

420-906-RO

Algorithmes et programmation structurée

Groupe 38453

Examen final

(remettez vos résultats en un ou plusieurs fichiers .java, .txt, .pdf, .odt ou .docx)

(vous avez droit à toutes vos notes de cours, aux livres, ainsi qu'à l'internet, mais vous ne pouvez pas communiquer avec d'autres personnes.)

Vendredi 29 avril 2022

17h00 à 19h30

(mais vous pouvez prendre jusqu'à 22h00 pour le faire)

Note maximale : 20/20

Poids dans le bulletin : 30%

Q1. La couverture des lignes de code [3 points]

Soit les exigences suivantes :

- EF1. Le logiciel doit saisir une valeur entière au clavier.
- EF2. Le logiciel doit déterminer l'une des quatre situations suivantes :
 - Est-ce que la valeur entrée est divisible par deux et par trois ?
 - Est-ce que la valeur entrée est divisible par deux, mais pas par trois ?
 - Est-ce que la valeur entrée est divisible par trois, mais pas par deux ?
 - Est-ce que la valeur entrée n'est pas divisible ni par trois, ni par deux ?
- EF3. Le logiciel doit afficher un message approprié selon la situation déterminée.

Votre collègue a rédigé le code source suivant afin de répondre à ces exigences.

```
- package finalq1;
-
- import java.util.Scanner;
-
- public class FinalQ1 {
-
-     public static void main(String[] args) {
-
-         Scanner lecteur = new Scanner(System.in);
1         System.out.println("Entrez une valeur entière : ");
2         int valeur = lecteur.nextInt();
-
-         if (valeur%2==0) {
3             if (valeur%3==0) {
4                 System.out.println("La valeur est divisible par 2 et par 3.");
5             }
6             else {
7                 System.out.println("La valeur est divisible par 2 mais pas par 3.");
8             }
9         }
10        else if (valeur%3==0) {
11            System.out.println("La valeur est divisible par 3 mais pas par 2.");
12        }
13        else {
14            System.out.println("La valeur est divisible par 2 et par 3.");
15        }
16        lecteur.close();
17    }
18 }
```

Vous devez rédiger un jeu de tests ayant le moins de cas de test que possible mais qui assure une couverture des lignes de code exécutables (*statement coverage*) de 100%. Les lignes de code exécutables sont celles numérotées dans la colonne de gauche de l'encadré ci-haut. Vos tests ne sont pas obligés de découvrir des bogues, s'il y en a dans ce code.

Vos cas de test doivent être rédigés en utilisant la forme ci-dessous, basé sur un cas de test rédigé pour un autre logiciel :

Entrée	Sortie attendue	Sortie obtenue	Résultat
55	12345	54321	Échec

Q1a. Les cas de tests donnés permettent d’avoir une couverture des lignes de code exécutables (*statement coverage*) de 100%, c’est-à-dire que chaque ligne de code est exécutée au moins une fois. [0.75 points]

Q1b. Le nombre de cas de test donné est le minimum nécessaire afin d’avoir une couverture du code de 100%. [0.75 points]

Q1c. Dans le tableau, le contenu des colonnes « entrées » et « sorties attendues » est correct. [0.75 points]

Q1d. Dans le tableau, le contenu des colonnes « sorties obtenues » et « résultats » est correct. [0.75 points]

Q2. Les tableaux [3 points]

Soit le code ci-dessous, qui a pour but de créer et remplir un tableau de valeurs au carré, et un autre tableau de valeurs au cube.

```
package finalq2;

public class FinalQ2 {

    public static void main(String[] args) {
        int[] carres = new int[10];
        int[] cubes = new int[5];

        for(int i=0 ; i<carres.length ; i++) {
            carres[i] = i*i;
            cubes[i] = i*i*i;
        }
    }
}
```

Ce code compile, mais il plante à chaque exécution.

