Algorithmes et Pensée Computationnelle

Classes abstraites et interfaces en Java

Le but de cette séance est d'approfondir les notions de programmation orientée objet vues précédemment. Les exercices sont construits autour des concepts d'héritage, de classes abstraites et d'interface. Au terme de cette séance, vous devez être en mesure différencier une classe abstraite d'une interface, savoir à quel moment utiliser l'un ou l'autre, utiliser le concept d'héritage multiple, factoriser votre code afin de le rendre mieux structuré et plus lisible.
Cette série d'exercices est divisée en define section sections dont une section facultative (avec le label Optionnel). Le seul langage qui sera utilisé sera **Java**.
Le code présenté dans les énoncés se trouve sur Moodle, dans le dossier **Ressources**.

1 Abstract Classes

Question 1: (*10 minutes*)

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.

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

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

— 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;
```

> **Conseil**
>
> Les variables d'instance sont créées lors de l'instanciation d'un objet (à l'aide du mot clé **new**) et détruites lors de la destruction de l'objet. Les variables de classes (variables statiques), quant à elles, sont créées lors de l'exécution du programme et détruites lors de l'arrêt du programme. En Java, les variables de classes sont accessibles en utilisant le nom de la classe soit : **ClassName.VariableName**.

— Créer un constructeur pour initialiser les variables **name, price, ingredients et** id. La variable 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 variables 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

```
1  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
31    public ArrayList<String> getIngredients() {
32       return this.ingredients;
33    }
34
35    public boolean equals(Object o) {
36       if (o instanceof Item) {
37          Item i = (Item) o;
38          return i.getID() == this.getID();
39       }
40       return false;
41    }
42
43    public String toString() {
44       return "*-*-*-*-*-*-*" +
45             "\nID: " + this.getID() +
46             "\nName: " + this.getName() +
47             "\nPrice: " + this.getPrice() + " CHF" +
48             "\nList of ingredients: " + this.getIngredients().toString() +
49             "\n*-*-*-*-*-*-*";
50    }
51 }
```

Question 2: (*10 minutes*)

1. Implémentez une classe abstraite **Figure** contenant deux attributs protégés : **largeur** et **longueur** et deux méthodes abstraites : **getaire**()et **getperimetre**()

2. Etendez la classe Figure avec une classe **Carre**. Définir les classes getaire() et getpermietre() dans cette classe

3. Faire de même pour une classe **Rectangle**

Conseil

Pour rendre une méthode abstraite utiliser ”Abstract” comme pour les classes.
Un attribut protégé est accessible partout dans la classe mère et dans les classes étendant la classe mère. On utilise le mot clé **protected** pour rendre les attributs protégés.
Pour étendre une classe utilisez : public class Carre extends Figure.

Solution

```
1  package com.company;
2
3  public abstract class Figure {
4
5      protected float largeur;
6      protected float longueur;
7
8      public Figure(float largeur, float longueur){
9          this.largeur = largeur;
10         this.longueur = longueur;
11     }
12
13     public abstract float getperimetre();
14     public abstract float getaire();
15 }
16
17 class Carre extends Figure {
18
19     public Carre(float largeur) {
20         super(largeur, largeur);
21     }
22
23     public float getperimetre(float largeur) {
24         return largeur + largeur;
25     }
26
27     public float getaire(float largeur) {
28         return largeur * largeur;
29     }
30 }
31
32 class Rectangle extends Figure {
33
34     public Rectangle (float largeur, float longueur){
35         super(largeur, longueur);
36     }
37     public float getpertimetre(float largeur, float longueur){
38         return largeur + longueur;
39
40     }
41     public float getaire(float largeur, float longueur){
42         return largeur * longueur;
43     }
44 }
```

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;
```

```
    void makeSound();
}
```

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

```

public class Cat extends Animal implements IMakesound {

    void makesound(){
        System.out.println("I meow at" + MY_DECIBEL_VALUE + "decibel.");
    }
}
```

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 extends Food et implémente Drinkable. Ensuite, implémentez à la fois un constructeur pour Soup ainsi que la méthode drinkMe (dans la classe Soup).

