Algorithmes et Pensée Computationnelle

Programmation orientée objet : Héritage et Polymorphisme

Le but de cette séance est d'approfondir les notions de programmation orientée objet vues précédemment. Nous commencerons par un rappel des notions de surcharge d'opérateurs avant d'introduire des exercices sur l'héritage et le 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 Code.

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: (O *5 minutes*) **Création de classe**

Dans un projet que vous aurez au préalable préparé, créez un fichier nommé surcharge.py. À l'intérieur de ce fichier, créez une classe Fraction qui aura comme attributs privés un numérateur et un dénominateur.

Question 2: (5 minutes) Constructeur par défaut

Définir un constructeur à votre classe. Assignez des valeurs par défaut à vos attributs.

Si un seul argument est passé à votre constructeur, la fraction devra être égale à l'entier correspondant. Empêchez l'utilisateur d'assigner la valeur zéro au dénominateur.



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 à 0 et un dénominateur à 1 soit $\frac{0}{1}$.

En Python, vous pouvez donner des valeurs par défaut aux arguments de vos méthodes lors de leur définition. Par exemple, vous pouvez faire : def add (self, x=10, y=5) : . . .

Question 3: (**O** 5 minutes) **Type casting**

Convertir les attributs en entier.



Pensez à utiliser la fonction int.

Question 4: (O *5 minutes*) **Redéfinition de méthodes**

Redéfinir la méthode __str__() pour produire une représentation textuelle de vos objets Fraction.

Conseil

Une fois la méthode __str()__ redéfinie, lorsqu'on fera un print() sur une instance de votre classe Fraction, il affichera le message suivant : *Votre fraction a pour valeur numérateur/dénominateur*. Numérateur et dénominateur étant les valeurs que vous passerez à votre objet Fraction.

Question 5: (O *5 minutes*) Accesseurs et mutateurs

Créer des getters et setters pour chacun des attributs de votre classe Fraction.

Question 6: (**1**) 10 minutes) **Simplification de fractions**

Définir une méthode simplification qui réduit la Fraction. Cette méthode ne renverra rien, elle modifiera simplement l'instance. Pour la suite des exercices, assurez-vous de toujours manipuler des fractions

simplifiées. Pour ce faire, vous pouvez faire appel à votre méthode simplification après chaque opération sur un objet de type Fraction.

Conseil

Afin de simplifier une fraction, vous devez diviser le numérateur et le dénominateur par leur plus grand diviseur commun. Pensez à utiliser la méthode math.gcd pour trouver le plus grand diviseur commun entre deux nombres.

Question 7: (15 minutes) Redéfinition de méthodes - __eq__

Redéfinir la méthode d'instance eq qui prend en entrée un objet Fraction que vous nommerez other (en plus de self) et qui renvoie True si self et l'objet passé en argument ont la même valeur.



Conseil

Pour vérifier l'égalité entre a/b et c/d, tester que a*d est égal à b*c.

Utilisez la méthode isinstance () afin de vérifier que other est bien de type Fraction. Dans le cas contraire, affichez un message d'erreur.

La fonction isinstance prend en entrée une valeur et un type. Elle vérifie que cette valeur est du type défini. Par exemple : isinstance (nombre, int) renverra True si la variable nombre est de type int et False dans le cas contraire.

