Exam of Computer Science
SCAN 1st year - April 2021

Total duration : *1h30*
Allowed materials : *None.*
Smartphones are not allowed

- The graduation can still change.
- The assignment is on 9 pages.
- All the questions are independant. If it is necessary for A to be solved to solve B, then you can do as if you had solved A (and clearly indicate your assumption) to solve B.

Thought of the day:
« Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live. » *Martin Golding*

Indicative marking scheme :

- Exercise 1 : 4 points
- Exercise 2 : 8 points
- Exercise 3 : 8 points

Before you start :

- parts are independent. You can answer them in the order of your choice.
- the subject is long. Take some time to read everything before you start.
- the cleanliness of your copy will be taken into account.

1 Program understanding (4pts)

We assume that the method `afficheTab` displays the content of an array line by line.

(Q1.1) What is displayed by the following program ?

```java
public class Exo1 {
   public static void main(String args[]){
      int[][] t1 = { { 1 , 2 , 3 } , { 4 , 5 , 6 } , { 7, 8 , 9 } };
      int[][] t2 = method1(t1);
      afficheTab(t2);

      t2 = t1;
      displayArray(t2);
      method2(t2);
      displayArray(t2);
      displayArray(t1);

      int[][] t3 = method3();
      displayArray(t3);
   }

   public static int[][] method1 (int[][] tab) {
      int tab2[][] = new int[3][3];
      for(int i = 0; i < tab.length; i++){
         for(int j = 0; j < tab[i].length; j++){
            if(i > j){
               tab2[i][j] = 0;
            } else {
               tab2[i][j] = tab[i][j];
            }
         }
      }
      return tab2;
   }

  public static void method2 (int[][] tab) {
      for(int i = 0; i < tab.length; i++){
         tab[i][i] = 0;
      }
   }

   public static int[][] method3(){
      int t3[][] = new int[3][];
      for(int i = 0; i < t3.length; i++){
         t3[i] = new int[i+1];
         for(int j = 0; j < i + 1; j++){
            t3[i][j] = j + 1;
         }
      }
      return t3;
   }
}
```

```
1 2 3
0 5 6
0 0 9

1 2 3
4 5 6
7 8 9

0 2 3
4 0 6
7 8 0

0 2 3
4 0 6
7 8 0

1
1 2
1 2 3
```

2 Square arrays (8 points)

The Exo2 class considers the square grid of a game represented as an array with n lines and n columns that contains integer values between 1 and $n \times n$. In a first time, we will consider a 1D array of size $n \times n$ and we will write the methods required for the following steps.

(Q2.1) Complete the `creation1DArray` method that has the following signature. This method creates an array of integers that contains integer values from 1 to `length` in the successive=. For instance, if `length` is equal to 9, the array is $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$

```java
public static int[] creation1DArray (int length)
```

```java
public static int[] creationTab1D (int longueur){
int[] tab1D = new int[longueur];
for( int i = 0; i < longueur; i++) {
tab1D[i] = i + 1;
}
return tab1D;
}
```

(Q2.2) Complete the `exchange` method that has the following signature. This method exchange the content of two cells at positions `pos1` and `pos2`.

```java
public static void exchange (int pos1 , int pos2 , int[] tab)
public static void echange (int pos1, int pos2, int[] tab){
int temp = tab[pos2];
tab[pos2] = tab[pos1];
tab[pos1] = temp;
}
```

(Q2.3) Provide the signature and the implementation of the `shuffle` method that takes as parameter an array of integer `tab` and that uses the previous method two exchange two randomly selected cells. This random exchange is repeated N times, where N is the length of the array `tab`.

Reminder : the method ***`Math.random()`*** *generates a random number in* $[0;1[$

```java
public static void melange (int[] tab){
for( int i = 0; i < tab.length; i++) {
int pos1 = (int) (Math.random() * tab.length);
int pos2 = (int) (Math.random() * tab.length);
echange (pos1, pos2, tab);
}
}
```

(Q2.4) Provide the signature and the implementation of `creation2DArray` method that takes as parameter a 1D array and returns a 2D array. Without verification, we assume that the 1D array is of size $n \times n$ such that the returned 2D array has exactly n lines and n columns. The value of the 1D array will be stored line by line in the 2D array. For instance, if the 1D array is $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$, the 2D array will be $\{\{1, 2, 3\}, \{4, 5, 6\}, \{7, 8, 9\}\}$.

