# Algorithmes et Pensée Computationnelle

*Programmation orientée objet : Héritage et Polymorphisme*

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

## 1 Rappel : Surcharge des opérateurs - Python

Dans cette section, vous manipulerez des fractions sous forme d'objets. Vous ferez des opérations de base sur ce nouveau type d'objets.
**Question 1:** (🕒 *5 minutes*) Dans un projet que vous aurez au préalable préparé, créez un fichier appelé **surcharge.py**.À l'intérieur de ce fichier, créer une classe **Fraction** qui aura comme attributs un numérateur et un dénominateur.

**Question 2:** (🕒 *5 minutes*) Définir un constructeur à votre classe. Assignez des valeurs par défaut à vos attributs.

> **Conseil**
>
> Les valeurs par défaut seront assignées à votre objet au cas où il est instancié sans valeurs. Ainsi en faisant **f = Fraction()**, on obtiendra un objet **Fraction** ayant pour valeurs un numérateur et un dénominateur à 1 soit $\frac{1}{1}$.

**Question 3:** (🕒 *5 minutes*)

## 2 Notions d'héritage - Java

Le but de cette partie est de pratiquer et d'assimiler les notions liées à l'héritage. Pour cela. nous allons nous inspirer de l'exemple présenté dans le cours.
Nous allons créer une classe **Livre()** qui contiendra deux sous-classes, **Livre_Audio()** et **Livre_Illustre()**. Les sous-classes hériteront des attributs et méthodes de la classe mère.
**Question 4:** (🕒 *10 minutes*) Création de classe et sous-classes

Créez la classe mère **Livre()** avec les caractéristiques suivantes :
- une variable privée **titre**
- une variable privée **auteur**
- une variable privée **annee**
- une variable privée **note** (initialisée à **–1**)
- le constructeur public prenant en argument les trois premières variables ci-dessus
- une méthode **printInfo()** qui affiche le titre, l'auteur, l'année et la note d'un ouvrage
- une méthode **setNote()** qui permet de définir la variable **note**

Créez les classes filles avec les caractéristiques suivantes :

**class Livre_Audio extends Livre**
- un attribut **narrateur**

**class Livre_Illustre extends Livre**
- un attribut **dessinateur**

```
1  public class Livre {
2
3  }
4
5  public class Livre_Audio extends Livre {
6
7  }
8
9  public class Livre_Illustre extends Livre {
10
11 }
```

**Conseil**

En java, lors de la déclaration d'une classe, le mot clef **extends** permet d'indiquer qu'il s'agit d'une sous-classe de la classe indiquée.
Le mot clef **super** permet à la sous classe d'hérité d'éléments de la classe mère. **super** peut être utilisé dans le constructeur de la sous-classe selon l'example suivant : **super(variable_mère_1, variable_mère_2, variable_mère_3, etc.);**. Ainsi, il n'est pas nécessaire de redefinir toutes les variables d'une sous-classe !
L'instruction **super** doit toujours être la première instruction dans le constructeur d'une sous-classe.

**Solution**

```
1   public class Livre {
2
3     private String titre;
4     private String auteur;
5     private int annee;
6     private int note = -1;
7
8     public Livre(String titre, String auteur, int annee){
9       System.out.println("Création d'un livre");
10      this.titre = titre;
11      this.auteur = auteur;
12      this.annee = annee;
13    }
14
15    public void printInfo() {
16      System.out.println("A propos du livre");
17      System.out.println("--------------------");
18      System.out.println("Titre : \""+titre+"\"");
19      System.out.println("Auteur : "+auteur);
20      System.out.println("Année : "+annee);
21      System.out.println("Note : "+note);
22    }
23
24    public void setNote(int note) {
25      this.note = note;
26    }
27  }
28
29  public class Livre_Audio extends Livre {
30    private String narrateur;
31
32    public Livre_Audio(String titre, String auteur, int annee, String narrateur){
33      super(titre, auteur, annee);
34      System.out.println("Création d'un livre audio");
35      this.narrateur = narrateur;
36    }
37  }
38
39  public class Livre_Illustre extends Livre {
40
41    private String dessinateur;
42
43    public Livre_Illustre(String titre, String auteur, int annee, String dessinateur) {
44      super(titre, auteur, annee);
45      System.out.println("Création d'un livre illustré");
46      this.dessinateur = dessinateur;
47    }
48  }
```

**Question 5:** (*5 minutes*) Méthode et héritage

Maintenant que vous avez créé la classe et les sous classes correspondantes, vous pouvez créer un objet **Livre** à l'aide du constructeur de la sous-classe **Livre_Audio**. Si vous manquez d'inspiration vous pouvez indiquer

les valeurs suivantes : titre : ”Hamlet”, auteur : ”Shakespeare”, année : ”1609” et le narrateur ”William.