Q2a. Pourquoi ce problème se produit-il ? [1.5 points]

Q2b. Comment pourrait-on corriger ce problème afin que le tableau « carres » soit rempli de valeurs au carré, et que le tableau « cubes » soit rempli de valeurs au cube ? Vous n'avez pas à remettre un code corrigé, seulement d'expliquer votre solution. [1.5 points]

Q3. Les procédures et fonctions [3 points]

Soit le code ci-dessous, qui remplit un tableau de racines carrées (valeurs à la puissance 0.5).

```
package finalq3;

import java.util.ArrayList;

public class FinalQ3 {

    public static void main(String[] args) {
        ArrayList<Float> tableau = new ArrayList<Float>();

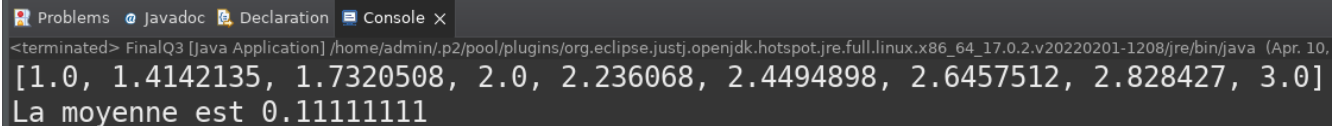
        for(int i=1 ; i<10 ; i++) {
            float valeur = (float) Math.pow(i, 0.5);
            tableau.add(valeur);
        }

        System.out.println(tableau);
        float moy = calculerMoyenne(tableau);
        System.out.println("La moyenne est " + moy);
    }

    public static float calculerMoyenne(ArrayList<Float> valeurs) {
        float total = 0;
        for(float une_valeur : valeurs) {
            total = total + une_valeur;
            float moyenne = total / valeurs.size();
            return moyenne;
        }

        return -1f;
    }
}
```

Ce code compile et s'exécute sans planter. Le problème est que le calcul semble incorrect. Voici l'affichage que l'on obtient :



```
<terminated> FinalQ3 [Java Application] /home/admin/.p2/pool/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.linux.x86_64_17.0.2.v20220201-1208/jre/bin/java (Apr. 10, 2022)
[1.0, 1.4142135, 1.7320508, 2.0, 2.236068, 2.4494898, 2.6457512, 2.828427, 3.0]
La moyenne est 0.11111111
```

Toutes les valeurs étant entre 1 et 3, cela ne fait pas de sens que la moyenne calculée soit inférieure à 1 ...

Q3a. Pourquoi le calcul de la moyenne est-il incorrect ? [1.5 points]

Q3b. Comment pourrait-on corriger ce problème ? Vous n'avez pas à remettre un code corrigé, seulement d'expliquer votre solution. [1.5 points]

Q4. Utilisation du débogueur [3 points]

Soit les trois fragments suivants possédant une ligne qui affiche « ICI ». Pour chaque code source, répondez à la question à choix de réponse en lien avec l'état des variables à l'endroit où se trouve l'affichage de « ICI ». Vous n'avez pas à justifier votre choix de réponse, ni de trouver le bogue s'il y en a un, ni de corriger le problème.

Notez qu'il s'agit de fragments de code qui doivent être mis dans un projet approprié.

Q4a. Soit le fragment de code ci-dessous [1 point].

```
public static void main(String[] args) {
    ArrayList<Integer> diviseurs = new ArrayList<Integer>();

    for(int i=3 ; i>=-3 ; i--) {
        diviseurs.add(i);
    }

    System.out.println("ICI");

    for(int diviseur : diviseurs) {
        double quotient = 100/diviseur;
        System.out.println(quotient);
    }
}
```

Quel est l'état du tableau statique « diviseurs » au moment où le code affiche « ICI ».

- (a) Ce code ne compile pas et ne peut donc pas s'exécuter.
- (b) Ce code plante avant d'atteindre l'affichage du « ICI ».
- (c) Le tableau statique « diviseurs » contient [-3, -2, -1, 0, 1, 2, 3].
- (d) Le tableau statique « diviseurs » contient [3, 2, 1, 0, -1, -2, -3].
- (e) Le tableau statique « diviseurs » n'existe pas au moment de l'affichage du « ICI ».

