Algorithmes et Pensée Computationnelle

Consolidation POO + Classes abstraites et interfaces

Les exercices sont construits autour des concepts d'héritage, de classes abstraites et d'interfaces. Au terme de cette séance, vous devez être en mesure de différencier une classe abstraite d'une interface, savoir à quel moment utiliser l'un ou l'autre, factoriser votre code afin de le rendre mieux structuré et plus lisible.

Cette feuille d'exercices est divisée en 3 sections. La première portant sur des notions de bases en POO, la deuxième sur les classes abstraites et la dernière sur les interfaces.

Les exercices de cette feuille d'exercices doivent être faits uniquement en Java.

Le code présenté dans les énoncés se trouve sur Moodle, dans le dossier Code.

1 Consolidation - Programmation Orientée Objet

Question 1: (**1**) *10 minutes*) Encapsulation - Java

L'encapsulation sert à cacher les détails d'implémentation. Elle sert uniquement à montrer que les informations essentielles aux utilisateurs.

En Java, il est recommandé de déclarer les attributs comme étant **private** et de mettre à disposition des utilisateurs des méthodes publiques d'accès afin qu'ils puissent accéder ou modifier la valeur des attributs privés.

Les méthodes publiques d'accès comme getName et setName doivent être nommées avec soit get soit set suivi du nom de l'attribut avec la 1ère lettre en majuscule (Java Naming convention). Le mot-clé this fait référence à l'objet en question.

- 1. Dans votre IDE, créez une classe Person,
- 2. Ajoutez-y un attribut privé name,
- 3. Créez un getter et un setter pour l'attribut name en suivant la convention de nommage des méthodes.
- 4. Dans votre main, créez une instance de Person,
- 5. En utilisant le setter défini précédemment, donnez un nom (name) à votre instance.
- 6. Affichez le nom de l'instance en utilisant le getter de l'attribut name.

```
>_ Solution
    public class Main {
       public static class Person {
 3
         private String name;
 4
 5
6
         public String getName() {
           return name;
7
8
9
         public void setName(String newName) {
10
           this.name = newName;
11
12
13
       public static void main(String[] args) {
14
         Person myObj = new Person();
15
16
         myObj.setName("John");
17
         System.out.println(myObj.getName());
18
      }
19
    }
```

Question 2: (10 minutes) Héritage - Java

Comme vous le savez, il est possible que des classes-filles héritent des attributs ou méthodes de classesmères. En Java, il faut utiliser le mot-clé extends lorsqu'on défini une classe-fille. Ainsi, l'héritage permet la réutilisation des attributs et méthodes d'une classe existante.

- 1. Créez une classe-mère Vehicle ayant un attribut protégé appelé brand.
- 2. Créez un constructeur pour la classe Vehicle et assignez une valeur à l'attribut brand.
- 3. Définissez une méthode honk qui affiche "Tuut, tuut!"
- 4. Créez une classe-fille Car qui hérite de Vehicle et ayant pour attribut modelName avec pour valeur par défaut "Mustang".

Conseil

Dans le constructeur de la classe-fille, n'oubliez pas de faire appel au constructeur de la classe-mère en utilisant le mot-clé super.

>_ Solution public class Main { public static class Vehicle { 3 protected String brand; 4 5 public Vehicle(String brand) { 6 this.brand = brand; 7 8 9 public void honk() { 10 System.out.println("Tuut, tuut!"); 11 12 13 public static class Car extends Vehicle{ 14 15 public String modelName = "Mustang"; 16 public Car(String brand) { 17 super(brand); 18 19 } 20 21 public static void main(String[] args) { 22 Car myCar = new Car("Ford"); 23 myCar.honk(); 24 System.out.println(myCar.brand + ""+myCar.modelName); 25 26

Question 3: (**Q** *30 minutes*) Surcharge de méthodes, égalité et identité - Java En Java, créez une classe **Fraction** ayant pour attributs privés un numérateur et un dénominateur.

Conseil

Vous pouvez vous inspirer des solutions d'exercices de la semaine 8.

