Algorithmes et Pensée Computationnelle

Programmation orientée objet - Exercices de base

Le but de cette séance est de se familiariser avec un paradigme de programmation couramment utilisé : la Programmation Orientée Objet (POO). Ce paradigme consiste en la définition et en l'interaction avec des briques logicielles appelées **Objets**. Dans les exercices suivants, nous manipulerons des objets, aborderons les notions de classe, méthodes, attributs et encapsulation. Au terme de cette séance, vous serez en mesure d'écrire des programmes mieux structurés. Afin d'atteindre ces objectifs, nous utiliserons principalement le langage **Java** qui offre une panoplie d'outils pour mieux comprendre ce paradigme de programmation.

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

1 Création de votre première classe en Java

Le but de cette première partie est de créer votre propre classe en Java. Cette classe sera une classe nommée **Dog()** représentant un chien. Elle aura plusieurs attributs et méthodes que vous implémenterez au fur et à mesure.

Question 1: (*10 minutes*) Création de classe et encapsulation
Commencez par créer une nouvelle classe **Java** dans votre projet. Ensuite, créez les attributs suivants :

1. Un attribut **public** String nommé **name**
2. Un attribut **private** List nommé **tricks**
3. Un attribut **private** String nommé **race**
4. Un attribut **private** int nommé **age**
5. Un attribut **private** int nommé **mood** initialisé à 5 (correspondant à l'humeur du chien)
6. Un attribut de classe (**static**) **private** int nommé **nb_chiens**

Créez une méthode publique du même nom que la classe (**Dog**). Cette méthode est appelée le **constructeur**, elle va servir à initialiser les différentes instances de notre classe. Un **constructeur** en **Java** aura le même nom que la classe, et le **constructeur** en **Python** sera défini par la méthode **__init__**. Cette méthode prendra en argument les éléments suivants qui seront utilisés pour initialiser les attributs de notre instance :

1. Une chaîne de caractère **name**,
2. Une liste **tricks**,
3. Une chaîne de caractère **race**,
4. Un entier **age**.

Pour finir, cette méthode doit incrémenter l'attribut de classe **nb_chiens** qui va garder en mémoire le nombre d'instances crées.

Conseil

Pour revoir les notions de base du langage Java, n'hésitez pas à consulter le guide de démarrage en Java sur Moodle : `https://moodle.unil.ch/mod/folder/view.php?id=1132337`
Pour cet exercice, n'oubliez pas de préciser si vos attributs sont **public** ou **private**.
Le mot **static** correspond à un élément de classe (attribut ou méthode), cet élément pourra ensuite être appelé via la classe directement.
Pour attribuer des valeurs à vos attributs d'instance, utilisez le mot-clé **this.attribut**.
Pour accéder aux attributs de classe, utilisez **nom_classe.nom_attribut**

Solution

```
1  public class Dog {
2      public String name;
3      private List tricks;
4      private String race;
5      private int age;
6      private int mood = 5;
7      private static int nb_chiens = 0;
8
9      public Dog(String name, List tricks, String race, int age) {
10         this.name = name;
11         this.race = race;
12         this.tricks = tricks;
13         this.age = age;
14         nb_chiens++;
15     }
16
17 }
```

Question 2: (*10 minutes*) Getters et setters

Il faut maintenant créer des méthodes de type **getter** et **setter** afin d'interagir avec les attributs **private** des instances de la classe. Les **getters** renverront les attributs souhaités tandis que les **setters** les modifieront. Les **setters** sont souvent utilisés pour modifier la valeur d'attributs privés et ne renvoient rien.

Conseil

Exemple de **getters** et de **setters** :

```
1  public String getName() { // Exemple de getter
2      return name;
3  }
4
5  public void setName(String name) { // Exemple de setter
6      this.name = name;
7  }
```

Vous pouvez directement accéder à des attributs publics en utilisant **nom_instance.attribut** à l'intérieur ou à l'extérieur de la classe.

