ITI 1120 Fall 2012 - Assignment 2

Available: Monday Oct 8, 2012
Due: Monday Oct 22, 2012, 11:59 pm

Instructions

This assignment is to be done INDIVIDUALLY. Follow the instructions in the lab manual for submitting assignments through the Virtual campus. The following are specific instructions for this assignment:

- Question 1 should be answered in a Word file A2Q1.doc, a template is provided for you to complete.
- Question 2 should be answered in a Word file A2Q2.doc, a template is provided for you to complete..
- Question 3 should be answered in a Java File A2Q3.java. Paste the content of the A2Q3.java file into the Word file A2Q3.doc.
- Question 4 should be answered in a Word file A2Q4.doc, a template is provided for you to complete. In addition a Viso file A2Q4VisioTrace.vsd has been provided to help answer the question.
- Question 5 should be answered in a Java File A2Q5.java. Paste the content of the A2Q5.java file into the Word file A2Q5.doc.
- Inserting the Java source code into Word files will allow the marker to make comments on the source code in the Word file.
- Zip all the .doc, .java, and .class files in a2_xxxxxx.zip, where xxxxxx is your student number, and submit it through the Virtual Campus).

Your algorithm traces should use the format shown in class; a separate table is to be used for each call to each algorithm.

- Be sure to start the assignment soon and not wait until the last weekend to start. Better to put many small efforts over the next two weeks to complete the assignment than one single large effort.

Marking Scheme (total 100 marks)

- **Regulations and Standards: 5 marks**
- Question 1: 5 marks
- Question 2: 20 marks
- Question 3: 20 marks
- Question 4: 25 marks
- Question 5: 25 marks

Question 1 (5 marks)

Time-of-Use periods chart shown below [http://www.hydroone.com/TOU/Pages/Default.aspx] was developed by the Ontario Energy Board (OEB) in order to encourage residents of Ontario to use electricity during low-demand periods. According to this chart, the price of electricity depends on the time of the day. There are three types of Time-of-Use periods in this chart including off-peak, mid-peak, and high-peak. During off-peak periods electricity is the cheapest and during high-peak periods electricity is the most expensive. Note that time takes on values between 00:00 (midnight) and 23:59.

Season	Time	Weekends/ Holidays	Time-of-Use periods
ANY	ANY	YES	off-peak
Summer	19:00-7:00	NO	off-peak
Winter	19:00-7:00	NO	off-peak
Summer	7:00-11:00	NO	mid-peak
Summer	17:00-19:00	NO	mid-peak
Winter	11:00-17:00	NO	mid-peak
Summer	11:00-17:00	NO	high-peak
Winter	7:00-11:00	NO	high-peak
Winter	17:00-19:00	NO	high-peak

The following variables are defined:

- *season:* it takes one of two character values: 's': summer and 'w': winter,
- *hour:* it takes the value of the hour in the Time column
- *minute*: it takes the value of the minute in the Time column.
- *holiday* (for weekends/holidays which is TRUE if it is a holiday and FALSE otherwise).

Provide three Boolean expressions that give a result of TRUE when the Time-of-Use period is off-peak, mid-peak, and high-peak, respectively. (Hint: for the case 19:00-7:00, divide the range with respect to midnight.) Use the **algorithm** format for the Boolean expressions. The Boolean expressions that for this question shall use only the comparison operators (<, >, ≤, ≥, =, etc.), Boolean operators (AND, OR, NOT), and math operators +, −, ×, /, or MOD. Use parentheses where necessary and for clarifying the expression. Do not use Java syntax; otherwise you get a ZERO mark!

Question 2 (20 marks)

In this question, you will develop a guessing game, in which users will play against a computer. A number between 1 and 100 will be secretly chosen by the computer and the player will try to discover this number by repeatedly guessing numbers. If a guess is greater than the chosen number, the program prints out '>', if the guess is smaller than the chosen number, the program prints out '<', and otherwise the guess is correct and the player wins. The number of guesses is limited.

The maximum number of guesses is determined by the player who indicates how difficult the game should be. There are three levels of difficulty, including easy (between 20 and 30 guesses), difficult (between 10 and 15 guesses), and very difficult (between 1 and 5 guesses). The user determines his desired level by entering 1 for easy, 2 for difficult and 3 for very difficult. For example, if the user wishes an easy game he will enter 1 and the program generates a random number between 20 and 30 to limit the number of guesses. The following shows an example output for a game.

```
Guessing Game
Do you want to play a
   1. Easy game?
   2. Difficult game?
   3. Very difficult game?
Make a choice (1, 2, or 3) : 4
Invalid Choice!
Make a choice (1, 2, or 3) : 1
You have 22 guesses to find the secret number.
Guess 1 of 22: 30
>
Guess 2 of 22: 10
<
Guess 3 of 22: 120
The number 120 is not valid.
Guess 3 of 22: 20
<
Guess 4 of 22: 25
>
Guess 5 of 22: 23
You WIN in 5 guesses – the secret number is 23!
```

If the player does not guess the number, the following message appears:

```
GAME OVER – the secret number is 23!
```

a) Develop an algorithm, *guess*, which gets from the player guesses and evaluates them. The algorithm uses the givens *numGuesses* that gives the number of guesses allowed to the user and *secretNum*, the secret number to be guessed. The algorithm shall return the number of guesses used to find the secret number or the value -1 if the player could not guess the number. The algorithm shall prompt the user for guesses and indicate too high or too low with the characters '>' or '<' as shown above. If the user gives a number outside the valid range between 1 and 100, then the error message is displayed and prompted for a valid guess as shown above.

b) Develop the main algorithm to get from the user the level of difficulty before starting the game. The algorithm checks for invalid values entered by the user for difficulty level and will print an error message as shown above. Any value other than 1, 2, and 3 is invalid for difficulty level. Using the level of difficulty, the algorithm determines the number of guesses allowed for the user. Then the algorithm determines the secret number to be guessed. The algorithm then calls the *guess* algorithm to let the user play the game. The main algorithm uses the results of the *guess* algorithm to print the final message.

