Assignment 2

Engi 8893 / 9869 2009

Due 23:59PM Friday 2009 Mar 11

Programs should be submitted by Websubmit. Submit as assignment 2 in course 8893.

- I will publish the monitor package and some test code. The test code I publish, however, may not represent all tests your code will be subjected to.
- In all cases use comments extensively to explain the logic of your code.
- Where possible and appropriate, use assertions and invariants.
- Any code submitted must compile as is.
- Code must be presented professionally with careful attention to style and documentation. For some of my opinions on the elements of program style see the web-site

`http://www.engr.mun.ca/~theo/Courses/ds/pub/style.pdf`

Preparation: Read up on Java's Thread class and study my monitor package.

Q0 [20]. We need to implement a bag of tasks. For the sake of definiteness, let's suppose each task is represented by a string. Now each worker thread does the following in its run method.

```
while( true ) {
    String task = bag.getTask() ;
    if( task == null ) break ;
    .... bag.putTask( another ) ... }
```

When all workers are waiting in getTask() and the bag is empty, the computation is done, so at this point, all calls to getTask() return with a result of **null**. In addition to putTask and getTask methods, the Bag class needs a constructor that takes the number of workers.

The coordinator might go as follows

```
Bag b = new Bag(N) ;
Worker[] worker = new Worker[N] ;
for( int i=0 ; i < N ; ++i) {worker[i] = new Worker(b) ; }
while( true ) {
    ... put initial tasks in the bag...
    for( int i=0 ; i < N ; ++i) {worker[i].start() ;
    for( int i=0 ; i < N ; ++i) {worker[i].join() ; }
```

Design and implement a suitable monitor class named bag.Bag using the monitor package. Document your class carefully and professionally. Create self-checking code: write as strong a monitor invariant as you can; associate each Condition object with an appropriate Assertion object.

Q1 [20]. The rendezvous monitor presented in class allowed only one server thread at a time to be working on a request. You will design a rendezvous monitor that allows any number of requests to be worked on at the same time. As before, each client behaves as follows:

```
Reply reply = m.submitRequest( req ) ;
```

Each server acts as follows

```
Rendezvous<Request,Reply>.Pair p = m.getPair( ) ;
Request req = p.getRequest() ;
calculate a reply
p.putReply( reply ) ;
```

Design and implement a monitor named rendezvous.Redezvous. It should be a generic class that takes two type parameters. The Pair type is a nested class whose objects are used to communicate one request/reply pair between a client and a server. Thus the public interface of your class should look like this

```
package rendezvous ;
public class Rendezvous<Request, Reply> ... {
    public class Pair ... {
        public Request getRequest() ...
        public void putReply( Reply rep ) ...
    }
```

```
    public Reply submitRequest( Request req ) ...
    public Pair getPair() ...
}
```

Design and implement rendezvous.Rendezvous using the monitor package. Document your class carefully and professionally. Create self-checking code: write as strong a monitor invariant as you can; associate each Condition object with an appropriate Assertion object.

Q2 [20]. Santa has a busy life. While it is true he spends most of it sleeping, whenever three elves are ready to meet with him he wakes up and consults with his elves, and, whenever nine of his reindeer are ready to deliver presents, he goes with them to deliver presents. If there is ever a choice, Santa should deliver presents.

Here are the algorithms for Santa, the elves, and the reindeer. Santa:

```
while(true) {
    boolean x = m.santaIsReady() ;
    if( x ) deliver presents
    else meet with elves
    m.santaIsDone() ; }
```

Each elf:

```
while( true ) {
    make toys
    m.elfIsReady() ;
    meet with Santa
    m.elfIsDone() ; }
```

Each reindeer:

```
while( true ) {
    eat lichen and play reindeer games
    m.deerIsReady() ;
    deliver presents
    m.deerIsDone() ; }
```

Design and implement a suitable monitor class named santa.SantaMon using the monitor package. Document your class carefully and professionally. Create self-checking code: write as strong a monitor invariant as you can; associate each Condition object with an appropriate Assertion object.