Algorithmes et Pensée Computationnelle

Programmation orientée objet - Exercices basiques

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.
Les languages de programmation qui seront utilisés pour cette série d'exercices sont Java et Python.
Le temps indiqué (🕓) est à titre indicatif.

1 Création de votre première classe en Java (40 minutes)

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 **Dog** 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 caractères **name**,
2. Une liste **tricks**,
3. Une chaîne de caractères **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=1357549`.
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 }
```

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 add_trick(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**".

Conseil

La méthode **toString()** permet d'obtenir une représentation textuelle d'un objet. Elle est définie par défaut pour tout objet. Pour la redéfinir, on utilise le mot-clé **@Override** avant la méthode.

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** :

```
1  public class Main {
2      public static void main(String[] args) {
3          Dog lola = new Dog("Lola",List.of("rollover"),"Bouvier",10);
4          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 Notions de POO en Python (15 minutes)

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 **question7.py**.

Question 7: (*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)))
```

3 Notions d'héritage - Java (30 minutes)

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 8: (*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**

Voici le squelette du programme à compléter :

```
1  class Livre {
2  }
3  class Livre_Audio extends Livre {
4  }
5  class Livre_Illustre extends Livre {
6  }
```

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 classe-fille 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
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 9: (*10 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** soit affichée.
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 narrateur){
6        super(titre, auteur, annee);
7        System.out.println("Création d'un livre audio");
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);
23    System.out.println("Création d'un livre illustré");
24    this.illustrateur = illustrateur;
25 }
26    public String toString() {
27       return super.toString() + "\nIllustrateur: "+ illustrateur + "\n"; //Ajoute illustrateur à la chaine de
          caractère crée par la classe mère (super)
28    }
29 }
```

```
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  }
```

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

```
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
```

4 Notions d'héritage - Python (30 minutes)

Question 10: (*15 minutes*) **Classe Point (Suite)**

Dans la section 2, vous avez défini une classe **Point** représentant un point de 2 dimensions, avec des coordonnées **x** et **y**, ainsi que des opérations basiques sur des points 2D. Pour cet exercice, vous allez implémenter une classe de points à 3 dimensions qui héritera de la classe **Point** créée précédemment. Avant de commencer, modifiez les attributs privés de la classe **Point**. Remplacez les par des attributs protégés.
Écrivez une classe qui hérite de **Point**. Nommez la **Point3D**. Après avoir rajouté la 3ème dimension comme attribut, effectuez les opérations ci-dessous :

- Rajoutez une méthode qui renvoie une représentation vectorielle du point. Vous pouvez utiliser une liste.
- Recalculez la distance euclidienne et le milieu pour le point 3D.
- Pour aller plus loin Si vous voulez vous familiariser encore plus avec les méthodes de classe en Python, implémentez deux autres algorithmes de calculs de distance : les distances de Manhattan et Minkowski.

Vous pouvez compléter le code suivant (qui se trouve sur Moodle, dans le dossier **Code**).

```
class Point3D(Point):
    def __init__(self, x, y):
        super().__init__(x, y)
        # Rajoutez un nouvel attribut z propre à la classe Point3D


    # Définir le getter et le setter sur le nouvel attribut z

    # Complétez les méthodes suivantes
    def vector_representation(self): # représentée sous forme de liste
        ...

    def distance_euclidean(self, p2): # i.e norme
        ...

    def distance_manhattan(self, p2):
        ...

    def distance_minkowski(self, p2, order=3):
        ...

    def milieu(self, p2):
        ...
```

Conseil

- En Python, pour définir un attribut protégé, on le précède d'un seul underscore (par exemple `_x = 45` est un attribut protégé).
- L'instruction `super().__init__(x, y)` permet de faire appel au constructeur de la classe-mère.
- Dans un espace à 3 dimensions, la formule pour calculer la distance entre un $p_1 = (x_1, y_1, z_1)$ et $p_2 = (x_2, y_2, z_2)$ est $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (y_1 - y_2)^2}$.

Solution

```
class Point3D(Point):
    def __init__(self, x, y, z):
        super().__init__(x, y)
        self.__z = z

    def get_z(self):
        return self.__z

    def set_z(self, z):
        self.__z = z

    def vector_representation(self): # représentée sous forme de liste
        return [self.__x, self.__y, self.__z]

    def distance_euclidean(self, p2): # i.e norme
        other_x = p2.get_x()
        other_y = p2.get_y()
        other_z = p2.get_z()
        return math.sqrt((self.__x - other_x)**2 + (self.__y - other_y)**2 + (self.__z - other_z)**2)

    def distance_manhattan(self, p2):
        other_x = p2.get_x()
        other_y = p2.get_y()
        other_z = p2.get_z()
        return sum((abs(self.__x - other_x), abs()), abs(self.__y - other_y), abs(self.__z - other_z))

    def distance_minkowski(self, p2, order=3):
        other_x = p2.get_x()
        other_y = p2.get_y()
        other_z = p2.get_z()
        return sum((abs(self.__x - other_x)**order), abs(self.__y - other_y)**order,\
                abs(self.__z - other_z)**order)**(1/order)

    def milieu(self, p2):
        other_x = p2.get_x()
        other_y = p2.get_y()
        other_z = p2.get_z()

        x_M = (self.__x + other_x)/2
        y_M = (self.__y + other_y)/2
        z_M = (self.__z + other_z)/2
        return Point3D(x_M, y_M, z_M) # renvoie un point!


point1 = Point3D(1, 2, 3)
point2 = Point3D(3, 4, 5)

# exemple
point1.vector_representation()
```