***** There is a pre-defined algorithm *random()* which generates random numbers greater than or equal to 0.0 and less than 1.0, that is in the range [0.0,1.0). You should use this algorithm to generate random numbers between two values, say contained in the variables a and b*.* A random number between the values of a and b (for example between 1 and 100) is assigned to the variable *randomNum* by the following instruction:

$$randomNum \leftarrow a + random() * ((b - a) + 1)$$

Question 3 (20 marks)

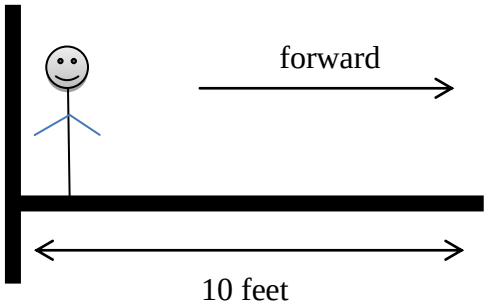
Translate the algorithm models from question 2 into a Java program. To translate pre-defined algorithm *random*(), you should use the Java method `Math.random()` (see Chapter 5 (Methods) Section 10.5 in the course text book, or the Java API documentation). Note that `Math.random()` generates a random **double** value in the range **[0.0,1.0)** (that is greater than or equal to 0.0 and less than 1.0). To generate a random **integer** value in the range ***[a,b]*** (that is, greater than or equal to the value of a or less than or equal to the value of b)*,* e.g. *randomNum*, you should type-cast the double value to an int value as follows:

```
randomNum = a + (int)(Math.random() * ((b - a) + 1));
```

where `(int)` performs the type conversion from double to int.

Question 4 (25 marks)

Assume a man is preparing to dive, as shown in the following picture. The man needs to move 10 feet forwards to go over the edge.



The man takes one step every second, and each step moves him two feet. He may, however, not always move forwards. Because he is nervous, he may step backwards, particularly as he gets near the edge. The probability that the man will take a step forward is given by

$$P(\textit{forward step}) = 1 - (x/10)$$

where x is the man's distance (in feet) from his starting point

Note that, when x is 0 (man at his starting point), the man is certain to move forwards, but that, when x is 8 (the man is on the brink of going over the edge), the probability of him moving forwards is just 0.2. Note also that, if the man does not go forwards, he goes backwards. If the man does not go over the edge within *tMax* seconds, he will lose his nerve completely and will not dive at all.

a) Develop an algorithm, dive, which simulates a dive attempt according to the following guidelines:

- The algorithm has one given, tMax, which is the number of seconds during which the man should reach the edge to dive, otherwise he will not dive.
- Two results, *moves* which is a reference to a character array showing the sequence of man's moves either F (forward) or B (backward) or D (dive) and *num* which gives the number of elements in the array.
- Since each move takes one second, the result *num* also gives the number of seconds taken by the man to either dive or lose his nerve. When the man dives the last element of the array will have the character D.

- To determine whether or not the man will move forward, the algorithm first computes the probability of moving forward according to the above equation using the distance of the man from the starting point. The algorithm then generates a random number between 0.0 and 1.0 (recall the predefined algorithm *random()*). If the generated random number is **less than** the probability of moving forward, it means that the man moves forward, otherwise he moves backward.

b) Develop the main algorithm according to the following guidelines:

- Define a loop to allow the user to run many simulations. When the program is started a first dive is simulated and then the user prompted to run another simulation. If the user decides to run another simulation the steps described in this point are repeated, otherwise the program terminates. An example run is given below.
- To run a simulation, the user is first prompted to give the maximum time that the man has to dive. Then the *dive* algorithm is called to simulate the dive and the returned results shall be displayed as shown below.

Sample run:

```
Please enter the maximum time in seconds to make a dive: 15
The man jumped with the moves FBFFFBFBFFD in 11 seconds

Do you wish to make another dive attempt?(y/n): y
Please enter the maximum time in seconds to make a dive: 10
The man lost his nerve after the moves FFBFBFFBFB

Do you wish to make another dive attempt?(y/n): n
Program terminated!
```

c) Trace the following call (*moves*, *num*) ← *dive*(5) using the provided programming model in the Visio file A2Q4VisioTrace.vsd. Please cut and paste the completed diagram in the Word file A2Q4.doc. In the programming model indicate how the contents of variables and arrays change (you do not need to show how values are exchanged between the algorithms). For reference variables, give the values *Adr1*, *Adr2*, etc. to show the different values. Also indicate with arrows what the addresses reference. Also complete the trace table given in A2Q4.doc. Assume that the calls to the predefined algorithm *random()* returns the sequence of numbers: 0.59, 0.27, 0.41, 0.07, 0.42. This means that the first call to *random*() returns 0.59, the second call returns 0.27, and so on.

Question 5 (25 marks)

Translate the algorithm models from question 4 into a Java program. Note that the pre-defined algorithm model *random()* can be translated to the Java method *Math.random*() (see Chapter 5 (Methods) Section 10.5 in the course text book, or the Java API documentation).