Written test Questions - Software Developer (Java)

Instructions:

- Answer question 5 and any other.
- Create a separate main class for each question with the name of the main class corresponding to the question number. E.g. The main class for question 1 will be called *Question1*. You are free to create other classes and name them as you wish, following proper naming convention and clean coding principles
- Question ***5*** must be attempted, but you may answer the questions in any order
- Use any IDE of your choice. All implementation should be enclosed within one project and the name of the project should be <Surname_Firstname>
- To submit, zip up your project folder, and any other input or output file in an archive with the same name as your project name and submit as attachment to careers@byteworks.com.ng with the subject "Software Developer Assessment Center - <Surname, Firstname>"
- Do not write on the question papers and return when done. You may request for plain sheets if you need
- You have 1hour for this. And may request for extra time not more than 30 mins
- Do not use classes from java.util.* package

Good luck!

1.
Create a class called Meal with attributes (2 Marks)

- name(String)
- proteinContentInMiligram(Double)
- carbonHydrateContentInMiligram(Double)
- fatContentInMiligram(Double)

2.
Implement a Linked List to hold objects of class Meal you created above with the following methods (30 marks)

- **add** - for adding a new meal to the linked list (takes a meal and returns void)
- **remove** - for removing a meal from the linked list (takes an object, if the object is an integer type, it removes the meal at that index and returns it, if the object is a Meal type, it removes the meal and returns it, throws an exception for any other type)
- **find** - to get the index of a meal, returns -1 if the meal is not found and the index of the meal if found
- **toString** - this should properly override the toString method in the Object class and when called, it should printout the names of all the meals in your list.

3.

Write a java program that uses an interface named Shape, having methods:

- double area()
- double perimeter().

Use this interface to find the area and perimeter of circle, rectangle or square based on the choice of user. Prompt the user to enter the name of the shape and value(s) required to compute area and perimeter.

Example: The output of the program may be as follows:

```
Menu-
Enter your choice: Circle
Enter the radius:3.5
Area of Circle is:38.465
Perimeter of Circle is:21.98

Menu-
```

```
Enter your choice: Rectangle
Enter the length:4
Enter the width:2
Area of Rectangle is:8
Perimeter of Rectangle is:12

Menu-
Enter your choice: Square
Enter the side:2.2
Area of Square is: 4.84
Perimeter of Square is:8.8

Menu-
Enter your choice: Ellipse
This is not a recognized shape. Allowed shapes are: Circle, Square, Rectangle
```

4.

You are given an integer N. Print the factorial of this number.

$$N! = N \times (N-1) \times (N-2) \times \cdots \times 3 \times 2 \times 1$$

Input
Input consists of a single integer N, where $22 \le N \le 100$.

Output
Print the factorial of N.

Example
For an input of 25, you would print 15511210043330985984000000. No exponentials. No decimal places.

Hint: Use BigInteger where appropriate

5.

In the world of Teradata, each country has one central bank and its own payment gateway associated with it. They all are connected to each other using a *tree topology*. If a customer is trying to transfer funds from country X to Y then the funds have to flow through all intermediate banks connecting X and Y. For each fund transfer multiple data packets are needed to be transferred from source to destination bank.

Tree topology is structured like a tree, where it has a root node, intermediate nodes and leaves. Root node is the head node of the structure, and the leaves are the last nodes, which has no further child nodes. This structure is arranged in a hierarchical form, each node can have any number of the child nodes.

Let's say there are N central banks in the Teradata world and all of them are uniquely numbered between $[1\ldots N]$. Root bank will be represented by 1. All edges are bidirectional in nature, i.e., funds can flow in any direction. It is guaranteed that there will be exactly one path between each pair of banks. There will be $N-1$ connections which will be used to connect the banks.

But there is an issue with the connections. None of them are completely reliable and only a certain percentage of packets successfully pass through them. Each packet has a constant probability of successfully transmitting through each link and is independent of other links. *A data packet can only be transported from one bank to another if the probability of its successful transmission between the end-points exceeds a certain threshold value.* Note that this value can be different each time.