Q4b. Soit le fragment de code ci-dessous [1 point].

```
public static void main(String[] args) {
    String consonnes = "bcdfghjklmnpqrstvwxyz";
    String voyelles = "aeiouy";
    String mot = "";

    for(int i=0 ; i<voyelles.length() ; i++) {
        mot = mot + consonnes.charAt(i);
        mot = mot + voyelles.charAt(i);
    }

    System.out.println("ICI");
}
```

Quel est l'état de la variable « mot » au moment où le code affiche « ICI ».

- (a) Ce code ne compile pas et ne peut donc pas s'exécuter.
- (b) Ce code plante avant d'atteindre l'affichage du « ICI ».
- (c) La variable « mot » contient "aeiouy".
- (d) La variable « mot » contient "bacedifoguhy".
- (e) La variable « mot » n'existe pas au moment de l'affichage du « ICI ».

Q4c. Soit le fragment de code ci-dessous [1 point].

```
public static void main(String[] args) {
    int v1 = 12;
    int v2 = 5;

    int max = getMax(v1, v2);

    System.out.println("ICI");
}

public static int getMax(int valeur1, int valeur2) {
    if (valeur1 > valeur2)
        return valeur1;
}
```

Quel est l'état de la variable « max » au moment où le code affiche « ICI ».

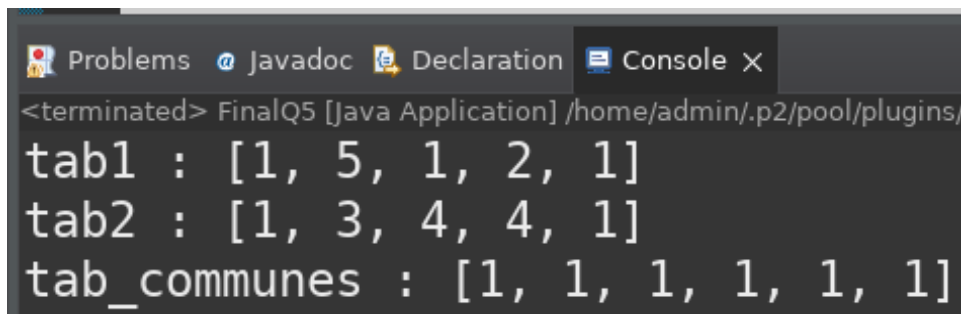
- (a) Ce code ne compile pas et ne peut donc pas s'exécuter.
- (b) Ce code plante avant d'atteindre l'affichage du « ICI ».
- (c) La variable « max » contient la valeur 12.
- (d) La variable « max » contient la valeur 5.
- (e) La variable « max » n'existe pas au moment de l'affichage du « ICI ».

Q5. Programmation [8 points]

Rédigez le code source d'un logiciel qui répond aux exigences suivantes :

