This is an individual work.
Any program that fails to compile or that crashes will receive a 0.

There is a successful travel agency which receives a lot of requests from clients. However, as one agent is only capable of processing a certain number of orders at a time, some of the clients’ requests need to wait in a queue. The owner of that travel agency wants to know how long a client needs to wait before his request can be completed. You program will simulate the activities in the travel agency and report the average waiting time of clients’ requests.

a) There are 3 agents. Each can complete a client’s request in half of a day (4 hours), and can handle 10 orders at a time. When there is no clients’ requests for him to process, an agent sleeps.
b) Clients request to meet with the agents when they have the itinerary ready. There is only one nice meeting room that everybody wants to use. However, only one group of people can use it at a time. The meeting takes 30 minutes.
c) The agents work 7 days a week. They go to work at 8:00 am and leave at 6:00 pm. (Your program starts at 8:00 am.)
d) In the simulation, we use 1000 millisecond to simulate one hour.

Classes:
a) The following two classes are provided for you to use.

```java
class CountingSemaphore {
    private int s;

    public CountingSemaphore(int n) {
        s = n;
    }

    public synchronized void Wait() {
        try {
```
s--;  
    if(s < 0) wait();  
}  
    catch(InterruptedException e) {}  
    catch(IllegalMonitorStateException e) {} 
}

public synchronized void Signal() {  
    try {  
        s++;  
        if(s<=0) notify();  
    }  
    catch(IllegalMonitorStateException e) {}  
}

class BinarySemaphore {  
    private boolean s;  
    int nwait = 0;  

    public BinarySemaphore(boolean n) {  
        s = n; nwait = 0;  
    }

    public synchronized void Wait() {  
        try {  
            if(!s) {  
                nwait++; wait();  
            } else s = false;  
        }  
        catch(InterruptedException e) {}  
        catch(IllegalMonitorStateException e) {}  
    }

    public synchronized void Signal() {  
        try {  
            if(nwait > 0) {nwait--; notify();} else s = true;  
        }  
        catch(IllegalMonitorStateException e) {}  
    }
}

b) You will need to write 3 classes.  
i. Driver.java is the entry class which starts the agent thread and generate the clients’ requests.  
ii. Agent.java simulates the behavior of an agent.  
iii. MeetingRoom.java is the meeting room.
Input:
Your program takes two parameters: the number of clients and a flag indicating whether to print the status information.
The command line uses the following format
java Driver –c [# of clients] -s [Y/N]

Output:
i) When the program starts
   Welcome to one-globe travel agent!

ii) status information (can be turned off with –s option)

When an agent start working on one order:
Agent [thread number of that agent thread] receives client’s request [number of that client].
When an agent finished one order:
Agent [thread number of that agent thread] finishes client’s request [number of that client].
When an agent meets with a client:
Agent [thread number of that agent thread] meets client [number of that client].
When an agent finished meeting with a client:
Agent [thread number of that agent thread] finished the meeting with client [number of that client].

iii) When program finishes:
We finish [number of clients] requests in [total running time] hours.
The average waiting time of a client is [waiting time] hours.
(Waiting time the time between when a client submitted his request, until some agent start working on his case including the night time).

Extra credits:
   i) Adding a parameter which controls the number of agents. –a [# of agents]
   ii) Adding a parameter which controls the number of meeting rooms
       –r [# of rooms]
   iii) Which parameter has the most impact on the average waiting time of the clients’ requests? Number of clients, number of agents or number of meetings rooms?

To turn in the program:

   a. tar and gzip your programs
   b. Turn in your program using the following command
      /nfs/cs/classes/juan/turinin/turinin -p 6435 -s ix.cs.uoregon.edu pa1 pa1.tar.gz