Name/Nom : .....

Klasse/Classe: .....

Hes·so VALAIS WALLIS
Haute Ecole d'Ingénierie
Hochschule für Ingenieurwissenschaften

TEST ANNUEL - ÉVALUATION FORMATIVE

SOLUTION

Informatique 1 | Informatik 1

Consigne :
Cet examen est à but formatif uniquement, ceci afin de que vous puissiez vérifier vos connaissances en programmation en fin d'année. Cela signifie que vous ne recevrez pas de note pour cet examen. Il ne sera pas non plus corrigé pour vous.

Ceci dit, cet examen est écrit comme un examen standard et je vous propose de le passer dans les mêmes conditions, c'est-à-dire en le faisant par vos propres moyens, avec vos propres connaissances. N'oubliez pas qu'un examen sert à vérifier votre niveau d'apprentissage et, normalement, vous réconforter dans le fait que vous avez appris beaucoup de choses durant ces quelques mois passés ensemble.

Vous avez droit pour cet examen à un aide-mémoire de 4 pages (2 feuilles recto-verso). Aucun moyen électronique n'est permis.

Question	Points	Bonus	Score
Short questions	4	0	
True or false	6	0	
Recursively listing files	5	0	
Electric streams ↯	10	0	
MMORPG	10	0	
Total:	35	0	

This exam has 5 questions, for a total of 35 points.

Rev 1.10

Question 1 – Short questions (4 points)

Cette question est séparée en plusieurs exercices indépendants. Le nombre de points pour chaque exercice est indiqué dans la marge.

[1 Pt] (a) Qu'est-ce qu'une classe abstraite ? Expliquez théoriquement et donnez un exemple.

Solution: An abstract class is a class which can not be instantiated. Its purpose is to provide some functionality (attributes, methods) and force the class that will inherit from it to implement some of its characteristics. For example, an abstract class `Shape` could force all its children to implement the method `getSurface`.

[1 Pt] (b) Que vaut `a` après l'exécution du code suivant ?

```
1 int a = 12;
2 float b = a % 5;
3 b += (0.4 * b);
4 a = (int)(a / b);
```

(b) **4**

[1 Pt] (c) Qu'affiche le code suivant ?

```
1 for (char a = 'z'; a > 'a'; a -= 7)
2   System.out.print("" + a + 'a');
```

Solution:

```
zasalaea
```

[1 Pt] (d) Soit :

```
1 int method(int n){
2   if (n == 1 || n == 0)
3     return n;
4   else
5     return method(n-1) + method(n-2);
6 }
```

Est-ce que la complexité de la méthode ci-dessus est linéaire ? Expliquez votre démarche.

Solution: The complexity of this method is worse than linear. If it were linear, the number of calls would rise linearly with the size of the problem (which is n in our case). In fact, the complexity is $\mathcal{O}(2^n)$ because :

$$\begin{aligned} 1 + 2 + 4 + ... + (n-1) &= 1((2^n) - 1)/(2-1) \\ &= 2^n - 1 \end{aligned}$$

Question 2 – True or false (6 points)

	True	*False*
Il n'est pas possible de faire une boucle infinie avec un *for*.	☐	⊗
Une classe peut implémenter 2 interfaces.	⊗	☐
Une classe abstraite ne peut pas contenir l'implémentation d'une méthode.	☐	⊗
Plusieurs classes peuvent hériter de la même classe.	⊗	☐
Un vecteur est bien plus efficace qu'une liste chaînée pour y insérer des éléments au début.	☐	⊗
Un algorithme de complexité $\mathcal{O}(5)$ est plus efficace qu'un algorithme en $\mathcal{O}(n^2)$ pour un grand nombre d'éléments.	⊗	☐
Dans une structure *try-catch*, la partie *catch* est exécutée quand il y a eu un problème de compilation dans la partie *try*.	☐	⊗
Une méthode récursive est toujours plus efficace qu'une méthode itérative (boucle).	☐	⊗
La méthode `Math.cos` est une méthode statique.	⊗	☐
Une méthode abstraite ne peut exister que dans une classe abstraite.	⊗	☐
Pour hériter d'une classe abstraite, on utilise le mot clef *implements*.	☐	⊗
Si la classe A hérite de la classe B et la classe B hérite de la classe C, A n'hérite pas des attributs définis dans C.	☐	⊗

