CS 345 - Programming Languages
Fall 2010

Homework #4

Due: 2pm CDT (in class), October 28, 2010

Collaboration policy

No collaboration is permitted on this assignment. Any cheating (*e.g.*, submitting another person's work as your own, or permitting your work to be copied) will automatically result in a failing grade. The Computer Science Department Code of Conduct can be found at `http://www.cs.utexas.edu/academics/conduct/`.

Late submission policy

This homework is due at the **beginning of class** on **October 28**. All late submissions will be subject to the following policy.

You start the semester with a credit of 3 late days. For the purpose of counting late days, a "day" is 24 hours starting at 2pm on the assignment's due date. Partial days are rounded up to the next full day. You are free to divide your late days among the take-home assignments any way you want: submit four assignments 1 day late, submit one assignment 3 days late, *etc.* After your 3 days are used up, no late submissions will be accepted and you will automatically receive 0 points for each late assignment.

Write the number of late days you are using: _____

Homework #4 (35 points)

A classic example of a problem requiring coordinated access to shared resources is the **Dining Philosophers** problem.

Imagine N philosophers seated at a round table. They are thinking and concurrently eating from a shared plate with a big fish in the middle of the table. There is a total of N forks placed at the table so that each philosopher has 1 fork between himself and his neighbor to the right and 1 fork between himself and his neighbor to the left.

Because Miss Manners says that fish is eaten with two forks, a philosopher must obtain both left and right forks before he can eat. The order in which the forks are obtained depends on a coin toss. The philosopher tosses a fair coin; if it comes up heads, the philosopher picks up the right fork, then the left fork. If the coin comes up tails, he picks up the left fork first, then the right fork. If a fork is not available, the philosopher must wait for his neighbor to release it before he can pick it up.[1] He cannot eat unless he has both forks.

After obtaining both forks, a philosopher eats for 1 second, then releases both forks to think again. The order in which the forks are released is also determined by a coin toss. The philosopher thinks for a random number of milliseconds (less than 1 second), then tries to obtain both forks in order to eat again.

You will need to implement the `Philosopher` class. This class should inherit from the `java.lang.Thread` class. You can fill in this skeleton:

```
public class Philosopher extends Thread {
   private String name;
   private Fork left, right;

   // Each philosopher is assigned an integer id and two forks
   public Philosopher (int id, Fork f1, Fork f2) {
      name="Philosopher #"+id;
      left=f1; right=f2; }

   // The run() method is invoked by calling p.start(), where
   // p is an instance of the Philosopher class
   public void run() {
      while(true) {
        System.out.println(name+" is thinking");
        // Think for a random number of milliseconds (less than 1 second)
        // Toss a coin, try to acquire both forks
        System.out.println(name+" is chewing");
        // Eat for 1 second
        // Toss a coin, release both forks }
   }
}
```

[1] Try not to think about the sanitary aspects of sharing the same fork. These are dirty philosophers.

You will also need to implement a `Fork` class, which must enforce synchronized access to each fork. A philosopher can only acquire a fork when it is available; otherwise, he must wait. **At no time may two philosophers hold the same fork.**

Your main program must create the `Fork` objects and the `Philosopher` objects, assign to each philosopher a pair of forks (one shared with the philosopher on the left and one with the philosopher on the right), and start all philosopher threads. Run the program for at least 3 minutes. Each execution should run long enough so that each philosopher gets to eat several times before the program is terminated. Use ctrl-C on Unix or ctrl-Z on Windows to terminate the program.

We will test your program for $N = 2$, $N = 5$, and N equal to some random number between 10 and $10,000$. Your program should suffer from neither *starvation*, nor *deadlock*, nor *unfairness*. To verify this, the program must report on termination how many milliseconds each philosopher spent eating and thinking (you can use `addShutdownHook` to get your statistics object to execute when the program is shut down). It **must** be the case that each philosopher got to eat, and that all philosophers spent similar time eating.

Tips and hints

- Use these parameters to JVM to increase the resources needed to run a large number of threads:
  - `-Xms` set initial heap size
  - `-Xmx` set maximum heap size
  - `-Xss` set thread stack size

- To get a Java thread to sleep for a certain number of milliseconds, call `Thread.sleep()`. The easiest way to do this is write a `randomSleep()` function as follows:

```
int randomSleep() {
// Note that Math.random() returns a double value between 0.0 and 1.0,
// so it must be converted to obtain the right number of milliseconds
   int ms = (int)(... Math.random() ...);
   try {
       Thread.sleep(ms); }
   catch(InterruptedException e) { ... }
   return ms;
 }
```

- To print the statistics after the program is terminated on a command-line interrupt, you can create a "statistics" class as a subclass of `Thread`, and register it with the Java virtual machine to run after the program is shut down, *e.g.*:

```
myStatistics s = new myStatistics();
Runtime.getRuntime().addShutdownHook(s);
```

Submission instructions

1. Submit a paper printout of your Java code, stapled to the first page of this homework (the one showing your name and the number of late days you are using, if any). The printout **must** be processed using the following command:

   ```
   enscript -C -2Gr -Ec <yourfile> -o <outputfile.ps>
   ```

2. Submit your source code electronically using the following command:

   ```
   turnin --submit austin hw4 <filename1> <filename2> ...
   ```