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

```
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*;


public interface Edible{
    void eatMe();
}
public interface Drinkable{
    void drinkMe();
}
public class Food extends Item implements Edible{
    public Food (String name, double price, ArrayList<String> ingredients){
        super(name, price, ingredients);
    }
    public void eatMe(){
        System.out.println("Eat me!" + toString());
    }
}

public class Soup extends Food implements Drinkable{
    public Soup(String name, double price, ArayList<String> ingredients){
        super(name, price, ingredients);
    }
    public void drinkMe(){
        System.out.println("Drink the soup !" + toString());
    }
}
```

3 Exercises Complémentaires

Question 4: (*15 minutes*) **Poker Game**

Cet exercice vous demande de construire un jeu de poker en implémentant un programme Java. Celui-ci comprend trois parties importantes : un paquet de cartes (un « deck »), un joueur, et une carte, et elles devraient être représentées par trois classes respectives.

Suivez les instructions ci-dessous pour écrire le programme.

La class Carte

- Attributs : **valeur (int), couleur (int)**. Attention, la couleur d'une carte est représentée ici par une valeur de 0 à 3 au lieu d'une string.
- Implémentez un constructeur qui prend en argument une valeur et une couleur.
- Implémenter une méthode getter **getInfo()** qui affiche dans la console la valeur et la couleur de la carte, vous pouvez vous aider d'une fonction privée **getCouleur()** et **getValeur()** pour afficher les cas particuliers.

```
1  public class Carte {
2
3     private int valeur;
4     private int couleur;
5
6     public static final int HEART = 0;
7     public static final int DIAMOND = 1;
8     public static final int SPADE = 2;
9     public static final int CLUB = 3;
10
11
12    public Carte(int valeur, int couleur){
13       ...
14       ...
15    }
16
17    public void getInfo(){
18       ...
19    }
20
21    public String getValeur() {
22       ...
23    }
24
25    public String getCouleur() {
26       ...
27    }
28 }
```

La class Joueur

- Attributs : **mainDeCartes (Card[])** (un tableau de carte de taille maximal 2), **balance (int)** (la balance total du joureur), **mise (int)** (la mise du joueur), **monTour (boolean)** (valant true si c'est au tour du joueur).
- Implémentez le constructeur qui prend en argument **deux cartes** (puis les ajoute à sa main), une balance initiale, une mise initiale de 0. Vous pouvez initialiser **monTour** comme **false** au début.
- Implémentez une fonction **miser()** qui propose une mise sur la table mais ne retourne rien (utiliser le mot void). Définir la nouvelle mise du joueur.
- Implémentez la méthode **montrerMain()** qui montre les cartes sur la table.
- Implémentez une methode **gagner()** qui prend en argument la somme des gains sur la table que le joueur vient de gagner, ajouter là à sa balance.
- **Attention : miser()** et **gagner()** s'appliquent seulement si c'est au tour du joueur.

```
1  import java.util.ArrayList;
2
3  public class Joueur {
4
5     private ...
6     private ...
7     private ...
8     private ...
9
10
11    public Joueur(...){
12       ...
13    }
```

```
14
15
16      public void montrerMain(){
17          ...
18      }
19
20      public void miser(int valeur){
21          ...
22      }
23
24      public void gagner(int valeur){
25          ...
26      }
27
28  }
```

La classe Paquet :

- Attributs : **paquet (ArrayList<Carte>)** (un jeu de cartes complet), **NBR_CARTES (int)** (un attribut final et static (constante) égale à 52), **NBR_MELANGEs (int)** (une constant qui correspond au nombre de mélange effectué de valeur 100).
- Dans le constructeur ne prenant aucun argument, générer le deck en créant au fur et à mesure des cartes.
- Implémenter une fonction public **melanger()** qui mélange le jeu de carte et appeler là dans le constructeur après avoir construit le jeu de carte.
- Implémenter une fonction getter **getCarte()** qui retourn la carte en haut de la pil et la retire du jeu.

