CS 3345, Programming Project 1

First, read the note on Segmented Virtual Memory.

Write a program to simulate the arrivals, departures, and placements of segments in a segmented virtual memory, as described below.

Create the following classes:

```
class Node {
      boolean segment;   // equals false if this Node represents a hole
      int location;      // position in memory of first byte
      int size;
      int timeToDepart;  // only valid when this Node represents a segment
      Node next;

     /*
      constructor for a segment
     */
      Node(int locn, int sz, int endOfLife, Node nxt) {
          segment = true;
          location = locn;
          size = sz;
          timeToDepart = endOfLife;
          next = nxt;
      }

     /*
      constructor for a hole
     */
      Node(int locn, int sz, Node nxt) {
          segment = false;
          location = locn;
          size = sz;
          next = nxt;
      }
}

class Memory {
      Node head;          // reference to first Node in memory - could be a hole or a segment
      Node lastPlacement; // references the last segment placed, or a hole if that segment is removed

    /*
       constructor for Memory, generates a single hole Node of the given size
    */
      Memory(int size);

    /*
      attempt to place a request using the Next Fit policy. Returns false if there isn't a hole big enough.
      Prints a confirmation if placed and verbose==true
    */
      boolean place(int size, int timeOfDay, int lifetime, boolean verbose);

    /*
       remove segments whose time to depart has occurred
    */
      void removeSegmentsDueToDepart(int timeOfDay);

    /*
       print a 3-column tab-separated list of all segments in Memory
    */
      void printLayout(); // see below
```

```
    . . .
}
```

Simulating the Segmented Virtual Memory
Initially the memory is represented by a single hole Node with size equal to memorySize. As the simulation progresses the holes and segments in memory become represented by a singly linked list of Nodes.

Variable `head` always points to the Node with the lowest address.

You will simulate the **Next Fit** placement policy, where each search begins at the location in memory where the last placement took place. It and scans the Nodes in the Memory object in a circular fashion until a hole is found that is large enough, or it fails if no such hole exists.

On a successful placement variable `lastPlacement` is set to reference the newly placed segment.

Care must be taken when deleting a segment. **There must never be two neighboring holes in memory**. And, if the `lastPlacement` variable refers to the segment being deleted, that variable must be changed to refer to the hole that is created by the departing segment.

The main method has variables:

```
    Memory memory;  // the memory object
    int timeOfDay;  // the simulated wall clock, begins with value zero
    int placements; // the number of placements completed, begins with value zero
    long totalSpaceTime; // the sum of placed segmentSize(i) x sementLifetime(i)
```

The program reads and obeys command lines from System.in.

Input Commands

```
N           // print your name followed by a newline
C s         // create a Memory object of size s
A u v       // add a segment of size u and lifetime v and print a confirmation record:
            // "Segment of size u placed at time t1 at location l, departs at t2"
            // where u, t1, l, t2 are printed with format "%4d"
P           // print a list of all the segments in memory (see below)
R s u v w x // create a new Memory object of size s. Simulate x randomly generated
            // placements (see below). Do not print confirmation records on each placement
            // but do print stats as described below
E           // end of data file, print a newline (‘‘\n’’) and exit
```

Data files containing the R command will not contain any C, A, or P commands.

Here is the section of my `main()` method that deals with commands:

```
    // Scanner sc = new Scanner(System.in);  // switch the comments before submitting
    Scanner sc = new Scanner(new File("p115sd1.txt"));
    String line = "";
    boolean done = false;
    while(!done) {
        line = sc.nextLine();
        String [] tokens = line.split(" ");
        switch(tokens[0]) {
        case "N": {
            System.out.println("Ivor Page");
            break;
        }
```

```
        case "C": {
            memory = new Memory(Integer.parseInt(tokens[1])); // create a new Memory object
            break;
        }
        case "E": {
            done = true;
            break;
        }
        . . .
    } // end of switch
```

The A Command
On receiving each `A u v` command the following takes place:

```
timeOfDay++;
memory.removeSegmentsDueToDepart(timeOfDay);
while(!memory.place(size,timeOfDay,lifeTime,true))  { // timeToDepart=timeOfDay+lifeTime
   timeOfDay++;
   memory.removeSegmentsDueToDepart(timeOfDay);
}
placements++;
// then print the confirmation message
```

The P Command
On receiving the P command your program will print a tab-separated 3-column list of the segments in memory as follows

```
seg1-location       seg1-size       seg1-timeOfDeparture\\
seg2-location       seg2-size       seg2-timeOfDeparture\\
seg3-location       seg3-size       seg3-timeOfDeparture\\
. . .
```

where the segments are listed in order of increasing position in memory.