Assignez des valeurs à vos attributs dans le constructeur que vous définirez. Dans le corps de votre classe, effectuez les opérations suivantes :

— Définir une méthode nommée gcd qui prend comme arguments les deux attributs définis précédemment et qui retourne leur plus grand diviseur commun. Pour cela, vous devez implémenter l'algorithme d'Euclide de façon récursive.

Conseil

L'algorithme d'Euclide fonctionne comme suit :

- Si A = 0 alors gcd(A, B) = B.
- Si B = 0 alors gcd(A, B) = A.
- Si aucune des conditions ci-dessus n'est remplie, faire appel à gcd de façon récursive avec pour premier argument B et pour deuxième R avec R = A%B. R étant le reste de la division entre A et B.

- Définir une méthode simplify qui sera appelée dans le constructeur après l'initialisation des attributs.
- Redéfinir la méthode toString pour produire une représentation textuelle des instances de Fraction.
- Redéfinir la méthode equals pour tester l'égalité entre deux fractions. Cette méthode prendra comme argument un objet other de type Object.

Conseil

En Java, par défaut toutes les classes héritent d'une classe-mère nommée Object

— Dans la méthode equals, vérifier que l'instance Fraction (this) n'est pas identique à l'objet other. Si les deux objets sont identiques, renvoyer true. Vérifier également que les deux objets sont de même type. Dans le cas contraire, renvoyer false.

Conseil

En Java, pour vérifier que deux objets sont de même type, vous pouvez accéder à la classe de chacun des éléments en utilisant la méthode .getClass()) et les comparer.

— Dans le main, exécuter le code suivant (disponible sur Moodle) :

```
public class Main {
      public static void main(String[] args) {
2
3
         Fraction f1 = new Fraction(8, 32);
4
        Fraction f2 = new Fraction(1, 4);
        System.out.println(f1==f2);
        System.out.println(f1.equals(f2));\\
   }
```

— Pourquoi les lignes 5 et 6 renvoient-elles des résultats différents?

>_ Solution public class Fraction { 2 private int numerateur; 3 private int denominateur; 4 5 $public\ Fraction(int\ numerateur, int\ denominateur) \{$ 6 this.numerateur = numerateur; 7 this.denominateur = denominateur; 8 simplify(); 9 10 11 public int gcd(int a, int b){ 12 $if (a==0){$ 13 return b: 14 } else if (b==0) { 15 return a; 16 } else{ 17 return gcd(b%a, a); 18 19 20 21 private void simplify(){ 22 int pgcd = gcd(this.numerateur, this.denominateur);

L'instruction $\mathbf{f1} == \mathbf{f2}$ vérifie que les deux objets sont identiques. Dans notre cas, $\mathbf{f1}$ et $\mathbf{f2}$ sont deux objets différents. On peut le vérifier en comparant leur représentation numérique. Cela peut être fait en utilisant la méthode .hashcode(). Vous pouvez essayer en exécutant la ligne suivante dans le main : System.out.println($\mathbf{f1}$.hashcode()).

return this.numerateur * ((Fraction) other).denominateur == ((Fraction) other).numerateur *

L'instruction f1.equals(f2) fait appel à la méthode equals() de notre classe Fraction et exécute les instructions que nous avons défini dans cette méthode.

2 Classes abstraites

23

24

25 26 27

28

29 30 31

32 33

34

35

36

37

} }

@Override

@Override

public String toString() {

Question 4: (15 minutes) Création d'une classe abstraite

this.numerateur = this.numerateur / pgcd;

return numerateur + "/" + denominateur;

if (other == null || this.getClass() != other.getClass()) return false;

public boolean equals(Object other) {

if (this == other) return true:

this.denominateur:

this.denominateur = this.denominateur / pgcd;

