Algorithmes et Pensée Computationnelle

Programmation orientée objet : Héritage et Polymorphisme - exercices basiques

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 surcharge d'opérateurs/méthodes et de polymorphisme. Au terme de cette séance, vous devez être en mesure de factoriser votre code afin de le rendre mieux structuré et plus lisible. À chaque exercice, le langage de programmation à utiliser sera spécifié.
Le code présenté dans les énoncés se trouve sur Moodle, dans le dossier **Ressources**.

1 Notions d'héritage - Java

Le but de cette partie est de mettre en pratique les notions liées à l'héritage. Nous allons créer une classe **Livre()** qui représentera notre classe mère. Nous allons également créer deux classes filles, **Livre_Audio()** et **Livre_Illustre()**. Les classes filles hériteront des attributs et méthodes de la classe mère.

Question 1: (*20 minutes*) **Création des différentes classes**

Créez la classe mère **Livre** avec les caractéristiques suivantes :

- un attribut **privé String** nommé **titre**,
- un attribut **privé String** nommé **auteur**,
- un attribut **privé** int nommé **annee**,
- un attribut **privé** int nommé **note** (initialisé à **–1**),
- le **constructeur** de la classe qui prendra les trois premiers arguments cités ci-dessus,
- une méthode **setNote()** qui permet de définir l'attribut **note**,
- une méthode **getNote()** qui permet de retourner l'attribut **note**,
- une méthode **toString()** qui retournera le titre, l'auteur, l'année et la note d'un ouvrage **note** (réécrire cette méthode permettra d'afficher un objet **Livre** en utilisant **System.out.println()**

Attention, si la **note** n'a pas été modifiée et qu'elle vaut toujours **–1**, affichez "Note : pas encore attribuée" au lieu de "Note : **note**" via la méthode **toString()**.

Créez les classes filles avec les caractéristiques suivantes :

class Livre_Audio extends Livre

- un attribut **privé String** nommé **narrateur**

class Livre_Illustre extends Livre

- un attribut **privé String** nommé **illustrateur**

Conseil

En Java, lors de la déclaration d'une classe, le mot clé **extends** permet d'indiquer qu'il s'agit d'une classe fille de la classe indiquée.
Le mot clé **super** permet à la sous classe d'hériter d'éléments de la classe mère. **super** peut être utilisé dans le constructeur de la sous-classe selon l'example suivant : **super(attribut_mère_1, attribut_mère_2, attribut_mère_3, etc.);**. Ainsi, il n'est pas nécessaire de redéfinir tous les attributs d'une classe fille !
L'instruction **super** doit toujours être la première instruction dans le constructeur d'une sous-classe.
Vous pouvez vous servir de **n** dans une chaîne de caractères pour effectuer un retour à la ligne lors de l'affichage.

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 int getNote(){
16    return this.note;
17 }
18
19 public void setNote(int note) {
20    this.note = note;
21 }
22
23 public String toString() {
24    if (note == -1){
25       return "A propos du livre \n---------- \nTitre : " +titre+ "\nAuteur : "+auteur+ "\nAnnée
          : "+annee+ "\nNote : non attribuée";
26    }
27    else{
28       return "A propos du livre \n---------- \nTitre : " +titre+ "\nAuteur : "+auteur+ "\nAnnée
          : "+annee+ "\nNote : "+note;
29    }
30 }
31 }
32
33 class Livre_Audio extends Livre {
34    private String narrateur;
35
36    public Livre_Audio(String titre, String auteur, int annee, String narrateur){
37    super(titre, auteur, annee);
38    System.out.println("Création d'un livre audio");
39    this.narrateur = narrateur;
40 }
41
42 }
43
44 class Livre_Illustre extends Livre {
45
46    private String illustrateur;
47
48    public Livre_Illustre(String titre, String auteur, int annee, String illustrateur) {
49    super(titre, auteur, annee);
50    System.out.println("Création d'un livre illustré");
51    this.illustrateur = illustrateur;
52 }
53
54 }
```

Question 2: (*5 minutes*) **Méthode et héritage**

Maintenant que vous avez créé la classe mère et les classes filles correspondantes, vous pouvez créer un objet **Livre** à l'aide du constructeur de la classe **Livre_Audio** (et des arguments donnés lors de la création de l'objet).
En instanciant l'objet, vous pourriez utiliser 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 **setNote()** définie précédemment.

Finalement, utilisez la méthode **System.out.println()** 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 classe fille pour y inclure l'information sur le narrateur.

Faites pareil avec la classe **Livre_Illustre** et son attribut **Illustrateur**

Conseil

Attention, on vous demande de créer un objet **Livre** et non pas **Livre_Audio**.
Le mot-clé **super** peut être utilisé dans la redéfinition d'une méthode selon l'exemple suivant : **super.nom_de_la_methode();**. Le mot clé**super** représente la classe parent, tout comme le mot clé **this** représentait l'instance avec laquelle la méthode était appelée.
L'instruction **super** doit toujours être la première instruction dans le redéfinition d'une méthode dans une classe fille.

Solution