Créez les méthodes suivantes :

- **getTricks()**
- **getRace()**
- **getAge()**
- **getMood()**
- **setTricks()**
- **setRace()**
- **setAge()**
- **setMood()**

Créez également une méthode de classe permettant de retourner le nombre de **Dog** instanciés (un **getter**).

Conseil

Intellij vous permet de générer automatiquement certaines méthodes telles que les getters et setters. Vous pouvez consulter le lien suivant pour plus d'informations : `https://www.jetbrains.com/help/idea/generating-code.html#generate-delegation-methods`.
Toutefois, pour cet exercice, nous vous encourageons à le faire manuellement.

Solution

```
1   public List getTricks() {
2       return tricks;
3   }
4
5   public int getAge() {
6       return age;
7   }
8
9   public int getMood() {
10      return mood;
11  }
12
13  public String getRace() {
14      return race;
15  }
16
17  public static int getNb_chiens() {
18      return nb_chiens;
19  }
20
21  public void setTricks(List tricks){
22      this.tricks=tricks;
23  }
24
25  public void setAge(int age) {
26      this.age = age;
27  }
28
29  public void setMood(int mood) {
30      this.mood = mood;
31  }
32
33  public void setRace(String race) {
34      this.race = race;
35  }
```

Question 3: (*5 minutes*) Manipulation d'attributs - Listes
Créez une méthode publique nommée **addTrick(String trick)** qui prend en entrée une chaîne de caractères et l'ajoute à la liste **tricks**.

Conseil

La liste **tricks** est une liste comme les autres. Si vous voulez la modifier, vous aurez besoin de passer par une **LinkedList** temporaire.

Solution

```
1  public void addTrick(String trick) {
2      LinkedList temp = new LinkedList(this.tricks);
3      temp.add(trick);
4      this.tricks = temp;
5  }
```

Question 4: (*5 minutes*) Manipulation d'attributs - **setter**
Créez deux méthodes permettant de modifier l'attribut **mood** de l'objet **Dog**. La méthode **leash()** décrémentera **mood** de 1 et **eat()** l'incrémentera de 3.

Solution

```
1    public void eat() {
2        this.mood = mood + 3;
3    }
4
5    public void leash() {
6        this.mood --;
7    }
```

Question 5: (*5 minutes*) Manipulation d'attributs d'une autre instance
Créez une méthode nommée **getOldest (Dog other)** qui prend comme argument un élément de type **Dog**, puis retourne le nom et l'âge du chien le plus agé sous le format suivant : "**nomChien** est le chien le plus agé avec **ageChien** ans".

Conseil

L'élément **Dog** que vous passez en argument est un objet de type **Dog**, vous pouvez donc lui appliquer les méthodes que vous avez créé tout à l'heure. Faites attention à la façon d'accéder aux différents attributs de votre deuxième chien (pour rappel, les attributs privés ne sont accessibles qu'à travers des **getters** que vous aurez préalablement définis).

Solution

```
1  public String getOldest(Dog other) {
2      if (other.getAge() < this.getAge()){
3          return this.name + " est le chien le plus agé avec " + this.age + " ans";
4      }
5      else{
6          return other.name + " est le chien le plus agé avec " + other.getAge() + " ans";
7      }
8  }
```

Question 6: (*5 minutes*) Redéfinition de méthodes
Créez une méthode **toString()** de type **public** qui retourne une chaîne de caractères contenant toutes les informations d'une instance de **Dog**. Ainsi, dans votre **main**, en faisant **System.out.println(...)** sur une instance de **Dog**, vous obtiendrez un texte sous le format suivant : "**nomChien** a **ageChien** ans, est un **raceChien** et a une humeur de **moodChien**. Il sait faire les tours suivants : **tricksChien**".

