IE Janvier 2016
Duration : 1h30

*This subject contains **6 pages and 3 Exercises** – Indicative grading scale on 20 pts*
Authorized documents : none – Calculators, phones and computers are not authorized

The name of the classes, methods and parameters order must EXACTLY be the same as those specified in subject.

Warning : not respecting Java coding conventions may result in penalties.

Exercise 1 – Algorithm understanding (5 pts)

Reminder: the length() method gives the length of a string of characters. The next() method associated to the sc Scanner (in the main) reads the string of character in input.

```
import java.util.Scanner;
public class IE_Exo1Ter{
      public static void computation (String s){
            int u, v, x,y ;
            u = 0; v = 0;x=0;y=0;
            if ((s.charAt(0)>='0')&&(s.charAt(0)<='9'))
                  u=1;
            else u=0;
            for (int i=0; i<s.length(); i++){
                  if (s.charAt(i)== '@'){
                        u = -1 ; v = 0; y=x;
                  } else if (s.charAt(i)== '%' ){
                        u= 1 ; v = 0; y=x;
                  } else if (((s.charAt(i)>='0')&&(s.charAt(i)<='9'))){
                        v =10*v+ (int) s.charAt(i)- (int) '0';
                        System.out.println("v="+v);
                  }
                  x = y+u*v;
                  System.out.println ("fin:"+x);
            }
      }

      public static void main (String[] args) {
            String myString;
            Scanner sc = new Scanner(System.in);
            System.out.println ("Enter a character string :");
            myString=sc.next();
            computation(myString);
      }
}
```

Q1. (3,5 pts) Fill the following table with values taken by the variables and parameters of the *computation* method during its execution when the string provided as input is "@12%4". The table must contain 6 lines: the initialization before the loop, then each of the 5 turns in the loop. You will also indicate what is displayed by the program at each turn in the loop.

i	u	v	x	y	Display

Q2. (1,5 pts) What is this program doing ?

Exercise 2 – Square root of a positive number (3 pts)

Write a method return a real number of type double which computes the square root of a positive integer a, knowing that the following sequence converge towards the square root of a:

$U_0 = 1$

$U_{i+1} = 0.5\ (U_i + a/U_i)$

The computation will stop when : $U_i - \varepsilon \leq U_{i+1} \leq U_i + \varepsilon$. The value of ε will be defined as a small number (for instance $\varepsilon = 10^{-6}$).

- The method takes as parameters the integer a and the real number ε
- It will return the value of the square root of a, if it is positive. If not, it will return the value -1 and will display the message 'ERREUR' on the screen.

Make sure you define the header of the method with all the parameters as well as the local variables if needed.

Exercise 3 – Game of Nim (12 pts)

The objective of this problem is to program to version of the game of Nim (also called Fan Tan or Tiouk Tiouk).

In all this exercise, you will assume that you have a method `int readInteger()` allowing to read an integer provided by the user through the keyboard. Example : `int n = readInteger() ;`

Even if you have not written one of the methods asked in the subject, you can use it in the following questions.

Part 1 : class Nim_Version_1

We consider N heaps of token, numbered from 0 to N-1. The players take alternate turns. When it is his turn, the player indicates the number *i* of the heap from which he want to retrieve some tokens. If nbp_i designate the number of token in the heap *i*, the player can retrieve between 1 and nbp_i tokens in this heap (but he will be allowed to retrieve tokens from another heap at the next turn). The winner is the one who retrieve the last token of the game. The loser is the one who have to play when all heaps are empty.

We want to write a program allowing to human players to play, by displaying the number of tokens in each heap, asking to the first player to select a heap from which he wants to retrieve tokens as well as the number of tokens. The program then updates the number of token in the heap and displays the new states of the heaps. It then asks the second player, to select a heap and a number of

token. And so on as long as the current player can play.

The program itself does not play the game, it only asks players their choices, maintains the state of the heaps, displays the heaps and indicate the winner at the end of the game.

An example of game with N = 3 is provided in annex.

To implement this version of the game, we chose to store the number of token in each heap using an array.

Q1. (1pts) Write a method `inputBetweenBounds` that asks to the user to enter a number comprised between two numbers until the entered number belongs to the specified interval. This method returns an integer:

```
public static int inputBetweenBounds(int bMin, int bMax)
```

Q2. (0,5 pts) Write a method `displayArray` that displays over a single line the values contained in an array of integers. Values must be separated by spaces on the display.

```
public static void displayArray(int[] t)
```

Q3. (1 pt) Write a method that build and return an array of k integers, each of these integers corresponding to the number of token in a heap, this number is randomly chosen between 1 and an integer nbMax, nbMax included.

To generate a random number, you can use the method Math.random() that returns a real number of type double comprised in the interval [0..1[.

```
public static int[] generateArrayInit(int k, int nbMax)
```

Q4. (1 pt) Write a method that takes an array as input and return true if it is not possible to play (there is no token left on any of the heaps).

```
public static boolean over(int[] tab)
```

Provide a solution where the loop stops as soon as possible. For instance, if there are 4 heaps and that the second is not empty, we know that the game is not over without having to check the state of the following heaps.

Q5. (2,5 pts) Write a Java program allowing two players to play to the game of Nim with N heaps. Each player has a number (1 or 2). N will be declared as a constant. The number of token in a heap must not be greater than 20. The program stops as soon as there is no token left to retrieve. It then displays the number of the winner.

Part 2 : Class Nim_Version_2

In this part, a human plays against the computer. The objective is to program the methods for the computer to play. The number N still designates the number of heaps. The number of tokens is still randomly chosen but is now limited to 15. We note nbp_i, $0 \leq i \leq N-1$, the number of tokens in the heap

i.