```
1  class Livre_Audio extends Livre {
2     private String narrateur;
3
4     public Livre_Audio(String titre, String auteur, int annee, String narrateur){
5     super(titre, auteur, annee);
6     System.out.println("Création d'un livre audio");
7     this.narrateur = narrateur;
8  }
9
10 // redéfinition de la fonction toString dans la classe fille Livre_Audio
11 public String toString() {
12    return super.toString() + "\nNarrateur: "+ narrateur+"\n"; //Ajoute narrateur à la chaine de caractère crée
          par la classe mère (super)
13 }
14 }
15
16 class Livre_Illustre extends Livre {
17
18    private String illustrateur;
19
20    public Livre_Illustre(String titre, String auteur, int annee, String illustrateur) {
21    super(titre, auteur, annee);
22    System.out.println("Création d'un livre illustré");
23    this.illustrateur = illustrateur;
24 }
25 public String toString() {
26    return super.toString() + "\nIllustrateur: "+ illustrateur + "\n"; //Ajoute illustrateur à la chaine de caractère
          crée par la classe mère (super)
27 }
28 }
```

```
1  public class Main {
2
3     public static void main(String[] args) {
4     Livre Livre1 = new Livre_Audio("Hamlet", "Shakespeare", 1609,"William");
5     Livre1.setNote(5);
6     System.out.println(Livre1);
7  }
8
9  }
```

Lorsque toutes les étapes auront été effectuées, effectuez ce main :

```
public class Main {

    public static void main(String[] args) {
    Livre Livre1 = new Livre_Audio("Hamlet", "Shakespeare", 1609,"William");
    Livre1.setNote(5);
    System.out.println(Livre1);
    Livre Livre2 = new Livre("Les Misérables","Hugo",1862);
    System.out.println(Livre2);

}

}
```

Vous devriez obtenir :

```
Création d'un livre
Création d'un livre audio
Création d'un livre
A propos du livre
--------------------
   Titre : Hamlet
Auteur : Shakespeare
Année : 1609
Note : 5
Narrateur: William
A propos du livre
--------------------
   Titre : Les Misérables
Auteur : Hugo
Année : 1862
Note : non attribuée
Process finished with exit code 0
```

2 Polymorphisme - Java

Les exercices de cette section sont une suite des exercices de la section 2 de la semaine passée. Dans cette partie, vous serez amenés à créer 2 nouvelles sous-classes de la classe mère **Combattant**. La première classe représentera un **Soigneur**, qui, lorsqu'il "attaquera" un **Combattant**, le soignera au lieu de le blesser. La deuxième classe représentera un combattant spécialisé dans l'attaque **Attaquant**, qui aura la capacité d'attaquer un certain nombre de fois (ce nombre sera défini au moment où 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 :

```
1  import java.util.HashMap;
2  import java.util.List;
3  import java.util.ArrayList;
4  import java.util.Map;
5
6  public class Combattant {
7      private String name;
8      private int health;
9      private int attack;
10     private int defense;
11     private static List<Combattant> instances = new ArrayList<Combattant>();
12     private static HashMap<String, Integer> attack_modifier = new HashMap(Map.of("poing", 2, "pied", 2, "tete", 3));
13
14     public Combattant(String name, int health, int attack, int defense) {
15     this.name = name;
16     this.health = health;
17     this.attack = attack;
18     this.defense = defense;
19     instances.add(this);
20
21 public static void addInstances(Combattant other){
22     instances.add(other);
23 }
24
25 public int getAttack() {
26     return attack;
27 }
28
29 public int getHealth() {
30     return health;
31 }
32
33 public int getDefense() {
34     return defense;
35 }
36
37 public String getName() {
38     return name;
39 }
40
41 public void setAttack(int attack) {
42     this.attack = attack;
43 }
44
45 public void setDefense(int defense) {
46     this.defense = defense;
47 }
48
49 public void setHealth(int health) {
50     this.health = health;
51 }
52
53 public void setName(String name) {
54     this.name = name;
55 }
56
57 public Boolean isAlive() {
58     if (this.health > 0) {
59         return true;
60     } else {
```