Informations utiles

La méthode **toString()** hérite de la super classe **Object**. La notion d'héritage sera présentée la semaine prochaine. Retenez juste qu'il est possible de choisir ce que vaudra le texte descriptif de nos objets de type **Dog**. Il est également possible de redéfinir d'autres méthodes comme par exemple l'addition ou la soustraction, ce qui permettrait de choisir comment 2 objets de type **Dog** seraient additionnés ou soustraits. Avant de redéfinir la méthode **toString()**, ajoutez l'annotation **@Override**.

Solution

```
1  public String toString(){return this.name + " a " + this.age + " ans, est un " + this.race +
2          " et a une humeur de " + this.mood + ". Il sait faire les tours suivants : " + this.tricks;}
```

Pour contrôler que vos méthodes et attributs ont été implémentés correctement, vous pouvez essayer le code suivant à l'intérieur de votre méthode **main** :

```
public class Main {
    public static void main(String[] args) {
        Dog lola = new Dog("Lola",List.of("rollover"),"Bouvier",10);
        Dog tobi = new Dog("Tobi",List.of("rollover","do a barrel"),"Doggo",17);
        System.out.println(lola.getAge());
        System.out.println(lola.getMood());
        System.out.println(lola.getRace());
        System.out.println(lola.name);
        System.out.println(lola.getTricks());
        lola.setAge(13);
        lola.setMood(8);
        lola.setRace("Bouvier");
        lola.name = "lola";
        lola.setTricks(List.of("rollover","do a barrel"));
        lola.eat();
        lola.leash();
        lola.addTrick("sit");
        System.out.println(Dog.getNbChiens());
        System.out.println(lola.getOldest(tobi));
        System.out.println(lola);
    }
}
```

Vous devriez obtenir ce résultat :

```
10
5
Bouvier
Loola
[rollover]
2
Tobi est le chien le plus agé avec 17 ans
Lola a 13 ans, est un Bouvier et a une humeur de 10. Il sait faire les tours suivants : [rollover, do a barrel, sit]
```

2 Interaction entre plusieurs instances d'une même classe

Dans cette section, nous allons simuler un jeu de combat entre deux protagonistes représentant des instances d'une classe **Fighter** que nous allons créer. Chaque **Fighter** aura des attributs qui le définissent. Ces attributs sont :

- **nom:***(String)* chaque combattant sera identifié par un nom unique.
- **health:***(int)* représentant le nombre de points de vie d'un combattant. Il contient des valeurs comprises entre 0 et 10. À l'instanciation de l'objet, le combattant a 10 points de vie par défaut-
- **attaque:***(int)* représentant une valeur qui sera utilisée pour calculer le nombre de points de dégâts infligés à l'adversaire.
- **défense:***(int)* représentant une valeur qui sera utilisée pour calculer le nombre de points de dégâts reçus.

Deux attributs de classe seront également utilisés :

- **instances :** Liste comprenant les combattants qui ont été instanciés et qui sont toujours en vie.
- **attackModifier :** Dictionnaire comportant 3 types d'attaques, chacune modifiant les dégâts qui vont être infligés. Les trois types d'attaques sont **poing, pied et tête** modifiant respectivement l'attaque par 1, 2, 3.

Le but de cette partie est d'étudier les interactions entre deux instances d'une même classe. Cette classe se présentera sous la forme d'un **Fighter**. Chaque instance de la classe **Fighter** pourra attaquer les autres instances.
Vous devrez compléter les 4 méthodes suivantes :

1. **isAlive()**
2. **checkDead()**
3. **checkHealth()**
4. **attack(String type, Fighter other)**

Voici le squelette du code (à télécharger sur 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 Fighter {
7      private String name;
8      private int health;
9      private int attack;
10     private int defense;
11     private static List<Fighter> instances = new ArrayList<Fighter>();
12     private static HashMap<String, Integer> attackModifier = new HashMap(Map.of("poing",1,"pied",2,"tete",3));
13
14     public Fighter(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
22     public int getAttack() {
23         return attack;
24     }
25
26     public int getHealth() {
27         return health;
28     }
29
30     public int getDefense() {
31         return defense;
32     }
33
34     public String getName() {
35         return name;
```