Question 3 – Recursively listing files (5 points)

La classe `File` fournie avec Java permet d'accéder de manière uniforme aux fichiers ainsi qu'aux répertoires. À l'aide de cette classe et de récursion, implémentez la méthode `getAllFiles` qui permet de lister les fichiers présents dans un répertoire ainsi que dans tous ses sous-répertoires. Votre méthode doit posséder le prototype suivant (avec `where` qui est l'endroit où commence la recherche):

```
static LinkedList<File> getAllFiles(File where)
```

Ainsi, si le dossier `/tmp` contient les fichiers suivants:

```
|-- Bonjour.java
|-- a
|   |-- b
|   |   |-- Hi.java
|   |   |-- Ola.java
|   |-- Hello.java
```

Alors le code

```
1 for (File f : listFiles(new File("/tmp"))) {
2   System.out.println(f.getAbsoluteFile());
3 }
```

affichera

```
/tmp/Bonjour.java
/tmp/a/b/Hi.java
/tmp/a/b/Ola.java
/tmp/a/Hello.java
```

Pour vous aider, voici quelques méthodes utiles de la classe `File`:

- `File[] listFiles()` retourne un tableau contenant les fichiers et les répertoires dans le `File` courant (qui doit donc être un répertoire).
- `boolean isDirectory()` retourne `true` si le `File` est un répertoire.
- `boolean isFile()` retourne `true` si le `File` est un fichier et non un répertoire.
- `String getAbsoluteFile()` retourne un `String` représentant le chemin absolu du fichier.

Voici également quelques méthodes utiles de la classe `LinkedList`:

- Pour créer une liste de `String` : `LinkedList<String> l = new LinkedList<String>();`
- Ajouter un élément à une liste : `l.add("elem")`
- Ajouter tous les éléments d'une liste à une autre liste : `l.addAll(anotherList);`

Solution:

```
1  static List<File> listFiles(File where) {
2    List<File> l = new LinkedList<File>();
3    for (File f : where.listFiles()) {
4      if (f.isFile()) {
5        l.add(f);
6      }
7
8      if (f.isDirectory()) {
9        l.addAll(getAllFiles(f));
10     }
11   }
12
13   return l;
14 }
```

Question 4 – Electric streams ⚡ (10 points)

Un compteur électrique est connecté à un ordinateur. Il mesure à intervalle régulier la consommation de divers appareils ainsi que différentes informations comme le courant ainsi que la fréquence de la ligne. Pour réaliser un programme informatique utilisant de telles mesures, nous proposons la classe suivante:

```
1  public class Measure {
2    double power;
3    int frequency;
4
5    public Measure(double p, int freq){
6      power = p;
7      frequency = freq;
8    }
9  }
```

[2 Pt] (a) Nous aimerions pouvoir écrire le code suivant:

```
1  Measure m = new Measure(3);
2  System.out.println(m);
```

Modifiez le moins de code possible pour que ce code soit valide et produise le résultat suivant sur la console (en partant du principe que par défaut la fréquence vaut 50) :

```
Measure : 3.0 [W] @ 50 [Hz]
```

Solution:

```
1  public Measure(double w){
2      this(w, 50);
3  }
4
5  public String toString(){
6    return "Measure : " + watts + " [W] " + "@ " + frequency + " [Hz]";
7  }
```

Les valeurs sont stockées dans un fichier selon le format défini ci-dessous :

```
1  #,3,500
2  #,10,510
3  #,?,40,?,?,500
4  #,?,?,?,12,500
```

La première ligne correspond à une mesure de 3 watts à 50 Hz, la seconde ligne à une mesure de 10 watts à 51 Hz (notez que les valeurs en Hz sont multipliées par 10 dans le fichier). Chaque fichier contient un nombre variable de mesures et à chaque intervalle de mesure, une nouvelle ligne est ajoutée dans ce fichier. Chaque ligne doit commencer par le caractère # et les différentes valeurs sont séparées par des virgules.

Comme vous le voyez à la ligne 3 et 4, il se peut que le fichier contienne des caractères `?` qui correspondent à des valeurs de puissance indisponibles qui doivent être ignorées. Même en présence de mesures inconnues, la puissance vient toujours avant la fréquence et, dans tous les cas, la fréquence apparaît toujours en dernier.

