Algorithmes et Pensée Computationnelle

Classes abstraites et interfaces

Les exercices sont construits autour des concepts d'héritage, de classes abstraites et d'interfaces. Au terme de cette séance, vous devez être en mesure de différencier une classe abstraite d'une interface, savoir à quel moment utiliser l'un ou l'autre, utiliser le concept d'héritage multiple, factoriser votre code afin de le rendre mieux structuré et plus lisible.
Cette série d'exercices est divisée en 3 sections dont les premières portant sur les classes abstraites et les interfaces. La dernière section comporte des exercices pratiques sur les notions abordées précédemment.
Les exercices doivent être faits uniquement en **Java**.
Le code présenté dans les énoncés se trouve sur Moodle, dans le dossier **Ressources**.

1 Programmation Orientée Objet

Question 1: (🕓 *10 minutes*) Encapsulation - Java
L'encapsulation sert à cacher les détails d'implémentation. L'encapsulation sert uniquement à montrer que les informations essentielles aux utilisateurs.

En Java, il est recommandé de déclarer les attributs des classes comme étant **private** et de mettre à disposition des utilisateurs des méthodes publiques d'accès afin qu'ils puissent accéder ou modifier la valeur des attributs privés.

Les méthodes publiques d'accès comme **getName**et **setName** doivent être nommées avec soit **get** soit **set** suivi du nom de l'attribut avec la 1ère lettre en majuscule (Java Naming convention)[https ://www.oracle.com/java/technologies/javase/codeconve namingconventions.html]).
Le mot clé **this** fait référence à l'objet en question.

1. Dans votre IDE, créez une classe **Person**,
2. Ajoutez-y un attribut privé **name**,
3. Créez un getter et un setter pour l'attribut **name** en suivant la convention de nommage des méthodes.
4. Dans votre **main**, créez une instance de **Person**,
5. En utilisant le setter défini précédemment, donnez un nom (**name**) à votre instance.
6. Affichez le nom de l'instance en utilisant le getter de l'attribut **name**.

Solution

```
1  public class Main {
2      public static class Person {
3          private String name;
4
5          public String getName() {
6              return name;
7          }
8
9          public void setName(String newName) {
10             this.name = newName;
11         }
12     }
13
14     public static void main(String[] args) {
15         Person myObj = new Person();
16         myObj.setName("John");
17         System.out.println(myObj.getName());
18     }
19 }
```

Question 2: (🕓 *10 minutes*) Héritage - Java

Comme vous le savez, il est possible que des classes-filles héritent des attributs ou méthodes de classes-mères. En Java, il faut utiliser le mot-clé **extends** lorsqu'on défini une classe-fille. Ainsi, l'héritage permet la réutilisation des attributs et méthodes d'une classe existante.

1. Créez une classe-mère **Vehicle** ayant un attribut protégé appelé **brand**.
2. Créez un constructeur pour la classe **Vehicle** et assignez une valeur à l'attribut **brand**.
3. Définissez une méthode **honk** qui affiche **"Tuut, tuut!"**
4. Créez une classe-fille **Car** qui hérite de **Vehicle** et ayant pour attribut **modelName** avec pour valeur par défaut **"Mustang"**.

Conseil

Dans le constructeur de la classe-fille, n'oubliez pas de faire appel au constructeur de la classe-mère en utilisant le mot-clé **super**

Solution

```
1  public class Main {
2      public static class Vehicle {
3          protected String brand;
4
5          public Vehicle(String brand) {
6              this.brand = brand;
7          }
8
9          public void honk() {
10             System.out.println("Tuut, tuut!");
11         }
12     }
13     public static class Car extends Vehicle{
14
15         public String modelName = "Mustang";
16         public Car(String brand) {
17             super(brand);
18         }
19     }
20
21     public static void main(String[] args) {
22         Car myCar = new Car("Ford");
23         myCar.honk();
24         System.out.println(myCar.brand + " "+myCar.modelName);
25     }
26 }
```

2 Exercices complémentaires

Question 3: (*10 minutes*) **Programmation de base**

Ecrivez un programme Python qui imprime tous les nombres impairs à partir de 1 jusqu'à un nombre **n** défini par l'utilisateur. Ce nombre **n** doit être supérieur à 1. Exemple : si n = 6, résultat attendu : 1, 3, 5

Solution

```
def nombresImpairs(limite):
    for nb in range(limite+1):
        if nb % 2 == 1:
            print(nb)

limit = int(input("Entrez une valeur maximale: "))

print("Nombres impairs compris entre 1 et " + str(limit) + " : ")
nombresImpairs(limit)
```