Question 8: (**1** *15 minutes*) **Addition et multiplication**

Redéfinir les méthodes <u>add</u> et <u>mul</u> afin d'effectuer des opérations d'addition et de multiplication sur vos objets de type Fraction. Attention, ces méthodes devront renvoyer des objets de type Fraction. Dans vos méthodes add et mul, n'oubliez pas de simplifier ces fractions avant de les retourner. Gérer le cas où l'élément passé en argument n'est ni une Fraction, ni un int.

```
>_ Solution
   # Question 1: Création de la classe Fraction
1
2
   class Fraction:
       # Question 2: Déclaration du constructeur et initialisation des
       def __init__(self, numerateur=0, denominateur=1):
5
            # Question 3: type casting
6
            self.__num = int(numerateur)
7
           if int(denominateur) == 0:
8
                raise ZeroDivisionError("Le dénominateur ne peut pas être é
       gal à zéro")
9
            self.__den = int(denominateur)
10
           self.simplification()
11
12
       # Question 4: redéfinition de la méthode __str__()
13
       def __str__(self):
14
            return "Votre fraction a pour valeur {}/{}".format(self.__num,
       self.__den)
15
16
       # Question 5: Getters et Setters
       def get_num(self):
17
18
            return self.__num
19
20
       def get_den(self):
           return self.__den
21
22
23
       def set_num(self, n):
24
           self._num = n
25
26
       def set_den(self, d):
27
           d = int(d)
28
            # Si on passe 0 au dénominateur, on lève une exception ce qui
       arrêtera le programme
29
           if d == 0:
30
               raise ZeroDivisionError("Le dénominateur ne peut pas être é
       gal à zéro")
31
           self.\_den = d
32
33
       # Question 6
34
       def simplification(self):
35
            if self.__num == 0:
               self._den = 1
36
37
            if self.__den < 0:</pre>
38
               self.__num = -self.__num
39
                self.__den = -self.__den
40
            pgcd = math.gcd(self.__num, self.__den)
41
            self.__num = int(self.__num / pgcd)
42
            self.__den = int(self.__den / pgcd)
43
44
       # Question 7
45
       def _eq_(self, f):
46
            if isinstance(f, Fraction):
47
                # vu que les fractions sont toujours en représentation
       simplifiée, on pourrait se contenter de
48
                \# self.__numerateur == f.__numerateur and
       self.__denominateur = f.denominateur
49
               return self.__num * f.__den == f.__num * self.__den
50
            # Au cas où on recoit un seul argument, on créé une fraction
       ayant pour numérateur l'argument et 1 comme dénominateur
51
            elif isinstance(f, int):
52
               return self.__eq__(Fraction(f))
53
            else:
54
                return False
```

```
>_ Solution
1
   # Question 8
       def add(self, f):
2
3
            self.__num = self.__num * f.__den + f.__num * self.__den
4
            self.__den = self.__den * f.__den
5
            self.simplification()
6
7
       def plus(self, f):
8
           q = Fraction(self.__num, self.__den)
9
            q.add(f)
10
           return q
11
12
       def __add__(self, other):
13
            if isinstance(other, Fraction):
14
                return self.plus(other)
            elif isinstance(other, int):
15
16
                self.add__ = self.__add__ (Fraction(other))
17
                return self.add___
18
                print("Unsupported operand types for +: '" +
19
       self.__class__.__name__ + "' and '" + other.__class__.__name__ +
20
21
       def __mul__(self, other):
22
           if isinstance (other, Fraction):
23
               return Fraction(self.__num * other.__num, self.__den *
       other.__den)
24
           elif isinstance(other, int):
25
               return Fraction(self.__num * other, self.__den)
26
           # On affiche un message d'erreur lorsque other n'est pas une
       Fraction
27
          else:
               print("Unsupported operand types for *: '" +
28
       self.__class__.__name__ + "' and '" + other.__class__.__name__ +
   if __name__ == '__main__':
31
       f1 = Fraction()
32
       print(f1)
33
       f1 = Fraction(4)
34
       print(f1)
35
       f1=Fraction(numerateur=5, denominateur=0)
36
       print(f1)
37
       f1 = Fraction(denominateur=5)
38
       print(f1)
39
       f1 = Fraction(4, -6)
40
       print(f1)
41
       f2 = Fraction(2, -8)
42
       print(f2)
43
       print(f1+f2)
44
       print(f1 == Fraction(22, -24))
45
       print(f1 == Fraction(1, 2))
```

2 Notions d'héritage - Java Optionnel

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 9: (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 {
}
class Livre_Audio extends Livre {
}
class Livre_Illustre extends Livre {
}
```

Conseil

1

10 11

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 classe-fill 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 classe-fille

Vous pouvez vous servir de '\n' dans une chaîne de caractères pour effectuer un retour à la ligne lors de l'affichage d'une chaîne de caractères.