Une classe abstraite est une classe dont l'implémentation n'est pas complète et qui n'est pas instanciable. Elle est déclarée en utilisant le mot-clé abstract. Elle peut inclure des méthodes abstraites ou non. Bien que ne pouvant être instanciées, les classes abstraites servent de base à des sous-classes qui en sont dérivées. Lorsqu'une sous-classe est dérivée d'une classe abstraite, elle complète généralement l'implémentation de toutes les méthodes abstraites de la classe-mère. Si ce n'est pas le cas, la sous-classe doit également être déclarée comme abstraite.

```
1  // Exemple de classe abstraite
2  public abstract class Animal {
3  private int speed;
4  // Déclaration d une méthode abstraite
5  abstract void run();
6  }
```

```
8 public class Cat extends Animal {
9     // Implémentation d une méthode abstraite
10     void run() {
11     speed += 10;
12     }
```

- Créez une classe abstraite appelée Item. Elle doit avoir 4 attributs d'instance et un attribut de classe, qui sont les suivants :
- 1 private int id;
- 2 private static int count = 0; // Attribut de classe
- 3 private String name;
- 4 private double price;
- 5 private ArrayList < String > ingredients;

Conseil

Les attributs d'instance sont créés lors de l'instanciation d'un objet (à l'aide du mot-clé new) et détruits lors de la destruction de l'objet. Les attributs de classes (attributs statiques), quant à eux, sont créés lors de l'exécution du programme et détruits lors de l'arrêt du programme. En Java, les attributs de classes sont accessibles en utilisant le nom de la classe soit : ClassName.AttributeName.

 Créez un constructeur pour initialiser les attributs name, price, ingredients et id. L'attribut id incrémentera à chaque instanciation de la classe.

Conseil

Pensez à utilisez count pour initialiser la valeur d'id. Ainsi, dans le constructeur, id sera égal à ++count.

- Implémentez les getters des attributs id, name, price et ingredients.
- Implémentez les méthodes equals(Object o) et toString().

© Conseil

La méthode equals permet de comparer deux objets. Elle prend en entrée un objet de type Object et doit retourner True si l'objet instancié est égal à l'objet passé en paramètre.

```
>_ Solution
     import java.util.*;
 2
 3
 4
     public abstract class Item {
 5
 6
       private int id;
 7
       private static int count = 0;
 8
       private String name;
 9
       private double price;
10
       private ArrayList < String > ingredients;
11
12
       public Item (String name, double price, ArrayList<String> ingredients) {
13
         this.id = ++count;
14
          this.name = name;
15
         this.price = price;
16
         this.ingredients = ingredients;
17
18
19
       public int getID() {
20
         return this.id;
21
22
23
       public String getName() {
24
         return this.name;
25
26
27
       public\ double\ getPrice()\ \{
28
         return this.price;
29
30
       public ArrayList<String> getIngredients() {
31
32
         return this.ingredients;
33
34
35
       public boolean equals(Object o) {
36
          if (o instanceof Item) {
            Item i = (Item) o;
37
38
            return i.getID() == this.getID();
         }
39
40
         return false;
41
42
       public\ String\ toString()\ \{
43
44
          return "*-*-
               "\nID: " + this.getID() +
45
              "\nName: " + this.getName() +
46
47
              "\nPrice: " + this.getPrice() + " CHF" +
48
              "\nList of ingredients: " + this.getIngredients().toString() +
              "\n*-*-*-*-*;
49
50
       }
     }
51
```

Question 5: (**1** *15 minutes*) Classe abstraite et types d'attributs

- Implémentez une classe abstraite Figure contenant deux attributs protégés : largeur et longueur et deux méthodes abstraites : getAire()et getPerimetre().
- Créez deux classes Carre et Rectangle qui héritent de la classe Figure. À l'intérieur de ces classes, implémentez les méthodes getAire() et getPerimetre().

Conseil

Un attribut protégé est accessible aussi bien dans la classe-mère que dans la(les) classe(s)-fille(s). On utilise le mot-clé **protected** pour rendre des attributs protégés.

Pour rappel, pour créer une classe fille, on utilise le mot-clé extends. Par exemple : public class Carre extends Figure.