Une fois l'objet créé, attribuez lui une note à l'aide de la méthode définie précédemment.

Finalement, utilisez la méthode **printInfo()** pour afficher les informations du livre.

La méthode étant définie dans la classe mère, elle n'a pas connaissance de la variable **narrateur** définie dans la sous-classe. Redéfinissez la méthode dans la sous-classe pour y inclure l'information sur le narrateur.

**Conseil**

**Attention**, on vous demande de créer un objet **Livre** et non pas **Livre_Audio**.
Le mot clef **super** peut être utilisé dans la redéfinition d'une méthode selon l'example suivant : **super.nom_de_la_methode();**. Cette instruction permet d'inclure tout ce qui est défini dans la “méthode mère” et vous pouvez la complétez selon les caractéristiques de votre sous-classe.
L'instruction **super** doit toujours être la première instruction dans le redéfinition d'une méthode dans une sous-classe.

**Solution**

```

public class Livre_Audio extends Livre {
    private String narrateur;

    public Livre_Audio(String titre, String auteur, int annee, String narrateur){
        super(titre, auteur, annee);
        System.out.println("Création d'un livre audio");
        this.narrateur = narrateur;
    }

    // redéfinition de la fonction printInfo() dans la sous-classe Book
    public void printInfo() {
        super.printInfo(); //permet de reprendre les éléments de la fonction mère
        System.out.println("Narrateur: "+ narrateur); //On ajoute l attribut supplémentaire propre à la sous-classe
    }
}
```

```

class Main {
    public static void main(String[] args) {

        Livre monLivre = new Livre_Audio("Hamlet", "Shakespeare", 1609, "William");
        monLivre.setNote(5);
        monLivre.printInfo();

    }
}
```

# 3 Polymorphisme - Java

Dans cette partie de cette session d'exercice, vous serez amenés à créer 2 nouvelles sous-classes de la classe mère combattant. La première classe représentera un soigneur, qui, lorsqu'il attaquera quelqu'un, le soignera au lieu de le blesser. La deuxième classe représentera un combattant spécialisé dans l'attaque, qui aura la capacité d'attaquer un certain nombre de fois (ce nombre sera défini au moment ou vous l'instancierez). Pensez à télécharger la dernière version de la classe Combattant dans le dossier ressources.

Voici le squelette du code que vous trouverez également dans le dossier ressources du moodle :

```java
import java.util.HashMap;
import java.util.List;
import java.util.ArrayList;
import java.util.Map;

public class Combattant {
    private String name;
    private int health;
    private int attack;
    private int defense;
    private static List<Combattant> instances = new ArrayList<Combattant>();
    private static HashMap<String, Integer> attack_modifier = new HashMap(Map.of("poing", 2, "pied", 2, "tete", 3));

    public Combattant(String name, int health, int attack, int defense) {
        this.name = name;
        this.health = health;
        this.attack = attack;
        this.defense = defense;
        instances.add(this);
    }

    public static void addInstances(Combattant other){
        instances.add(other);
    }

    public int getAttack() {
        return attack;
    }

    public int getHealth() {
        return health;
    }

    public int getDefense() {
        return defense;
    }

    public String getName() {
        return name;
    }

    public void setAttack(int attack) {
        this.attack = attack;
    }

    public void setDefense(int defense) {
        this.defense = defense;
    }

    public void setHealth(int health) {
        this.health = health;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Boolean isAlive() {
        if (this.health > 0) {
            return true;
        } else {
            return false;
        }
    }

    public static void checkDead() {
        // Initialisation de la liste de Combattants en vie
        List<Combattant> temp = new ArrayList<Combattant>();
        //Ici, on parcourt les instances de Combattant
        for (Combattant f : Combattant.instances) {
            // Et on fait appel à la méthode isAlive() pour vérifier que le Combattant est en vie
            if (f.isAlive()) {
                temp.add(f);
```

```
74            } else {
75                System.out.println(f.getName() + " est mort");
76            }
77        }
78        Combattant.instances = temp;
79    }
80
81
82    public static void checkHealth() {
83        for (Combattant f : Combattant.instances) {
84            System.out.println(f.getName() + " a encore " + f.getHealth() + " points de vie");
85        }
86        System.out.println("--------------------------------");
87    }
88
89
90    public void attack(String type, Combattant other) {
91        if (!this.isAlive()) {
92            System.out.println(this.getName() + " est mort et ne peut plus rien faire");
93        }
94        else{
95            if (!other.isAlive()) {
96                System.out.println(other.getName() + " est déjà mort");
97            }
98            else{
99                int damage = (int) Combattant.attack_modifier.get(type) * this.attack - other.getDefense();
100               other.setHealth(other.getHealth() - damage);
101               Combattant.checkDead();
102               Combattant.checkHealth();
103           }
104       }
105
106   }
107 }
108
109 class Soigneur extends Combattant {
110
111   //TODO
112
113   public Soigneur(String name, int health, int attack, int defense, int soin)
114   {
115       //TODO
116   }
117
118   //TODO
119
120
121   public void résurrection(Combattant other){
122       //TODO
123   }
124
125   public void attack(Combattant other) {
126       //TODO
127   }
128 }
129
130 class Attaquant extends Combattant{ // a la capacité d attaquer deux fois
131
132   //TODO
133
134   public Attaquant(String name, int health, int attack, int defense, int multiplicateur){
135       //TODO
136   }
137
138   //TODO
139
140   public void attack(String type, Combattant other) {
141       //TODO
142   }
143 }
```