Conseil

Utilisez Random r = new Random() ; sa fonction **nextInt()** et utiliser des index et une variable **Carte** temporaire pour pouvoir échanger les cartes de position.

```
1   import java.util.ArrayList;
2   import java.util.Random;
3
4   public class Paquet {
5
6       private ...
7       ...
8       ...
9
10      public Paquet(){
11          ...
12      }
13
14
15      public void melanger(){
16          ...
17
18      }
19
20
21      public Carte getCarte(){
22          ...
23      }
24  }
```

Solution

```
1  public class Carte {
2
3     private int valeur;
4     private int couleur;
5
6     public static final int HEART = 0;
7     public static final int DIAMOND = 1;
8     public static final int SPADE = 2;
9     public static final int CLUB = 3;
10
11
12    public Carte(int valeur, int couleur){
13       this.valeur = valeur;
14       this.couleur = couleur;
15    }
16
17    public void getInfo(){
18       System.out.println("Carte: " + getValeur() + " de " + getCouleur());
19    }
20
21    public String getValeur() {
22       String res = "";
23       switch (valeur) {
24          case 11:
25             res = "Valet";
26             break;
27          case 12:
28             res = "Dame";
29             break;
30          case 13:
31             res = "King";
32             break;
33          default:
34             res = Integer.toString(valeur);
35             break;
36
37       }
38       return res;
39    }
40
41    public String getCouleur() {
42       String res = "";
43       switch (couleur) {
44          case 0:
45             res = "Coeur";
46             break;
47          case 1:
48             res = "Carreau";
49             break;
50          case 2:
51             res = "Pic";
52             break;
53          default:
54             res = "Trèfle";
55             break;
56
57       }
58       return res;
59    }
60 }
```

Solution

```
1  import java.util.ArrayList;
2
3  public class Joueur {
4
5     private Carte[] hand;
6     private int balance;
7     private boolean monTour;
8     private String nom;
9
10
11    public Joueur(Carte c1, Carte c2, int initialeBalance, String nom){
12       hand = new Carte[2];
13       hand[0] = c1;
14       hand[1] = c2;
15       monTour = false;
16       balance = initialeBalance;
17       this.nom = nom;
18    }
19
20
21    public void montrerMain(){
22       for (int i = 0; i < hand.length; i++){
23          hand[i].getInfo();
24       }
25    }
26
27    public void miser(int valeur ){
28       if (monTour && balance-valeur > 0 ){
29          balance -= valeur;
30          System.out.println("Le joueur " + nom + " vient de miser: " + Integer.toString(valeur));
31          System.out.println("La nouvelle Balance: "+  Integer.toString(balance));
32
33       } else {
34          System.out.println("Pas assez d'argent!");
35       }
36    }
37
38    public void gagner(int valeur){
39       if (monTour) {
40          this.balance += valeur;
41
42       }
43    }
44 }
```

Solution

```java
import java.util.ArrayList;
import java.util.Random;

public class Paquet {

   private ArrayList<Carte>  paquet;
   public static final int NBR_CARTES = 52;
   public static final int NBR_MELANGES = 100;

   public Paquet(){
      this.paquet = new ArrayList<>();

      for ( int couleur = Carte.HEART; couleur <= Carte.CLUB; couleur++ ) {
         for (int valeur = 1; valeur <= 13; valeur++) {
            paquet.add(new Carte(couleur, valeur));
         }
      }
      melanger();
   }


   public void melanger(){
      int index_1, index_2;

      Random generator = new Random();
      Carte temp;

      for (int i=0; i<NBR_MELANGES; i++) {
         index_1 = generator.nextInt( paquet.size() - 1 );
         index_2 = generator.nextInt( paquet.size() - 1 );
         temp = paquet.get( index_2 );
         paquet.set( index_2 , paquet.get( index_1 ) );
         paquet.set( index_1, temp );

      }

   }


   public Carte getCarte(){
      return paquet.remove(0);
   }


}
```

Question 5: (*15 minutes*) **Un jeu de rôle avec des personnages**

Vous allez implémenter un programme simple du jeu de rôle. Vous avez appris le concepte d'héritage, dans notre jeu de rôle celui-ci s'avère très utile car les différentes classes de personnages posssèdent certains attributs ou actions similaires. Il y a dans notre jeu le Guerrier, le Paladin, le Magicien et le Chasseur.