>_ Solution abstract class Figure { 2 protected float largeur; 4 protected float longueur; 5 6 public Figure(float largeur, float longueur){ 7 this.largeur = largeur; 8 this.longueur = longueur; 9 10 public abstract float getPerimetre(); 11 public abstract float getAire(); 12 13 14 15 class Carre extends Figure { 16 public Carre(float largeur) { 17 18 super(largeur, largeur); 19 20 21 @Override 22 public float getPerimetre() { 23 return this.largeur * 4; 24 } 25 26 @Override 27 public float getAire() { 28 return this.largeur * this.largeur; 29 30 31 } 32 33 class Rectangle extends Figure { 34 35 $public\ Rectangle\ (\textbf{float}\ largeur,\ \textbf{float}\ longueur) \{$ 36 super(largeur, longueur); 37 } 38 39 @Override 40 $public \ float \ getPerimetre() \{$ 41 return (this.largeur + this.longueur)*2; 42 43 44 @Override 45 $public \ float \ getAire() \{$ 46 return this.largeur * this.longueur; 47 48 } 49 50 public class Main $\{$ 51 public static void main(String[] args) { 52 Carre c = new Carre(5.0f);53 Rectangle r = new Rectangle(4.0f, 3.0f);54 System.out.println(c.getPerimetre());System.out.println(r.getAire()); 55 56 } 57

3 Interfaces

Question 6: (15 minutes) Interface et héritage (Liée à la question 4)

En Java, une interface se déclare comme suit :

```
public interface IMakeSound{
     final double MY_DECIBEL_VALUE = 75;
2
3
     void makeSound();
4
   }
   Les méthodes déclarées dans une interface doivent être implémentées dans des sous-classes :
   public class Cat extends Animal implements IMakeSound {
2
     void makeSound(){
       System.out.println("I meow at" + MY_DECIBEL_VALUE + "decibel.");
3
4
5
   }
```

- Créez une interface Edible contenant une méthode eatMe qui ne retourne aucune valeur.
- Créez une interface Drinkable contenant une méthode drinkMe qui ne retourne aucune valeur.
- Créez une classe Food qui hérite la classe Item (définie à la question 4) et qui implémente l'interface
 Edible. Ecrire le constructeur de Food et la méthode eatMe (dans la classe Food).

Conseil

Vous pouvez reprendre la classe Item de la question 4.

Dans la méthode eatMe(), vous pouvez simplement afficher un message en utilisant un println.

Certains aliments ne sont pas seulement Edible (mangeable) mais aussi Drinkable (buvable) comme les soupes par exemple.

4. Créez une classe Soup qui hérite de Food et implémente l'interface Drinkable. Ensuite, écrivez à la fois un constructeur pour Soup ainsi que la méthode drinkMe (dans la classe Soup).

Vous pouvez ensuite créer des instances de Soup et Food à l'aide des lignes suivantes pour tester les méthodes eatMe() et drinkMe().

```
1 Soup s1 = new Soup("Kizili soup", 7.7, new ArrayList<String>(Arrays.asList("bulgur", "meat", "tomato")));
```

Food f = new Food("Stuffed peppers", 12,new ArrayList<String>(Arrays.asList("rice", "tomato", "onion")));

>_ Solution

```
import java.util*;
  2
3
4
5
      public interface Edible{
        void eatMe();
  6
  7
      public interface Drinkable{
  8
        void drinkMe();
  9
 10
      public class Food extends Item implements Edible{
        public Food (String name, double price, ArrayList<String> ingredients){
 11
 12
           super(name, price, ingredients);
13
 14
        public void eatMe(){
15
           System.out.println("Eat me!" + toString());
16
      }
17
18
19
      public class Soup extends Food implements Drinkable{
20
        public Soup(String name, double price, ArrayList<String> ingredients){
21
           {\color{red} \textbf{super}} (\textbf{name}, \, \textbf{price}, \, \textbf{ingredients});
22
23
        public void drinkMe(){
24
25
26 }
           System.out.println("Drink the soup !" + toString());
```