**Question 6:** (🕓 *5 minutes*) Sous-classe Soigneur

Commencez par déclarer une nouvelle sous-classe soigneur. Cette sous-classe prendra un nouvel attribut privé, entier, nommé résurrection, qui vaudra 1 lors de l'instanciation.

Déclarez le constructeur de cette classe ainsi que les getter et setter permettant d'interagir avec ce nouvel attribut.

**Conseil**

Pensez à utiliser le constructeur de votre classe mère Combattant

**Solution**

```
1   class Soigneur extends Combattant { // a la capacité de soigner et ressusciter un Combattant
2
3       private int résurrection;
4
5       public Soigneur(String name, int health, int attack, int defense, int soin)
6       {
7           super(name,health,attack,defense);
8           résurrection = 1;
9       }
10
11      public int getRésurrection(){
12          return this.résurrection;
13      }
14
15      public void setRésurrection(int etat){
16          this.résurrection = etat;
17      }
```

**Question 7:** (*10 minutes*) Méthode résurection de la sous-classe Soigneur

Commencez par déclarer une nouvelle méthode nommée résurrection(Fighter other).

Cette méthode permettra de faire revenir un Combattant à la vie, mais le Soigneur ne pourra le faire qu'une seule fois.

Commencez par controler que l'instance depuis laquelle la méthode est appelée soit toujours en vie. Si ce n'est pas le cas, indiquez le en indiquant : nom_instance est mort et ne peut plus rien faire.

Contrôlez ensuite que l'instance other soit vraiment morte. Si ce n'est pas le cas, indiquez le via : nom_other est toujours en vie.

Pour finir, contrôlez que l'attribut résurrection de l'instance depuis laquelle la méthode est appelée soit égale à 1. Si ce n'est pas le cas, indiquez : nom_instance ne peut plus ressuciter personne.

Si tous ces éléments sont réunis, faites revenir le Combattant other à la vie en lui remettant 10 points de vie et en l'ajoutant à la liste des instances de la classe Combattant. Pensez également à mettre l'attribut résurrection de l'instance appelée à 0 afin de l'empêcher de réutiliser ce pouvoir, à appeler la méthode checkHealth(), et à indiquer : nom_other est revenu à la vie !

**Conseil**

Utilisez un branchement conditionnel pour les contrôles.

Une nouvelle méthode nommée addInstances(Combattant other) a été créée dans la classe Combattant. Regardez à quoi elle sert et utilisez la.

**Solution**

```
1   public void résurrection(Combattant other){
2       if(!this.isAlive()) {
3           System.out.println(this.getName() + " est mort et ne peut plus rien faire");
4       }
5       else{
6           if (other.isAlive()) {
7               System.out.println(other.getName() + " est toujours en vie !");
8           } else {
9               if (this.getRésurrection() == 0) {
10                  System.out.println(this.getName() + " ne peut plus ressusciter personne");
11              } else {
12                  other.setHealth(10);
13                  Combattant.addInstances(other);
14                  this.setRésurrection(0);
15                  System.out.println(other.getName() + " vient de revenir à la vie");
16                  Combattant.checkHealth();
17              }
18          }
19      }
20  }
```

**Question 8:** (*10 minutes*) Méthode attack de la sous-classe soigneur

Réécrivez la méthode attack de la sous-classe soigneur afin d'ajouter des points de vie à other au lieu de lui en retirer.

Le seul argument nécessaire pour cette méthode sera le Combattant other.

Commencez par contrôler que le Soigneur depuis lequel la méthode est appelée est encore en vie. Si ce n'est pas le cas, indiquez : nom_instance est mort et ne peut plus rien faire.
Contrôlez ensuite si other est toujours en vie. Si ce n'est pas le cas indiquez : nom_other est déjà mort, ressuscitez le afin de pouvoir le soigner. Contrôlez également qu'il ait moins de 10 points de vie. Si ce n'est pas le cas, indiquez le via : nom_other a déjà le maximum de points de vie.

Si toutes ces conditions sont réunies, ajoutez la valeur de l'attaque de l'instance qui appelle la méthode aux points de vie de other. Pour terminer, appelez la méthode de classe checkHealth().

