

Algorithmes et Pensée Computationnelle

Abstract Classes and Interfaces in Java

1 Abstract Classes

Question 1: (🕒 10 minutes)

Une classe abstraite est une classe dont l'implémentation n'est pas complète. Elle est déclarée en utilisant le mot-clé `abstract`. Elle peut inclure des méthodes abstraites ou non. Les classes abstraites ne peuvent pas être instanciées, mais 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
2
3
4 // an abstract class declaration\\
5 public abstract class Animal {
6     private int speed;
7     // an abstract method declaration
8     abstract void run();
9 }
10
11 public class Cat extends Animal {
12     // implementation of abstract method
13     void run() {
14         speed += 10;
15     }
```

Implémentez une classe abstraite appelée **Item**.

1. Elle doit avoir 4 variables d'instance et une variable de classe, qui sont les suivantes :

```
private int id;
private static int count = 0;
private String name;
private double price;
private ArrayList<String> ingredients;
```

2. Elle doit avoir un constructeur prenant les variables `name`, `price` et `ingredients` comme paramètres. Pour définir `id`, utilisez la ligne suivante : `this.id = ++count`;

3. Implémentez des méthodes d'accesseurs pour les variables `id`, `name`, `price` et `ingredient`.

4. Implémentez les méthodes `equals(Object o)` et `toString()`.

💡 Conseil

Pour implémenter la méthode `equals` vous pouvez comparer les **ID** afin de voir si les items correspondent.

>_ Solution

```
1 package com.company;
2 import java.util.*;
3
4
5 public abstract class Item {
6
7     private int id;
8     private static int count = 0;
9     private String name;
10    private double price;
11    private ArrayList<String> ingredients;
12
13    public Item (String name, double price, ArrayList<String> ingredients) {
14        this.id = ++count;
15        this.name = name;
16        this.price = price;
17        this.ingredients = ingredients;
18    }
19
20    public int getID() {
21        return this.id;
22    }
23
24    public String getName() {
25        return this.name;
26    }
27
28    public double getPrice() {
29        return this.price;
30    }
31
32    public ArrayList<String> getIngredients() {
33        return this.ingredients;
34    }
35
36    public boolean equals(Object o) {
37        if (o instanceof Item) {
38            Item i = (Item) o;
39            return i.getID() == this.getID();
40        }
41        return false;
42    }
43
44    public String toString() {
45        return "*-*-*-*-*" +
46            "\nID: " + this.getID() +
47            "\nName: " + this.getName() +
48            "\nPrice: " + this.getPrice() + " CHF" +
49            "\nList of ingredients: " + this.getIngredients().toString() +
50            "\n*-*-*-*-*";
51    }
52 }
```

Question 2: (🕒 10 minutes)

1. Implémentez une classe abstraite **Figure** contenant deux méthodes abstraites : **getaire()** et **getperimetre()**
2. Etendez la classe **Figure** avec une classe **Carre**. Définir les classes **getaire()** et **getperimetre()** dans cette classe
3. Faire de même pour une classe **Rectangle**

💡 Conseil

Pour rendre une méthode abstraite utiliser "Abstract" comme pour les classes.
Pour étendre une classe utilisez : `public class Carre extends Figure.`

>_ Solution

```
1 package com.company;
2
3 public abstract class Figure {
4
5     public abstract float getperimetre();
6     public abstract float getaire();
7 }
8
9 public class Carre extends Figure {
10     private float cote;
11
12     public Carre(float c) {
13         cote = c;
14     }
15
16     public float getperimetre(float cote) {
17         return cote + cote;
18     }
19
20     public float getaire(float cote) {
21         return cote * cote;
22     }
23 }
24
25 public class Rectangle extends Figure {
26     private float largeur;
27     private float longueur;
28
29     public Rectangle (float lar, float lon){
30         largeur = lar;
31         longueur = lon;
32     }
33     public float getperimetre(float largeur, float longueur){
34         return largeur + longueur;
35     }
36 }
37     public float getaire(float largeur, float longueur){
38         return largeur * longueur;
39     }
40 }
```

2 Interface

Question 3: (🕒 10 minutes)

Une déclaration d'interface comprend les modificateurs (public, etc) et le mot clé interface :

```
1
2 public interface IMakeSound{
3     final double MY_DECIBEL_VALUE = 75;
4     void makeSound();
5 }
```

Les méthodes déclarées dans une interface doivent être implémentées dans des sous-classes :

```
1
2 public class Cat extends Animal implements IMakesound {
3
4     void makesound(){
5         System.out.println("I meow at" + MY_DECIBEL_VALUE + "decibel.");
6     }
7 }
```

1. Implémentez une interface **Edible** contenant une méthode **eatMe** qui ne retourne aucune valeur.
2. Implémentez une interface **Drinkable** contenant une méthode **drinkMe** qui ne retourne aucune valeur.
3. Implémentez une classe **Food** qui étend la classe **Item** et qui implémente **Edible**. Implémentez le constructeur de **Food** et la méthode **eatMe** (dans la classe **Food**).

Conseil

Vous pouvez reprendre la classe `Item` du premier exercice.
Pour implémenter la méthode `eatMe()` vous pouvez simplement mettre un `println`.

Certains aliments ne sont pas seulement `Edible` mais aussi `Drinkable` comme les soupes par exemple.

4. Implémentez une classe `Soup` qui étend `Food` et implémente `Drinkable`. Ensuite, implémentez à la fois un constructeur pour `Soup` ainsi que la méthode `drinkMe` (dans la classe `Soup`).

Vous pouvez ensuite créer des instances `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")));
2
3  Food f = new Food("Stuffed peppers", 12, new ArrayList<String>(Arrays.asList("rice", "tomato", "onion")));
```

>_ Solution

```
1  import java.util.*;
2
3
4  public interface Edible{
5      void eatMe();
6  }
7  public interface Drinkable{
8      void drinkMe();
9  }
10 public class Food extends Item implements Edible{
11     public Food (String name, double price, ArrayList<String> ingredients){
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         super(name, price, ingredients);
22     }
23     public void drinkMe(){
24         System.out.println("Drink the soup !" + toString());
25     }
26 }
```