```
61          return false;
62      }
63  }
64
65  public static void checkDead() {
66      // Initialisation de la liste de Combattants en vie
67      List<Combattant> temp = new ArrayList<Combattant>();
68      //Ici, on parcourt les instances de Combattant
69      for (Combattant f : Combattant.instances) {
70          // Et on fait appel à la méthode isAlive() pour vérifier que le Combattant est en vie
71          if (f.isAlive()) {
72              temp.add(f);
73          } else {
74              System.out.println(f.getName() + " est mort");
75          }
76      }
77      Combattant.instances = temp;
78  }
79
80
81  public static void checkHealth() {
82      for (Combattant f : Combattant.instances) {
83          System.out.println(f.getName() + " a encore " + f.getHealth() + " points de vie");
84      }
85      System.out.println("------------------------------");
86  }
87
88
89  public void attack (String type,Combattant other){
90      if(other.isAlive()) {
91          int damage = (Integer)Combattant.attack_modifier.get(type) * this.attack - other.getDefense();
92          other.setHealth(other.getHealth() - damage);
93          Combattant.checkDead();
94          Combattant.checkHealth();
95      }
96      else{
97          System.out.println(other.getName() + " est déjà mort");
98      }
99  }
100 }
101
102 class Soigneur extends Combattant { // a la capacité de soigner et réssuciter quelqu un
103
104     //TODO
105
106     public Soigneur(String name, int health, int attack, int defense, int soin)
107 {
108     //TODO
109 }
110
111 //TODO
112
113
114 public void résurrection(Combattant other){
115     //TODO
116 }
117
118 public void attack(Combattant other) {
119     //TODO
120 }
121 }
122
123 class Attaquant extends Combattant{ // a la capacité d attaquer deux fois
124
125     //TODO
126
127     public Attaquant(String name, int health, int attack, int defense, int multiplicateur){
128     //TODO
129 }
130
131 //TODO
132
133 public void attack(String type, Combattant other) {
```

```
134     //TODO
135 }
136 }
```

Question 3: (*5 minutes*) Sous-classe **Soigneur**

Commencez par déclarer une nouvelle sous-classe **Soigneur**. Cette sous-classe prendra un nouvel attribut **private**, **int**, 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 (**résurrection**).

Conseil

Pensez à utiliser le constructeur de votre classe mère **Combattant**

Solution

```
1   class Soigneur extends 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 4: (*10 minutes*) Méthode **résurrection(Combattant other)** de la sous-classe **Soigneur**

Commencez par déclarer une nouvelle méthode nommée **résurrection(Combattant 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 contrôler que l'instance depuis laquelle la méthode est appelée soit toujours en vie. Si ce n'est pas le cas, indiquez : **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 est égale à 1. Si ce n'est pas le cas, indiquez : **nom_instance** ne peut plus ressusciter 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 **instances** de la classe **Combattant**. Pensez aussi à :

- Mettre l'attribut **résurrection** de l'instance appelée à 0 afin de l'empêcher de réutiliser ce pouvoir,
- 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.

Pour les indications en fonction des différentes conditions, imprimez simplement la phrase en question.

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 ressuciter 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 5: (*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**, puis appelez la méthode de classe **checkHealth()**.

Conseil

Pensez à utiliser du branchement conditionnel pour les contrôles.

Le nombre de points de vie à ajouter est simplement égal à l'attaque de l'instance depuis laquelle la méthode est appelée. Ajoutez la valeur de cet attribut **attack** au **Combattant other**

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 6: (*5 minutes*) Sous-classe **Attaquant**

Commencez par déclarer une nouvelle sous-classe **Attaquant**. Cette sous-classe prendra un nouvel attribut **private**, **int**, 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 **multiplicateur**.

Conseil

Pensez à utiliser le **constructeur** de votre classe mère **Combattant**.

Solution

```
1  class Attaquant extends Combattant{
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 7: (*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 l'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**.

Comment peut-on effectuer plusieurs fois une même séquence d'action en programmation ?

Solution

```
1  public void attack(String type, Combattant other) {
2      for (int i = 0; i < this.getMultiplicateur(); i++) {
3          System.out.println("Attaque n " + (i+1));
4          super.attack(type, other);
5      }
6  }
```

Si tout est correct, en utilisant ce main :

```
1  public class Main {
2      public static void main(String[] args) {
3      Combattant P1 = new Combattant("P1", 10, 2, 2);
4      Attaquant P2 = new Attaquant("P2", 10, 2, 2,2);
5      Soigneur P3 = new Soigneur("P3",10,4,2,4);
6      P1.attack("pied",P2);
7      P1.attack("poing",P2);
8      P1.attack("tete",P2);
9      P1.attack("tete",P2);
10     P3.résurrection(P2);
11     P1.attack("pied",P2);
12     P1.attack("poing",P2);
13     P1.attack("tete",P2);
14     P3.attack(P2);
15     P2.attack("tete",P1);
16 }
17 }
```

Vous devriez obtenir :

```
1  P1 a encore 10 points de vie
2  P2 a encore 8 points de vie
3  P3 a encore 10 points de vie
4  ------------------------------
5      P1 a encore 10 points de vie
6  P2 a encore 6 points de vie
7  P3 a encore 10 points de vie
8  ------------------------------
9      P1 a encore 10 points de vie
10 P2 a encore 2 points de vie
11 P3 a encore 10 points de vie
12 ------------------------------
13     P2 est mort
14 P1 a encore 10 points de vie
15 P3 a encore 10 points de vie
16 ------------------------------
17     P2 vient de revenir à la vie
18 P1 a encore 10 points de vie
19 P3 a encore 10 points de vie
20 P2 a encore 10 points de vie
21 ------------------------------
22     P1 a encore 10 points de vie
23 P3 a encore 10 points de vie
24 P2 a encore 8 points de vie
25 ------------------------------
26     P1 a encore 10 points de vie
27 P3 a encore 10 points de vie
28 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
----------------------------------
```