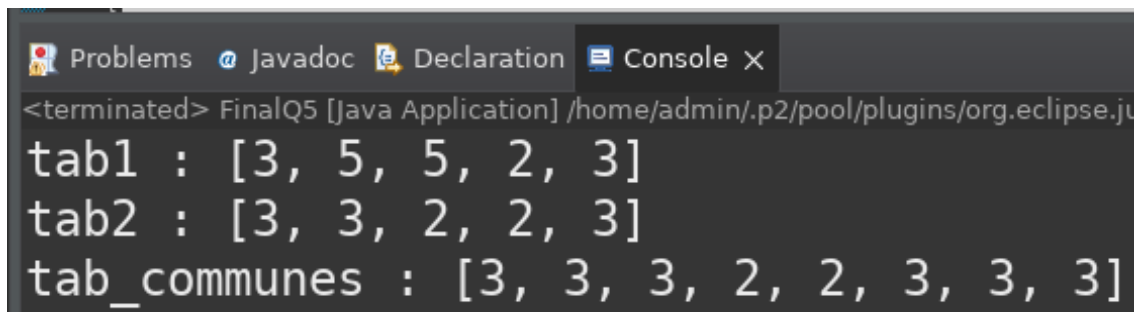
- E01. Le logiciel doit posséder une première fonction qui ne reçoit rien et qui retourne un tableau de cinq valeurs. [1 point]
- E02. Le tableau de cinq valeurs créé par cette première fonction est rempli de valeurs aléatoires entières entre 1 et 5 (≥ 1 et ≤ 5). [1.5 point]
- E03. La première fonction est appelée deux fois pour fabriquer deux tableaux appelés « tab1 » et « tab2 ». [1 point]
- E04. Les tableaux « tab1 » et « tab2 » sont envoyés à une deuxième fonction qui retourne un troisième tableau appelé « tab_communes » contenant les valeurs communes à « tab1 » et « tab2 ». [3 points]
 - Il peut y avoir des doublons dans le tableau « tab_communes », c'est-à-dire qu'une valeur commune peut s'y retrouver à plusieurs reprises.
- E05. Le contenu des trois tableaux « tab1 », « tab2 » et « tab_communes » sont affichés. [0.5 point]
- E06. Les deux fonctions sont documentées en utilisant des commentaires Javadoc appropriés. [1 point]

Voici un exemple d'affichage possible :



```
<terminated> FinalQ5 [Java Application] /home/admin/.p2/pool/plugins/  
tab1 : [1, 5, 1, 2, 1]  
tab2 : [1, 3, 4, 4, 1]  
tab_communes : [1, 1, 1, 1, 1, 1]
```

Voici un autre exemple d'affichage :



```
<terminated> FinalQ5 [Java Application] /home/admin/.p2/pool/plugins/org.eclipse.ju  
tab1 : [3, 5, 5, 2, 3]  
tab2 : [3, 3, 2, 2, 3]  
tab_communes : [3, 3, 3, 2, 2, 3, 3, 3]
```

Qbonus. La fonction [Bonus +2 points]

La racine-carré de 2 est une valeur irrationnelle, comme la valeur de Pi et de la constante d'Apéry. Il est possible de calculer une approximation de la racine-carrée de 2 en utilisant la formule suivante :

$$approximation_n = 2 + \frac{1}{approximation_{n-1}}$$

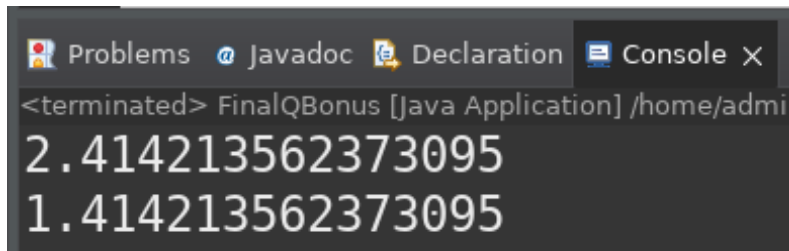
Il s'agit donc d'une formule qui est récursive, un peu comme le calcul de la valeur de Fibonacci, c'est-à-dire que le calcul de la $n^{ième}$ valeur dépend de la valeur précédente $n-1$. La valeur initiale, c'est-à-dire le « $approximation_0$ », est de 1.

Notez que cette approximation donne une unité de trop. Le résultat sera 2.4142135... alors que la racine-carrée de deux est 1.4142135...

Il existe aussi une manière de calculer la racine carrée de deux qui n'est pas récursive, mais pour cet exercice, vous devez utiliser la récursivité. Ainsi, pour une procédure « `main()` » qui serait la suivante :

```
public static void main(String[] args) {  
    double resultat = calculerRacineCarreeDeDeux(100);  
  
    System.out.println(resultat);  
    System.out.println(resultat - 1);  
}
```

L'affichage devrait ressembler au suivant :



```
<terminated> FinalQBonus [Java Application] /home/admin  
2.414213562373095  
1.414213562373095
```

Qbonus. Votre code source permet de calculer la racine carré de deux de manière récursive. [+2 points bonus]