```
>_ Solution
1 public class Livre {
2
3
       private String titre;
4
       private String auteur;
5
      private int annee;
6
       private int note = -1;
7
       public Livre(String titre, String auteur, int annee) {
8
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
   public void setNote(int note) {
19
20
       this.note = note;
21
  }
22
  public String toString() {
23
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);
           System.out.println("Création d'un livre audio");
38
39
           this.narrateur = narrateur;
40
41
   }
42
43 class Livre_Illustre extends Livre {
44
45
       private String illustrateur;
46
47
       public Livre_Illustre(String titre, String auteur, int annee,
       String illustrateur) {
48
           super(titre, auteur, annee);
           System.out.println("Création d'un livre illustré");
49
50
           this.illustrateur = illustrateur;
51
52 }
```

Question 10: (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.

Redéfinir la méthode toString() de la classe Livre_Audio afin que la valeur de l'attribut narrateur

Faites pareil avec la classe Livre_Illustre et son attribut Illustrateur

Conseil

Attention, vous devez créer un objet Livre et non 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 fait référence à l'instance avec laquelle la méthode est 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 3 private String narrateur; 4 5 public Livre_Audio(String titre, String auteur, int annee, String 6 super(titre, auteur, annee); System.out.println("Création d'un livre audio"); 7 8 this.narrateur = narrateur; 9 } 10 11 // redéfinition de la fonction toString dans la classe fille Livre_Audio 12 public String toString() { 13 return super.toString() + "\nNarrateur: "+ narrateur+"\n"; //Ajoute narrateur à la chaine de caractère crée par la classe mère (super) 14 15 } 16 17 class Livre_Illustre extends Livre { 18 19 private String illustrateur; 20 21 public Livre_Illustre(String titre, String auteur, int annee, String illustrateur) { 22 super(titre, auteur, annee); System.out.println("Création d'un livre illustré"); 23 24 this.illustrateur = illustrateur; 25 } 26 public String toString() { return super.toString() + "\nIllustrateur: "+ illustrateur + 27 "\n"; //Ajoute illustrateur à la chaine de caractère crée par la classe mère (super) 28 } 29 } 1 public class Main { public static void main(String[] args) { 3 Livre Livre1 = new Livre_Audio("Hamlet", "Shakespeare", 1609, "William"); 4 Livre1.setNote(5);

System.out.println(Livre1);

5

6 7 } Lorsque toutes les étapes auront été effectuées, placez le code suivant dans votre main et exécutez votre programme :