Reminder : the method ***`Math.sqrt(x)`*** *computes the square root of* ***`x`***

```java
public static int[][] creationTab2D (int[] tab){
int n = (int)(Math.sqrt(tab.length));
int[][] tab2D = new int[n][n];
for( int i = 0; i < tab2D.length; i++) {
for( int j = 0; j < tab2D[i].length; j++ ){
tab2D[i][j] = tab[i*n + j];
}
}
return tab2D;
}
```

(Q2.5) Write the `main` method that uses the previous methods to create and square arrray with n lines and n columns that contains integer values between 1 and $n \times n$, each value appearing only once and its position being random. The value of n is directly set in the program with for instance : `int n = 3;`

```
public class Exo2 {
public static void main(String[] args) {
int n = 3;
int[] t1 = creationTab1D(n*n);
melange(t1);
int[][] t2 = creationTab2D(t1);
}
}
```

We are now focusing on square grids that are **magic** : each number between 1 and $n \times n$ is included only once, and the sum of all the lines/columns and diagonals is identical. For instance, the following grid is magic because each sum is equal to 15, and all the numbers from 1 to 9 are included :

4	9	2
3	5	7
8	1	6

(Q2.6) Provide the signature and the implementation of the `verifSumLines` method. This method takes as parameter a 2D array of integers and return the boolean value true if the sums of each lines are identical.

```
public static boolean verifSommeLignes(int[][] tab) {
int sommeLigne0 = sommeLigne(tab,0);
boolean verifSommeLigne = true;
for ( int i = 1; i < tab.length; i++) {
if (sommeLigne(tab , i) != sommeLigne0) {
verifSommeLigne = false;
}
}
return verifSommeLigne;
}

public static int sommeLigne(int[][] tab , int l) {
int sommeL = 0;
for ( int i = 0; i < tab[l].length; i++ ) {
sommeL = sommeL + tab[l][i];
}
return sommeL;
}
```

(Q2.7) BONUS : Provide the signature and the implementation of the `verifUnicity method`. This method takes as parameter a 2D array of integers and returns a boolean value that is true if all the values of the array are different. We assume that all the possible values in this array are in the interval 1 to $n \times n$ for an square array of size n.

```
public static boolean verifUnicite(int[][] tab) {
boolean[] dejaDansTab = new boolean[tab.length*tab.length];
boolean unicite = true;
for ( int i = 0; i < tab.length; i++ ) {
for ( int j = 0; j < tab[0].length; j++ ) {
if (dejaDansTab[tab[i][j] - 1]) {
unicite = false;
}
dejaDansTab[tab[i][j] - 1] = true;
}
}
return unicite;
}
```

3 Object Oriented Programming (8 points)

We want to develop a `Wallet` class to handle the digital wallet of a client that owns various currencies.

```
public class Wallet {
   // Attributes declaration
   double[] money;
   String[] currencies;

   public Wallet(double[] t, String[] d){
      money = t;
      currencies = d;
   }


   public void display(){
      // A REMPLIR
   }

   public void exchange(double[][] rates, int soldCurrency, int boughtCurrency,
       double amountBought){
      // A REMPLIR
   }
}
```

This object is instanciated from two arrays representing the currencies, and the corresponding amount of money. Thus, to represent a wallet containing 3.5€, 10.4$ and 6¥, we create the following instance :

```
String[] currencies = { "EUR", "USD", "YEN" };
```

```java
double[] money = { 3.5 , 10.4, 6 };
Wallet p = new Wallet(money, currencies);
```

(Q3.1) Complete the the code of the `display` method of the Wallet class such that the instruction `p.display()` displays the following result :

```
The wallet contains 3.5 EUR, 10.4 USD, 6 YEN
```

```java
public void afficher() {
System.out.print("Le portefeuille contient ");
for(int i = 0; i < this.devises.length; i++){
System.out.print(this.argent[i] + " " + this.devises[i] + ",");
}
System.out.println();
}
```

(Q3.2) Provide the signature and the implementation of the `mergeWallet` method that, given two wallets, creates a wallet whose content is the sum of the money of the two wallets. **We first assume that the currencies are identical.**

```java
public static Portefeuille fusionnePortefeuille(Portefeuille p, Portefeuille q) {
double[] argent = new double[p.argent.length];
String[] devises = new String[p.devises.length];
for(int i = 0; i < argent.length; i++){
argent[i] = p.argent[i] + q.argent[i];
devises[i] = p.devises[i];
}
return new Portefeuille(argent, devises);
}
```

We represent the exchange rates for the currencies as a 2D array :

	EUR	USD	YEN
EUR	1	0.8	0.75
USD	1.2	1	0.9
YEN	1.34	1.05	1

For instance, 1 euro is exchanged against 0.8 Dollars with the current exchange rate.