Example:

Input Commands	Expected Output
N	Ivor Page
C 100	`Segment of size   20 placed at time    1 at location    0, departs at   11`
A 20 10	`Segment of size   50 placed at time    2 at location   20, departs at    7`
A 50 5	`Segment of size   70 placed at time    7 at location   20, departs at   27`
A 70 20	`0      20      11`
P	`20     70      27`
E	

Nothing is printed in response to the C command.

The R Command
In response to command `R s u v w x` the program creates a new Memory object with size s. Parameters `u` and `v` specify the minimum and maximum segment sizes, and `w` specifies the maximum lifetime of segments to be created during the following simulation. The simulation stops after `x` segments have been placed.

Here is a sketch of the main loop for the `R` command:

```
memory = new Memory(s); // initialize variables in main()
timeOfDay = 0;
placements = 0;
```

```
Random ran = new Random();  // pseudo random number generator
while (numberOfPlacements < x) {
    timeOfDay++;
    memory.removeSegmentsDueToDepart(timeOfDay);
    int newSegSize = u + ran.nextInt(v-u+1);
    int newSegLifetime = 1 + ran.nextInt(w);
    totalSpaceTime += newSegSize*newSegLifetime;
    while(!memory.place(newSegSize, timeOfDay, newSegLifetime, false)){
        timeOfDay++;
        memory.removeSegmentsDueToDepart(timeOfDay);
    }
    placements++;
}
```

The segment sizes should be uniformly distributed in [u, v] and the lifetime should be uniformly distributed in [1, w].

The final step in execution of the R command is to print the following stats:

```
Number of placements made = p
Mean occupancy of memory =  q
```

where the p value is printed with format `"%6d"` and q value is printed with format `"%8.2f"`.

The mean occupancy =

$$\frac{\sum_i segmentSize_i \times segmentLifetime_i}{\text{timeOfDay at the end of the simulation}}$$

Examples:

Input Commands	Expected Output
N	Ivor Page
R 100 5 25 20 10000	Number of placements made = `10000`
E	Mean occupancy of memory = `74.77`

Input Commands	Expected Output
N	Ivor Page
R 1000 5 100 80 10000	Number of placements made = `10000`
E	Mean occupancy of memory = `796.35`

Many datasets will be used in testing your code.

Source file structure
Use the following to get three classes into your one source file:

```
// file IVPAP1.java - use your name

import java.util.*;
import java.io.*;

class Node {
 . . .
}

class Memory {
 . . .
}

class IVPAP1 { // use your name
 . . .
 public static void main() {
   . . .
 }
 . . .
}
```

RULES FOR PROGRAMMING AND SUBMISSION:

(1) THIS IS NOT A GROUP PROJECT! YOUR SUBMISSION MUST BE YOUR OWN WORK. CITE ANY ELEMENTS OF CODE THAT YOU COPY FROM ELSEWHERE. YOU MAY USE THE CODE GIVEN IN THIS PROJECT SPECIFICATION.
(2) Write your program as one source file and do not include the `package` construct in your Java source.
(3) Name your source file as $N_1N_2F_1F_2$P1.java where your given name begins with the characters N_1N_2 and your family name begins with the characters F_1F_2. For example my name is Ivor Page, so my source file will be called IVPAP1.java. Note that in all but the "java" extension, the characters are upper case.
(4) Your program's output must exactly match the format of the Expected Output columns above
(5) Use Microsoft C++, Gnu g++, or Java 1.7x
(6) Do not use any Java Collection Classes, except Strings. If in doubt, ask me.
(7) Your program must compile and run in a DOS window or a UNIX terminal window using file redirection.
(8) Your program must read from System.in and output to System.out.
(9) Use good style and layout. Comment your code well.
(10) Submit your ONE source code file to the eLearning drop box for this assignment. Don't submit a compressed file.
(11) **There will be a 1% penalty for each minute of lateness. After 60 late minutes, a grade of zero will be recorded.**

Send any questions/corrections to ivor@utdallas.edu.