```
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
           Livre Livre2 = new Livre("Les Misérables", "Hugo", 1862);
8
           System.out.println(Livre2);
9
10
   }
   Vous devriez obtenir:
1 Création d'un livre
2 Création d'un livre audio
3 Création d'un livre
4 A propos du livre
5
6 Titre : Hamlet
7 Auteur : Shakespeare
8 Année : 1609
9 Note: 5
10 Narrateur: William
11 A propos du livre
12
13 Titre : Les Misérables
14 Auteur : Hugo
15 Année : 1862
16 Note : non attribuée
17 Process finished with exit code 0
```

3 Polymorphisme - Java

Les exercices de cette section s'inspirent des exercices de la section 1 des exercices avancés de la semaine 6. N'hésitez pas à réutiliser les solutions disponibles sur Moodle.

Dans cette partie, vous devez créer 2 nouvelles classe-filles de la classe-mère Fighter. La première classe représentera un Soigneur, qui, lorsqu'il "attaquera" un Fighter, le soignera au lieu de le blesser. La deuxième classe représentera un Fighter spécialisé dans l'attaque Attaquant, qui aura la capacité d'attaquer un certain nombre de fois (ce nombre sera défini lors de l'instanciation).

Avant d'effectuer la série de questions suivantes, téléchargez le fichier fighter-squelette. java disponible sur Moodle.

Question 11: (5 minutes) classe-fille Soigneur

Créez une nouvelle classe-fille Soigneur qui hérite de la classe Fighter. Cette classe-fille aura un nouvel attribut privé nommé résurrection qui vaudra 1 lors de l'instanciation de la classe.

Créez le constructeur de cette classe ainsi que les getter et setter permettant d'interagir avec le nouvel attribut (résurrection).

Conseil

Pensez à utiliser le constructeur de votre classe-mère Fighter

>_ Solution class Soigneur extends Fighter { 2 3 private int résurrection; 4 5 public Soigneur (String name, int health, int attack, int defense, int soin) { 6 super(name, health, attack, defense); 7 résurrection = 1; 8 } 9 10 public int getRésurrection(){ 11 return this.résurrection; 12 13 14 public void setRésurrection(int etat) { 15 this.résurrection = etat; 16 17 }

Question 12: (15 minutes) Méthode résurrection (Fighter other) de la classe-fille Soigneur Créez une méthode résurrection (Fighter other) qui permettra de faire revenir un Fighter 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 en affichant le texte suivant : "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 Fighter other à la vie en lui remettant 10 points de vie et en l'ajoutant à la liste instances de la classe Fighter. 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 (Fighter other) a été créée dans la classe Fighter. Regardez à quoi elle sert et utilisez la.

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

>_ Solution 1 public void résurrection(Fighter 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 Fighter.addInstances(other); 14 this.setRésurrection(0); 15 System.out.println(other.getName() + " vient de revenir à la vie"); 16 Fighter.checkHealth(); 17 } 18 } 19 } 20 }

Question 13: (10 minutes) Méthode attack de la classe-fille Soigneur

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

Le seul argument nécessaire pour cette méthode sera une autre instance de Fighter qu'on pourrait nommer other.

Commencez par contrôler que le Soigneur depuis lequel la méthode est appelée est encore en vie. Si ce n'est pas le cas, affichez : "nom_instance est mort et ne peut plus rien faire."

Contrôlez ensuite que le Fighter 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 le texte suivant : "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 un 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 Fighter other.

>_ Solution public void attack(Fighter 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, ressuscitez-le pour pouvoir le soigner"); 11 12 other.setHealth(other.getHealth() + this.getAttack()); 13 Fighter.checkHealth(); 14 } 15 } 16 }

Question 14: (5 minutes) Classe-fille Attaquant

Commencez par déclarer une nouvelle classe-fille Attaquant héritant de la classe Fighter. Cette classe-fille prendra un nouvel attribut privé nommé multiplicateur de type int, qui sera passé comme argument du constructeur de la classe-fille.

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 Fighter.

```
>_ Solution
   class Attaquant extends Fighter{
1
2
3
       private int multiplicateur;
4
       public Attaquant (String name, int health, int attack, int defense,
5
       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
17 }
```

Question 15: (10 minutes) Méthode attack de la classe-fille Attaquant

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

Assurez-vous de toujours indiquer le type de l'attaque lorsque vous faites appel à la méthode attack ().

Conseil

Pensez à contrôler que l'instance depuis laquelle la méthode est appelée est encore en vie. Aidez-vous de la méthode attack de la classe-mère Fighter.

Pour effectuer plusieurs attaques, vous pouvez utiliser une boucle allant de Oà multiplicateur.

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

Si tout est correct, en utilisant ce main:

```
1
   public class Main {
2
       public static void main(String[] args) {
            Fighter P1 = new Fighter("P1", 10, 2, 2);
3
4
            Attaquant P2 = \text{new Attaquant}("P2", 10, 2, 2, 2);
5
            Soigneur P3 = new Soigneur("P3", 10, 4, 2, 4);
            P1.attack("pied", P2);
6
7
            P1.attack("poing", P2);
            P1.attack("tete", P2);
8
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
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
29
30\, P1 a encore 10 points de vie
31
  P3 a encore 10 points de vie
32
  P2 a encore 2 points de vie
33
  ______
34 P1 a encore 10 points de vie
35\, P3 a encore 10 points de vie
36\, P2 a encore 6 points de vie
37
  _____
38 Attaque n 1
39 Pl a encore 6 points de vie
40 P3 a encore 10 points de vie
41 P2 a encore 6 points de vie
42 -----
43 Attaque n 2
44 P1 a encore 2 points de vie
45 P3 a encore 10 points de vie
46 P2 a encore 6 points de vie
```