Pour rappel, la méthode `split` de la classe `String` permet de découper un `String` à l'aide d'un caractère donné. Le type de résultat est un tableau de `String`.

Exemple : `"a-lo-ha".split("-") -> {"a", "lo", "ha"}`

[2 Pt] (b) Il se peut que des lignes de mesures ne soient pas valides. Les deux problèmes possibles sont que la ligne ne commence pas par # ou que la ligne ne contient que des valeurs inconnues en raison d'un problème de capteurs. Notez que l'on fait l'hypothèse que si une valeur numérique est présente dans la ligne, la seconde le sera également. Implémentez la fonction `isValid` qui prend comme paramètre une ligne du fichier de mesure et retourne `true` lorsque celle-ci est valide.

Solution:

```
1  public static boolean isValidAlternate(String m){
2    if(m.charAt(0) != '#' || m.charAt(m.length()-1) == '?')
3      return false;
4
5    return true;
6  }
```

other solution

```
1  public static boolean isValid(String m){
2    if(m.length() == 0 || m.charAt(0) != '#')
3      return false;
4
5    String[] tbl = m.split(",");
6
7    // We need at least 3 values split (#, val1, val2)
8    if (tbl.length < 3)
9      return false;
10
11   // Check if only ? are present
12   for(int i = 1; i < tbl.length; i++){
13     if(tbl[i].equals("?") == false)
14       return true;
15   }
16
17   return false;
18 }
```

[3 Pt] (c) Implémentez maintenant une fonction `decodeMeasure` qui doit retourner le contenu d'une mesure complète en décodant une ligne que l'on supposera valide. La valeur retournée est de type `Measure`.

Solution:

```
1  public static Measure decodeMeasure(String input) {
2    String[] t = input.split(",");
3
4    int power = 0;
5    int freq = 0;
6
7    for(int i = 1; i < t.length-1; i++){
8      if(t[i].contains("?"))
9        continue;
10
11     power = Integer.parseInt(t[i]);
12   }
13
14   freq = Integer.parseInt(t[t.length-1]) / 10;
15   return new Measure(power, freq);
16 }
```

[3 Pt] (d) Implémentez finalement une méthode nommée `decodeFile` qui prend en argument le nom d'un fichier de mesures, l'ouvre, et insère la mesure de chaque ligne valide dans un vecteur qui est retourné à la fin.

Solution:

```java
public static Vector<Measure> decodeFile(String fn) {
  Vector<Measure> measures = new Vector<Measure>();

  try {
    BufferedReader input = new BufferedReader(new FileReader(fn));
    String line = input.readLine();
    while (line != null) {
      if(isValid(line)){
        Measure m = decodeMeasure(line);
        measures.add(m);
      }

      line = input.readLine();
    }
    input.close();
  } catch (IOException e) {
    e.printStackTrace();
  }

  return measures;
}
```

Question 5 – MMORPG (10 points)

Une firme informatique a décidé de créer un nouveau MMORPG (Multi-Players Online Roleplaying Game) et une structure a été a été spécifiée pour réaliser les différentes unités du jeu.

[2 Pt] (a) Écrivez l'implémentation de la classe `Unit` qui représente le sommet de la hiérarchie. Cette classe comporte quatre attributs, le nom de l'unité (`name`), l'expérience (`XP`), les points de vie (`HP`) et les points d'armure (`armor`). Les trois derniers attributs, stockés sous forme de `int`, doivent avoir des valeurs par défaut : 0 pour l'expérience, 10 pour les points de vie et 5 pour l'armure. Cette classe, qui ne doit pas être instanciable, doit aussi comporter un constructeur qui permet de spécifier le nom de l'unité.

Solution:

```java
public abstract class Unit {
  public String name;
  public int XP = 0;
  public int HP = 10;
  public int armor = 5;

  public Unit(String name) {
    this.name = name;
  }
}
```

[3 Pt] (b) Écrivez maintenant une classe `MovingUnit` qui hérite de la classe `Unit`. Cette classe, qui elle aussi ne doit pas être instanciable, possède un attribut entier supplémentaire nommé `MP` pour représenter le nombre de points de mouvement. De plus, cette classe doit faire en sorte que toutes les classes qui hériteront de celle-ci seront obligées d'avoir une procédure `move` qui prend deux entiers en paramètres.