```
36      }
37
38      public void setAttack(int attack) {
39          this.attack = attack;
40      }
41
42      public void setDefense(int defense) {
43          this.defense = defense;
44      }
45
46      public void setHealth(int health) {
47          this.health = health;
48      }
49
50      public void setName(String name) {
51          this.name = name;
52      }
53
54      public Boolean isAlive() {
55          // à compléter
56      }
57
58      public static void checkDead() {
59          // à compléter
60      }
61
62      public static void checkHealth() {
63          // à compléter
64      }
65
66      public void attack (String type, Fighter other){
67          // à compléter
68      }
69  }
```

Question 7: (*5 minutes*) isAlive()

Définir une méthode **isAlive()** de type **boolean** qui retournera **true** si l'instance a plus que 0 points de vie et **false** si l'instance en a moins.

Solution

```
1       public boolean isAlive() {
2           if (this.health > 0) {
3               return true;
4           } else {
5               return false;
6           }
7       }
```

Question 8: (*10 minutes*) checkDead()

Définir une méthode **checkDead()** qui parcourt la liste des instances, et contrôle que chacune d'entre elle est encore en vie. Si ce n'est pas le cas, l'instance en question est supprimée de la liste des instances et le message "**nomInstance** est mort" sera affiché.

Conseil

Prenez le problème dans l'autre sens, créez une liste temporaire. Si l'instance est vivante, ajoutez la à cette nouvelle liste. Pour finir, mettez à jour votre liste d'instances à l'aide de votre liste temporaire.

L'attribut **instances** étant une liste, vous pouvez parcourir cette liste d'instances en utilisant une boucle **for**.

Solution

```
1  public static void checkDead() {
2    // Initialisation de la liste de Fighters en vie
3      List<Fighter> temp = new ArrayList<Fighter>();
4      //Ici, on parcourt les instances de Fighter
5      for (Fighter f : Fighter.instances) {
6        // Et on fait appel à la méthode isAlive() pour vérifier que le Fighter est en vie
7        if (f.isAlive()) {
8          temp.add(f);
9        } else {
10         System.out.println(f.getName() + " est mort");
11       }
12     }
13     Fighter.instances = temp;
14   }
```

Question 9: (*5 minutes*) checkHealth()
Définir une méthode **checkHealth()** qui parcourt la liste des instances et affiche le nombre de points de vie qui reste au combattant sous le format "**nomInstance** a encore **healthInstance** points de vie".

Solution

```
1  public static void checkHealth() {
2      for (Fighter f : Fighter.instances) {
3        System.out.println(f.getName() + " a encore " + f.getHealth() + " points de vie");
4      }
5    }
```

Question 10: (*10 minutes*) attack(String type Fighter other)
Définir une méthode **attack(String type, Fighter other)** qui permettra de retirer des points de vie au combattant **other** en fonction de l'attaque de l'instance appelée, du type d'attaque sélectionné et de la défense de **other**.

Commencez par contrôler si **other** est encore en vie.Si tel n'est pas le cas, indiquez qu'il est déjà mort : "**other_name** est déjà mort".

Si **other** est encore en vie, retirez des points de vie à **other**. Le nombre de points de vie devant être retiré se calcule en utilisant la formule suivante : **attack_modifier(type)** * **attack_instance** - **defense_other**. Appelez ensuite les fonctions **checkDead()** et **checkHealth()** afin d'avoir un aperçu des combattants restants et de leur santé.

Solution

```
1  public void attack (String type,Fighter other){
2    if(other.isAlive()) {
3      int damage = (int)Fighter.attackModifier.get(type) * this.attack - other.getDefense();
4      other.setHealth(other.getHealth() - damage);
5      Fighter.checkDead();
6      Fighter.checkHealth();
7    }
8    else{
9      System.out.println(other.getName() + " est déjà mort");
10   }
11 }
```

Pour terminer, vous pouvez exécuter le code ci-dessous (disponible dans le dossier Code sur Moodle) pour vérifier que votre programme fonctionne correctement :

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

Vous devriez obtenir ce résultat :

