review

• The Producer and Consumer must refer to the same buffer
• Producer should produce a total of 1,000,000 items
• Bounded buffer should hold at most 1000 items at a time
• Consumer should consume all the 1,000,000 items
• To synchronize access to the buffer, use wait() and notify()
Threads In Java

• Two ways to specify a thread what code it needs to execute
  ➢ By extending a Thread Class
  ➢ By implementing the Runnable interface

```java
# Extending a Thread Class
public class ThreadEx extends Thread{
    public void run(){
        System.out.println("Thread success");
    }
}

# Implements a Runnable interface
public class RunnableClass implements Runnable{
    public void run(){
        System.out.println("Thread Success");
    }
}
```
Creating and Starting a Thread

# Extending a Thread Class
public class ThreadEx extends Thread{
    public void run(){
        System.out.println(“Thread success”)
    }
}

# Implements a Runnable interface
public class RunnableClass implements Runnable{
    public void run(){
        System.out.println(“Thread Success”)
    }
}

# To create and start above Thread
ThreadEx example = new ThreadEx();
example.start();

# To create and start above Thread
Thread th =new Thread(new RunnableClass());
th.start();
public class ThreadJoin{
    public static void main(String args[]){
        Thread t1 = new Thread(new RunnableClass(), "t1");
        Thread t2 = new Thread(new RunnableClass(), "t2");
        Thread t3 = new Thread(new RunnableClass(), "t3");
        t1.start();
        t1.join(); // start t2 thread only after t1 is finished
        t2.start();
        t2.join();
        t3.start();
        t3.join();
    }
}
Synchronized Block

- This assignment requires use of wait() and notify() calls to ensure correct behavior
  - Do not use sleep()
  - Do not use flag variables to control the behavior of the Producer and Consumer

- These calls must be made from inside a synchronized block of code.
  - Only one thread can access at a time
Synchronized Block

• In java, these calls can only be made inside of synchronized blocks

```java
synchronized(syncObject){
    syncObject.wait();
}
```

# Example

```java
public void add(int val) {
    synchronized(this){
        This.count += val;
    }
}
```

# Code is synchronized on the object
Things To Do

• Build your code stepwise – don’t try to do it all at once
• Get your buffer working first
  ➢ Change the size to n = 10
  ➢ Be sure you can add 10 elements
  ➢ Be sure that you can’t add an 11th element
  ➢ Be sure you can remove an element
  ➢ Be sure you can then add a new element
  ➢ Print out the buffer so that you can see that everything is in the right place (i.e. the 11th element should be in the first spot)
• Make sure producer is properly inserting values into buffer and consumer is consuming them
Correctness Verification

• Producer and Consumer should maintain running totals of the values of the items added-to /removed from the buffer

• Maintain in bufferValueCounter variable

• First N elements generated by the producer = First N elements consumed by the consumer
Things Not To Happen

• You should not run out of memory
• There should not be any deadlock
• Do not Consume each item more than once. Item should be consumed in the order that it was produced
• Buffer should not hold more than 1000 items at a time
• Should not use any class from java.util.concurrent package
• Should not use any external library
References

• Example: http://www.careerbless.com/java/Threads/ImplementingRunnable.php
• https://docs.oracle.com/javase/tutorial/essential/concurrency/runthread.html
• https://tutorials.jenkov.com/java-concurrency/index.html
• https://www.tutorialpoint.com/java/java_multithreading.htm