**Conseil**

Pensez à utiliser du branchement conditionnel pour les contrôles.

**Solution**

```
1  public void attack(Combattant other) {
2      if(!this.isAlive()) {
3          System.out.println(this.getName() + " est mort et ne peut plus rien faire");
4      }
5      else{
6          if (other.getHealth() >= 10) {
7              System.out.println(other.getName() + " a déjà le maximum de points de vie");
8          }
9          if (!other.isAlive()) {
10             System.out.println(other.getName() + " est déjà mort, ressucitez le pour pouvoir le soigner");
11         } else {
12             other.setHealth(other.getHealth() + this.getAttack());
13             Combattant.checkHealth();
14         }
15     }
16 }
```

**Question 9:** (*5 minutes*) Sous-classe Attaquant

Commencez par déclarer une nouvelle sous-classe Attaquant. Cette sous-classe prendra un nouvel attribut privé, entier, nommé multiplicateur, qui sera passé en argument du constructeur de la sous-classe.

Déclarez le constructeur de cette classe ainsi que les getter et setter permettant d'interagir avec ce nouvel attribut.

**Conseil**

Pensez à utiliser le constructeur de votre classe mère Combattant

**Solution**

```
1  class Attaquant extends Combattant{ // a la capacité d attaquer deux fois
2
3      private int multiplicateur;
4
5      public Attaquant(String name, int health, int attack, int defense, int multiplicateur){
6          super(name,health,attack,defense);
7          this.multiplicateur = multiplicateur;
8      }
9
10     public int getMultiplicateur() {
11         return multiplicateur;
12     }
13
14     public void setMultiplicateur(int multiplicateur){
15         this.multiplicateur = multiplicateur;
16     }
```

**Question 10:** (*10 minutes*) Méthode attack de la sous-classe Attaquant

Réécrivez la méthode attack de la sous-classe Attaquant afin d'effectuer plusieurs attaques sur other en fonction de son attribut multiplicateur.

Y'a t-il besoin de contrôler si l'instance depuis laquelle la méthode est appelée est encore en vie ?

Indiquez systématiquement le numéro de l'attaque, puis effectuez l'attaque. Répétez le procédé jusqu'à ce que le numéro de l'attaque soit égal à celui de multiplicateur_instance.

**Conseil**

Aidez vous de la méthode attack de la classe mère Combattant.

**Solution**

```
public void attack(String type, Combattant other) {
    for (int i = 0; i < this.getMultiplicateur(); i++) {
        System.out.println("Attaque n " + (i+1));
        super.attack(type, other);
    }
}
```

Si tout est correct, en utilisant ce main :

```
public class Main {
    public static void main(String[] args) {
        Combattant P1 = new Combattant("P1", 10, 2, 2);
        Attaquant P2 = new Attaquant("P2", 10, 2, 2,2);
        Soigneur P3 = new Soigneur("P3",10,4,2,4);
        P1.attack("pied",P2);
        P1.attack("poing",P2);
        P1.attack("tete",P2);
        P1.attack("tete",P2);
        P3.résurrection(P2);
        P1.attack("pied",P2);
        P1.attack("poing",P2);
        P1.attack("tete",P2);
        P3.attack(P2);
        P2.attack("tete",P1);
    }
}
```

Vous devriez obtenir :

```
P1 a encore 10 points de vie
P2 a encore 8 points de vie
P3 a encore 10 points de vie
--------------------------------
P1 a encore 10 points de vie
P2 a encore 6 points de vie
P3 a encore 10 points de vie
--------------------------------
P1 a encore 10 points de vie
P2 a encore 2 points de vie
P3 a encore 10 points de vie
--------------------------------
P2 est mort
P1 a encore 10 points de vie
P3 a encore 10 points de vie
--------------------------------
P2 vient de revenir à la vie
P1 a encore 10 points de vie
P3 a encore 10 points de vie
P2 a encore 10 points de vie
--------------------------------
P1 a encore 10 points de vie
P3 a encore 10 points de vie
P2 a encore 8 points de vie
--------------------------------
P1 a encore 10 points de vie
P3 a encore 10 points de vie
P2 a encore 6 points de vie
--------------------------------
P1 a encore 10 points de vie
P3 a encore 10 points de vie
```

```
P2 a encore 2 points de vie
----------------------------------
P1 a encore 10 points de vie
P3 a encore 10 points de vie
P2 a encore 6 points de vie
----------------------------------
Attaque n 1
P1 a encore 6 points de vie
P3 a encore 10 points de vie
P2 a encore 6 points de vie
----------------------------------
Attaque n 2
P1 a encore 2 points de vie
P3 a encore 10 points de vie
P2 a encore 6 points de vie
----------------------------------

Process finished with exit code 0
```

# 4 Héritage en Python