```
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 8 points de vie
P3 a encore 10 points de vie
P1 a encore 10 points de vie
P2 a encore 4 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

Process finished with exit code 0
```

3 Notions de POO en Python

Dans cette section, nous créerons pas-à-pas une classe **Point** contenant des attributs et des méthodes utiles. Dans votre IDE, créez un nouveau projet Python (Fichier > Nouveau > Projet). Dans un dossier de votre choix, créez un fichier **question11.py**.

Question 11: (*15 minutes*) Classe **Point**

— Créez une classe **Point** et un constructeur par défaut contenant deux paramètres (**x** et **y**).

> **Conseil**
>
> Pour rappel, un constructeur est une fonction **__init()__** que vous redéfinirez dans votre classe.

— Définissez deux attributs privés pour votre classe **Point**. Ces attributs seront les coordonnées x et y de vos points. Par défaut, assignez leur les valeurs données dans le constructeur.

> **Conseil**
>
> À l'intérieur d'une classe, utilisez le mot-clé **self** pour accéder aux méthodes et attributs de l'instance que vous manipulez.
> En Python, pour spécifier qu'un attribut est privé, rajouter un double underscore au nom de l'attribut (Exemple : **__score=0**)

— Définir des getters et setters.

> **Conseil**
>
> En Python, le mot-clé **self** est l'équivalent de **this** utilisé en Java.

— Définissez une méthode **distance** qui prend en entrée l'instance du **Point** (**self**) et un autre **Point p2**. Cette méthode **distance** retournera la distance euclidienne entre le point **self** et **p2**.

> **Conseil**
>
> Pour rappel, la distance euclidienne entre deux points est définie par la formule $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.
> Utilisez la fonction **sqrt()** de la librairie **math** pour calculer la racine carrée. Pensez à importer la libraire **math**.

— Définissez une méthode **milieu** qui prendra en entrée **self** et **p2** et qui retournera un objet **Point** situé entre **self** et **p2**.

> **Conseil**
>
> Pour trouver les coordonnées d'un point M(x_M, y_M) situé au milieu du segment défini par des points A(x_A, y_A) et B(x_B, y_B), utilisez les formules suivantes : $x_M = \frac{x_1+x_2}{2}$ et $y_M = \frac{y_1+y_2}{2}$

— Redéfinissez une méthode **__str__()** dans la classe **Point** qui retournera une chaîne de caractères contenant les coordonnées (x, y) d'un point. Ainsi, lorsqu'on fera un **print** d'une instance de la classe **Point**, le message qui s'affichera sera le suivant : *Les coordonnées du Point sont : x = "remplacez par la valeur de x" et y = "remplacez par la valeur de y"*

Solution

```
1   import math
2
3   class Point:
4       def __init__(self, x, y):
5           self.__x = x
6           self.__y = y
7
8       def get_x(self):
9           return self.__x
10
11      def get_y(self):
12          return self.__y
13
14      def set_x(self, x):
15          self.__x = x
16
17      def set_y(self, y):
18          self.__y = y
19
20      def distance(self, p2):
21          return math.sqrt((self.__x - p2.get_x())**2 + (self.__y - p2.get_y())**2)
22
23      def milieu(self, p2):
24          x_M = (self.__x + p2.get_x()) /2
25          y_M = (self.__y + p2.get_y()) / 2
26          M = Point(x_M, y_M)
27          return M
28
29      def __str__(self):
30          return "Les coordonnées du point sont: x="+str(self.get_x())+", y="+str(self.get_y())
31
32  if __name__ == '__main__':
33      p = Point(3, 2)
34      p2 = Point(5,4)
35      print(str(p.distance(p2)))
36      print(str(p.milieu(p2)))
```