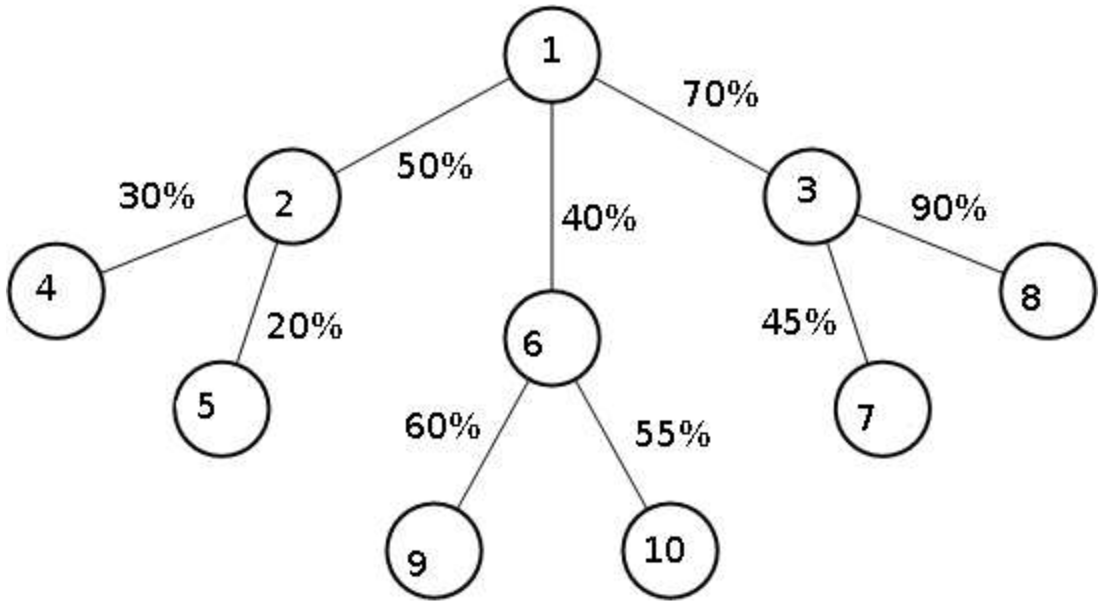

Fig 1: Teradata World

Above figure represents a sample bank structure which is built on tree topology. There are 10 banks in the country.

- Probability of packet transmission between bank #4 and #2 $=pedge4,2=30/100=0.3$.
- Probability of packet transmission between bank #5 and #1 $=pedge5,2*pedge2,1=(20/100)*(50/100)=0.1$.
- Probability of packet transmission between bank #9 and #10 $=pedge9,6*pedge6,10=(60/100)*(55/100)=0.33$.
- Probability of packet transmission between bank #7 and #10 $=pedge7,3*pedge3,1*pedge1,6*pedge6,10=451/00*70/100*40/100*55/100=0.0693$.

Write a program that reads a configuration input file "***teradata-world-setup.txt***" with the following sample content and description:

```
10
1,2,50
2,4,30
2,5,20
1,6,40
6,9,60
6,10,55
1,3,70
3,7,45
3,8,95
```

First line of "***teradata-world-setup.txt***"contains an integer, N, representing total number of banks in the topology. Then follows $N-1$ lines. Each line will contain three comma separated integers, u,v,p, where u is the parent of v and p represents the probability of successful transmission in percentage, i.e. probability for successful transmission of each packet between u and v is $p/100$.

Constraints

- $2 \leq N \leq 10$
- $1 \leq u < v \leq N$
- $0 < p \leq 100$

Also create three client request files named "***client-request-<i>.txt***" (i = 1, 2, 3, ...) with the following constraints for the content:

A client will send multiple requests to the server, represented by each line in the sample above. Each of these lines contains three comma separated numbers, a,b,q, where a and b represents the banks. And q is the threshold probability for that request (i.e. if the probability of transmission between two banks is greater than or equal to q then transmission will be successful) required for a package to be transmitted successfully. End of requests in a client request file will be represented by the string END and you can end the program when this string is encountered.

The file ***client-request-1.txt*** can have the content below. Use this as a guide, and then create two more client request files with content satisfying the constraints defined below:

Content for client-request-1.txt:

```
4,2,0.29897
5,1,0
2,4,0.3012
9,10,0.34879
7,10,0.2
1,5,0.1245
9,10,0.4771
7,10,0.1586

END
```

Constraints

- $2 \leq$ *Total requests by all clients* $\leq 10^5$
- $1 \leq a,b \leq N$ *AND* $a \neq b$, *where* $a, b \in \mathbb{Z}+$
- $0 \leq q \leq 1$, *where* $q \in \mathbb{R}$

Output:

For each client request file, your program should generate an output file "**client-response-<i>.txt** (i = 1, 2, 3, ... corresponding to the request). The output file should print YES or NO for each request in the client request file depending on whether the data packet can be successfully transmitted or not.

E.g. the content of a client response file can be:

```
YES
NO
NO
YES
YES
YES
NO
NO
```