Ainsi il semble intéressant de construire une première classe **Personnage**. Un personnage est un objet qui possède plusieurs arguments :

- **nom (String)** : le nom du personnage
- **niveau (int)** : le niveau du personnage
- **pv (int)** : les points de vie du personnage
- **vitalite (int)** : la vitaité du personnage
- **force (int)** : la force du personnage
- **dexterite (int)** : la dextérité du personnage
- **endurance (int)** : l'endurance du personnage
- **intelligence (int)** : l'intelligence du personnage

Comme avant, suivez les intructions ci-dessous pour compléter le programme.

- Implémentez le constructeur de la classe **Personnage** qui prend tout ces attributs en argument.
- Il peut être intéressant d'afficher les caractéristiques de votre personnage. Implémentez une méthode

getInfo() dans la classe **Character** qui affiche dans la console, après avoir implémenté également les getters nécessaires.

— Chaque personnage dans ce jeu a un compteur pour leur vie restante. Celui-ci va être manipulé par une méthode setter. Ecrivez-la dans la classe **Personnage**.

— Maintenant implémentez les classes **Guerrier, Paladin, Magicien et Chasseur** qui héritent de **Personnage** en écrivant tout d'abord leur constructeur respectif.

```
1  import java.util.Random;
2
3  public abstract class Personnage {
4
5     private ...
6     ...
7
8     public Personnage(...){
9        this.nom = ...
10       ...
11
12    }
13
14    public void getInfo() {
15       ...
16    }
17    ...
18    ...
19 }
```

— Pour chacun des personnages, implémentez une méthode **attaqueBasique()** qui prend un autre personnage en argument et ne retourne rien. Celle-ci crée une attaque de votre choix en fonction des caractéristiques des personnages (ex : l'attaque du guerrier dépendra de sa force, l'attaque du chasseur de son endurance etc..), et détermine les points de vie restant en soustrayant la gravité de l'attque de la vitalité du personnage. Affichez le nom de celui que vous avez attaqué et ses points de vie restant.

— Cette method est commune à toutes les sous-classes, doit être déclarée abstraite dans la classe parente **Personnage**. Changer cette classe pour qu'elle soit maintenant abstraite avec une méthode abstraite **attaqueBasique()** :

— **Attention :** il faut utiliser le setteur pour réduire les **pv** de l'autre personnage.

```
1
2  public class Guerrier extends Personnage {
3
4
5  }
6
7
8  public class Magicien extends Personnage {
9
10
11
12 }
13
14 public class Paladin extends Personnage {
15
16
17 }
18
19 public class Chasseur extends Personnage {
20
21
22 }
```

Solution

```java
import java.util.Random;

public abstract class Personnage {

    private String nom;
    private int niveau;
    private int pv;

    private int vitalite;
    private int force;
    private int dexterite;
    private int endurance;
    private int intelligence;


    public Personnage(String nom, int niveau, int pv, int vitalite, int force, int dexterite, int endurance, int
        intelligence){
        this.nom = nom;
        this.niveau = niveau;
        this.pv = pv;

        this.vitalite = vitalite;
        this.force = force;
        this.dexterite = dexterite;
        this.endurance = endurance;
        this.intelligence = intelligence;

    }

    public void getInfo() {
        System.out.println("Nom" + getNom());
        System.out.println("Niveau" + getNiveau());
        System.out.println("Vitalité" + getVitalite());
        System.out.println("Force" + getForce());
        System.out.println("Dexterité" + getDexterite());
        System.out.println("Endurance" + getEndurance());
        System.out.println("Intelligence" + getIntelligence());
    }


    public abstract void attaqueBasique(Personnage other);

    public String getNom() {
        return nom;
    }

    public int getIntelligence() {
        return intelligence;
    }

    public int getNiveau() {
        return niveau;
    }

    public int getPv() {
        return pv;
    }

    public int getVitalite() {
        return vitalite;
    }

    public int getForce() {
        return force;
    }

    public int getDexterite() {
        return dexterite;
    }

    public int getEndurance() {
```