Solution:

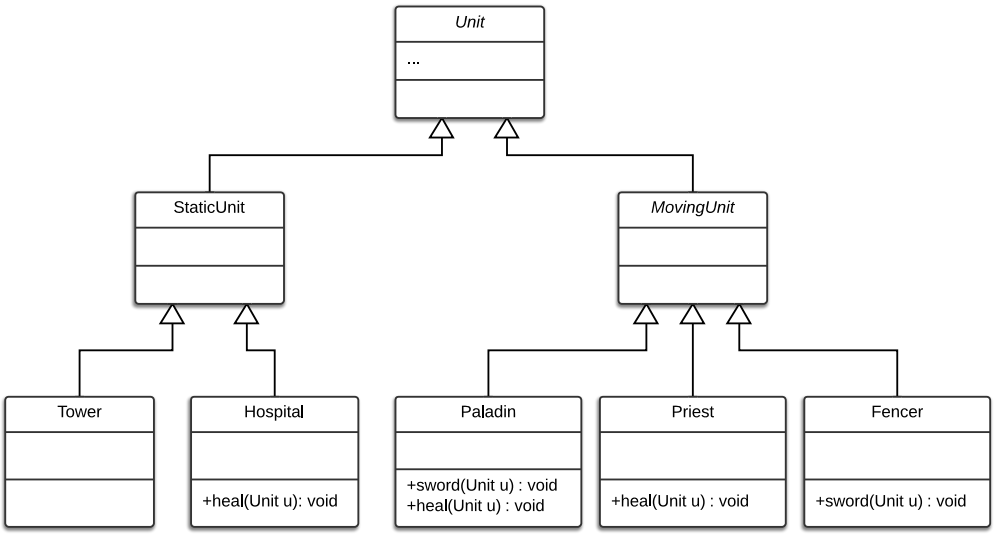
```java
public abstract class MovingUnit extends Unit {
  int MP = 1;

  public MovingUnit(String name) {
    super(name);
  }

  abstract public void move(int x, int y);
}
```

[1 Pt] (c) La structure UML ci-dessus représente une partie de la hiérarchie des unités du jeu. Cela permet de voir un peu mieux la structure d'héritage des classes désirée. Il est à noter que la description des classes est **incomplète**.

On voudrait maintenant rajouter des spécifications pour deux catégories d'unités, celles qui soigent (`Healer`) et qui doivent posséder une méthode nommée `heal` qui reçoit une unité en paramètre. Des exemples de classes dans le diagramme ci-dessus sont `Hospital`, `Priest` ou `Paladin`. On voudrait aussi spécifier celles qui se sont des guerriers (`Warrior`) et qui doivent posséder une méthode `sword` qui



reçoit aussi une unité en paramètre (p. ex. `Paladin` ou `Fencer`). Il y a donc que certaines classes qui sont `Healer` ou `Warrior`), ou les deux. Ces classes peuvent être autant dans la catégorie des `MovingUnit` ou des `StaticUnit`. Expliquez en une phrase quelle solution pourrait être utilisée pour créer ces spécifications.

Solution: To create the transversal specifications, interfaces could be used.

[2 Pt] (d) Écrivez maintenant le code qui permet de créer ces spécifications.

Solution:

```
1  public interface Warrior {
2    public void sword(Unit unit);
3  }
4
5  public interface Healer {
6    public void heal(Unit unit);
7  }
```

[2 Pt] (e) Écrivez maintenant une classe `Paladin` qui est une unité mobile (`MovingUnit`), et qui doit aussi remplir les spécifications des soigneurs (`Healer`) et des guerriers (`Warrior`). La valeur des points de vie (`HP`) pour les paladins doit être de 20 et celle de l'armure (`armor`) doit être de 10. Écrivez aussi toutes les méthodes que cette classe doit implémenter. Il n'est pas nécessaire d'écrire le corps des méthodes.

Solution:

```
1  public class Paladin extends MovingUnit implements Warrior, Healer {
2
3    public Paladin(String name) {
4      super(name);
5      this.armor = 10;
6      this.HP = 20;
7    }
8
9    public void heal(Unit unit) {
10   }
11
12   public void sword(Unit unit) {
13   }
14
15   public void move(int x, int y) {
16   }
17 }
```