(Q3.3) Write the code that defines the previous array in a variable named `rate`.

```java
double[][] taux = { {1, 0.8, 0.75}, {1.2, 1, 0.9}, {1.34, 1.05, 1} };
```

We are now interested in the exchange of money from one currency to another. For instance, with the current wallet, we could exchange 2 euros against 1.6 Dollars, and it will thus remain 1.5 Euros and 12 Dollars in the wallet.

(Q3.4) Complete the code of the `exchange` method such that we obtain the following execution on the example :

```
// 0 is for EUR, 1 for USD in the currency array
p.exchange(rate, 0, 1, 1.6);
p.display();
// Le portefeuille contient 1.5 EUR, 12 USD, 6 YEN
```

Before performing the exchange, one will check if the content of the wallet allows it.

```
public void echanger(double[][] taux, int monnaieVendue, int monnaieAchetee,
    double quantiteMonnaieAchetee){
double quantiteMonnaieVendue = quantiteMonnaieAchetee /
    taux[monnaieVendu][monnaieAchetee];
if(quantiteMonnaieVendue > this.argent[monnaieVendue]){
System.out.println("Vous n'avez pas assez d'argent !");
}
else {
this.argent[monnaieVendue] -= quantiteMonnaieVendue;
this.argent[monnaieAchetee] += quantiteMonnaieAchetee;
}
}
```

(Q3.5) Provide the signature and the code of the `equivalent` method that computes the equivalent of the wallet's content in a given currency. For instance, after exchange, the wallet contains 1.5€ that are worth 1.2$, 6¥ that are worth 6.3$, and 12$, thus a total of 19.5$.
We want to obtain the following execution :

```
// 1 is the index of USD in the currency table
p.equivalent(rate, 1);
// -> the wallet contains the equivalent of 19,5 USD
```

```
// Version avec affichage
public void equivalent(double[][] taux, int monnaieCible){
double valeur = 0;
for(int i = 0; i < this.argent.length; i++){
valeur += this.argent[i] * taux[i][monnaieCible];
}

System.out.println("Le portefeuille contient l'equivalent de " + valeur + " " +
    this.devises[monnaieCible]);
}

// Version sans affichage
public double equivalent(double[][] taux, int monnaieCible){
double valeur = 0;
for(int i = 0; i < this.argent.length; i++){
valeur += this.argent[i] * taux[i][monnaieCible];
}

return valeur;
}
```

With the exchage rates presented above, we can exchange 1¥ against 1.34€, then these 1.34€ into 0.75*1.34 = 1.005 ¥, thus allowing us to have more money than at the beginning.

(Q3.6) Provide the signature and the code of the `easyMoney` method of the `Wallet` class that finds if there is an exchange that can increase the amount of money. The method will display such interesting exchange if it exists, by displaying "we must exchange EUR into YEN !" for the example ; and "no interesting exchange" if none is possible.

```
public void argentFacile(double[][] taux){
boolean trouve = false;
for(int i = 0; i < this.argent.length; i++){
for(int j = 0; j < i; j++){
if(taux[i][j] * taux[j][i] > 1){
trouve = true;
System.out.println("Il faut echanger " + this.devises[i] + " et " +
    this.devises[j]);
}
}
}

if(!trouve){
System.out.println("Pas d'echange interessant");
}
}
```

(Q3.7) BONUS : Provide a new version of the `mergeWallet` method that does not assume that currencies are in the same order.

```
public static Portefeuille fusionnePortefeuille(Portefeuille p, Portefeuille q) {
double[] argent = new double[p.argent.length];
String[] devises = new String[p.devises.length];
for(int i = 0; i < devises.length; i++){
devises[i] = p.devises[i];
double argent = p.argent[i];
for(int j = 0; j < q.devises.length; j++){
if(p.devises[i].equals(q.devises[j])){
argent += q.argent[j];
}
}
argent[i] = argent;
}
return new Portefeuille(argent, devises);
}
```