Solution

```
71          return endurance;
72      }
73
74      public void setPv(int pv) {
75          this.pv = pv > 0 ? pv : 0 ;
76      }
77  }
```

Solution

```
1  public class Guerrier extends Personnage {
2
3    public Guerrier(String nom, int niveau, int pv,  int vitalite, int force, int dexterite, int endurance, int
         intelligence) {
4      super(nom,  niveau, pv, vitalite,  force,  dexterite,  endurance,  intelligence);
5
6    }
7    public void attaqueBasique(Personnage other){
8      int attaque = (int) (getForce()*0.5);
9
10     other.setPv(other.getVitalite()-attaque);
11
12     System.out.println(this.getNom() + " attaque " + other.getNom());
13     System.out.println("Basic attaque og : " + attaque);
14     System.out.println(this.getNom() + " PV:" + this.getPv());
15     System.out.println(other.getNom() + " PV:" + this.getPv());
16
17   }
18 }
19
20 public class Magicien extends Personnage {
21
22
23   public Magicien(String nom, int niveau, int pv, int vitalite, int force, int dexterite, int endurance, int
         intelligence) {
24     super(nom,  niveau, pv, vitalite,  force,  dexterite,  endurance,  intelligence);
25   }
26   @Override
27   public void attaqueBasique(Personnage other){
28     int attaque = (int) (getIntelligence()*0.5);
29
30     other.setPv(other.getVitalite()-attaque);
31
32     System.out.println(this.getNom() + " attaque " + other.getNom());
33     System.out.println("Basic attaque og : " + attaque);
34     System.out.println(this.getNom() + " PV:" + this.getPv());
35     System.out.println(other.getNom() + " PV:" + this.getPv());
36
37   }
38
39 }
40
41 public class Paladin extends Personnage {
42
43   public Paladin(String nom, int niveau, int pv, int vitalite, int force, int dexterite, int endurance, int
         intelligence) {
44     super(nom,  niveau, pv,  vitalite,  force,  dexterite,  endurance,  intelligence);
45   }
46
47
48   @Override
49   public void attaqueBasique(Personnage other){
50     int attaque = (int) (getEndurance()*0.7);
51
52     other.setPv(other.getVitalite()-attaque);
53
54     System.out.println(this.getNom() + " attaque " + other.getNom());
55     System.out.println("Basic attaque og : " + attaque);
56     System.out.println(this.getNom() + " PV:" + this.getPv());
57     System.out.println(other.getNom() + " PV:" + this.getPv());
58
59   }
60 }
61
62
63 public class Chasseur extends Personnage {
64
65   public Chasseur(String nom, int niveau, int pv, int vitalite, int force, int dexterite, int endurance, int
         intelligence) {
66     super(nom,  niveau, pv,  vitalite,  force,  dexterite,  endurance,  intelligence);
67   }
```

Solution

```
68
69      @Override
70      public void attaqueBasique(Personnage other){
71          int attaque = (int) (getDexterite()*0.5);
72
73          other.setPv(other.getVitalite()-attaque);
74
75          System.out.println(this.getNom() + " attaque " + other.getNom());
76          System.out.println("Basic attaque og : " + attaque);
77          System.out.println(this.getNom() + " PV:" + this.getPv());
78          System.out.println(other.getNom() + " PV:" + this.getPv());
79
80      }
81  }
```