A state is defined by a set of value (nbp_0, nbp_1, ... nbp_{N-1}) corresponding to the number of token in the heaps. Retrieving tokens in a heap move the game to one state to another.

To determine which move to play, the computer computes a value characterizing the state. This value is called sum of Nim. To compute it, we need to:

- Compute the binary representation of the nbp_i $0 \leq i \leq N-1$; let b_0, b_1 ... b_{N-1} be those binary representations.
- Compute the XOR (exclusive OR) of all the b_i $0 \leq i \leq N-1$. To do this, it is sufficient to perform a bitwise sum (sum bit per bit), without taking into account the carries. In other words, we get a 1 in one column if the number of 1s in this column is odd.
- The resulting binary number is the sum of Nim (XOR), noted *sumNim*, of which the base 10 representation can be computed.

An example of the sum of Nim of one state is provided in annex. We note that this operation is associative.

The binary representation of the number of token will stored in an array of size 4, the cell of index 0 corresponding to 2^3.

We assume that there is a method `decInBinary` that given a number encoded in base 10, build and return an array of size 4 containing the binary representation of this number.

```
public static int[] decInBinary(int n)
```

Q6. (1 pt) Write a method that transforms the binary representation of a number, stored in an array, into an integer in base 10.

```
public static int binToDecimal(int[] tab)
```

Q7. (1,5 pt) Write a method that takes as parameters 2 numbers written in base 2 and stored in an array, and that returns their sum of Nim (XOR) in base 2. We assume that the two integers have the same size.

```
public static int[] sumNim(int[] bi, int[] bj)
```

Q8. (2,5 pts) It has been shown that, when it is its turn to play, a player has an advantage if the current state has a sum of Nim different from 0. At the opposite, in a case of a null sum of Nim, the opponent can win if it plays well.

Write a method that, given a state characterized by an array of N numbers of tokens represented in base 10, returns true if the sum of Nim of this number is strictly positive.

```
public static boolean favorableState(int[] state)
```

Q9. (1 pt) Write a method main that allows to test whether the method `favorableState` works

properly.

The following question is a « BONUS », answer it if you have answered all the other questions. The objective is to make a human play against the computer.

Q10. (BONUS – 3,5 pts) When it is its turn to play, the strategy of the computer is the following :

1 If the current state has a sum of Nim equal to 0, the computer adopt a stalling strategy by taking only one token in a non-empty heap.

2 If the current sum of Nim is different from 0, the computer chooses a move that brings the opponent to a state having a null sum of Nim. To do this it needs to :

- Computes the XOR of each b_i with *sumNim*. Let sx_i be the corresponding values.
- The heap to choose is the one for which the number of tokens is greater than the sum of Nim. In other words, the heap for which: $sx_i < b_i$. Let k be the index of this heap.
- The number of token n_k to retrieve from the heap k must bring b_k to sx_k. In other words, $n_k = b_k - sx_k$.

Write a Java program that allows a human player to play against the computer.

An example of computation of move is presented in annex.

Annex 1 : Display of a Nim game with 3 heaps

The following example presents the expected display. `C`, `P1` and `P2` designate respectively the computer, the player 1 and the player 2. There are 3 heaps numbered 0, 1 and 2.

```
C : Number of tokens in the heaps : 12 5 3
C : Player1, which heap ?
P1 : 1
C : How many tokens ?
P1 : 6
O : You cannot retrieve more token than there are in the heap. How many tokens ?
P1 : 3
C : Number of token in the heaps : 12 2 3
C : Player1 which heap ?
P2 : 1
C : How many tokens ?
P2 : 0
C : You must take at least one token. How many tokens ?
P2 : 2
C : Number of token in the heaps : 12 0 3
C : Player1, which heap ?
P1 : 0
```

...

The program stops as soon as the heaps are all empty and displays the number of the winner 1 or 2.

Annex 2 : Example of computation of the sum of Nim

We consider a game of Nim where N = 3, with number of tokens nbp_0 = 1, nbp_1 = 5 and nbp_2 = 9. The binary representation on 4 bits of those numbers is respectively: b_0 = 0001, b_1 = 0101 et b_2 = 1001.

The sum of Nim of those three numbers is :

0001

0101

1001

1101

We get a 1 in a column each time the number of 1 in this column is odd. The sum of Nim is therefore 1101, or 13 in base 10. We note *sumNim* = 1101.

If we compute the sum of Nim of 1010 and 1010 we get : 0000

Annex 3 : Example of choice for the first move

In the state b_0 = 0001, b_1 = 0101 and b_2 = 1001, the sum of Nim is : *sumNim* = 1101. The state is therefore at the advantage of the current player. We must choose the heap and the number of tokens to retrieve such that the opponent will have a state with a sum of Nim equal to 0.

Computation of the sxi :

sx_0 = *sumNim* XOR b_0 sx_0 = 1101 XOR 0001 = 1100 > b_0

sx_1 = *sumNim* XOR b_1 sx_1 = 1101 XOR 0101 = 1000 > b_1

sx_2 = *sumNim* XOR b_2 sx_2 = 1101 XOR 1001 = 0100 < b_2

The heap to choose is the number k = 2 because $sx_2 < b_2$.

The number of token to retrieve is $n_2 = b_2 - sx_2$ = 1001 – 0100 = 0101

After having played this move, the new state will be : b_0 = 0001, b_1 = 0101 et b_2 = 1001 – 0101 = 0100.

The corresponding sum of Nim is equal to 0. The opponent will therefore not have an advantage.