TP1 - JAVA
Méthodes de classe et méthodes d'instance

Bachour Peter

10 Avril 2020

Table des matières

Table des figures

1 Introduction

Dans le cadre du premier TP en JAVA, nous allons nous focaliser sur les méthodes de classe et les méthodes d'instance.

2 Définition

Une méthode est une fonction qui va s'appliquer sur :

1. une classe.
2. une instance de classe.

Une **méthode de classe** c'est une simple fonction. Elle peut prendre des paramètres et peut retourner une valeur. On peut ici citer le constructeur d'une classe qui est une méthode qui porte le même nom que la classe et qui ne retourne aucune valeur (void). Il est utilisé pour l'instantion d'un objet.

```java
public Ville() {
    this.nom = "inconnu";
    this.superficie = 0;
    this.population = 0;
}

public Ville(String nom, double superficie, int population) {
    this.nom = nom;
    this.superficie = superficie;
    this.population = population;
}
```

FIGURE 1: Constructeur de la classe Ville.

Une **méthode d'instance** s'utilise sur l'instance d'une classe, appelée aussi objet. L'instance d'une classe c'est quand on utilise la forme suivant :

```java
Ville dijon = new Ville( nom: "Dijon",  superficie: 40.41,  population: 375831);
Ville quetigny = new Ville( nom: "Quetigny",  superficie: 8.19 ,  population: 9690);
Ville beaune = new Ville( nom: "Beaune",  superficie: 31.3,  population: 52741);
Ville macon = new Ville( nom: "Mâcon",  superficie: 27.0,  population: 100172);
Ville chalon = new Ville( nom: "Chalon-sur-Saône",  superficie: 15.22,  population: 133557);
```

FIGURE 2: Instances d'une Ville.

Pour l'appeler il faudra faire **objet.maMethode()**, elle peut biensûr prendre des paramètres et retourner quelque chose et peut modifier l'objet.

3 Création de la classe Ville

3.1 Constructeur

Comme nous avons pu le remarquer dans la Figure 1, la classe Ville comporte 2 constructeurs. Le premier constructeur ne prends pas de paramètres. C'est un constructeur par défaut. Alors que le second a 3 paramètres qui correspond aux attributs de la classe Ville.

3.2 Accesseurs et mutateurs

Afin de pouvoir accéder aux attributs d'une classe qui sont le plus souvent privés,

```
private String nom;
private double superficie;
private int population;
```

FIGURE 3: Attributs privés de la classe Ville.

on aura besoin d'accesseurs et de mutateurs qui d'une part nous permette d'accéder aux attributs en dehors de la portée de la classe. Ces derniers peuvent retourner la valeur de l'attributs ou de la changer.

```
public String getNom() { return nom; }

public void setNom(String nom) { this.nom = nom; }

public double getSuperficie() { return superficie; }

public void setSuperficie(double superficie) { this.superficie = superficie; }

public double getPopulation() { return population; }

public void setPopulation(int population) { this.population = population; }
```

FIGURE 4: Accesseurs et mutateurs de la classe Ville.

3.3 toString()

Afin d'avoir l'affichage d'une classe sous un format prédefinit, on aura recours à la méthode **toString()** qui va surcharger la méthode **System.out.println**.

```java
@Override
public String toString() {
    return nom + ", " +
            superficie + " kilomètres carrés, pour " +
            population + " habitants.";
}
```

FIGURE 5: Méthode toString() de la classe Ville

4 Création de la classe Département

Pour ce qui concerne le constructeur et les accesseurs et mutateurs de la classe Département, la procédure est pareil à celle de la classe Ville.

4.1 Méthode ajouterVille()

```java
public void ajouterVille(Ville ville){
    if(this.nbVillesSaisies < this.nbVilles){
        tabVilles[this.nbVillesSaisies] = ville;
        this.nbVillesSaisies++;
    }else{
        System.out.println("Le département a atteint le nombre " +
                "maximale de ville accordées. \r\nLa ville de " + ville.getNom() +
                " ne pourra pas etre ajouter à ce département.");
    }
}
```

FIGURE 6: Méthode ajouterVille() de la classe Département

Cette méthode permet d'ajouter une ville à un département si le nombre de ville déjà saisies est inférieur au nombre de villes prévu pour ce département. Sinon, un message d'erreur est affiché. De plus, si la ville a été ajouter au département, le compteur de ville déjà saisies sera augmenter de 1.

4.2 Méthode toString()

Pour la classe département, on utilisera la même méthodologie que la classe Ville. En ce qui concerne la boucle **for**, c'est pour afficher les villes du départe-

```java
@java.lang.Override
public java.lang.String toString() {
    String x=  "Villes du département " + nom + "(" + numero + ") : \r\n" ;
    for (int i = 0; i < tabVilles.length; i++) {
        x+= (i+1) + " " + tabVilles[i] + "\r\n";
    };
    return x;
}
```

FIGURE 7: Méthode toString() de la classe Département

ment qui sont dans le tableau **tabVilles** de la classe département.

5 Classe Main

La classe Main contient une méthode main. La méthode **main** est le point d'entrée d'une application Java classique. Elle est exécutée par la Java virtual machine (JVM) directement, si la classe qui la contient est fournie à cette dernière. La méthode est exécutée de manière séquentielle, c'est à dire que la méthode exécute les commandes ligne par ligne. Après avoir créer les cinq objets

```java
public class Main {

    public static void main(String[] args) {
        Ville dijon = new Ville( nom: "Dijon",  superficie: 40.41,  population: 375831);
        Ville quetigny = new Ville( nom: "Quetigny",  superficie: 8.19 ,  population: 9690);
        Ville beaune = new Ville( nom: "Beaune",  superficie: 31.3,  population: 52741);
        Ville macon = new Ville( nom: "Mâcon",  superficie: 27.0,  population: 100172);
        Ville chalon = new Ville( nom: "Chalon-sur-Saône",  superficie: 15.22,  population: 133557);

        Departement coteDor = new Departement( nom: "Côte d'or",  numero: 21,  nbVilles: 3);
        Departement saoneEtLoire = new Departement( nom: "Saône-et-Loire",  numero: 71,  nbVilles: 2);

        coteDor.ajouterVille(dijon);
        coteDor.ajouterVille(quetigny);
        coteDor.ajouterVille(beaune);
        coteDor.ajouterVille(macon);

        saoneEtLoire.ajouterVille(macon);
        saoneEtLoire.ajouterVille(chalon);

        System.out.println(coteDor);
        System.out.println(" ");
        System.out.println(saoneEtLoire);
    }
}
```

FIGURE 8: Méthode main de la classe Main

de type Ville, les deux départements. Et après avoir ajouter les villes concernées au départements, l'affichage est le suivant :

```
Villes du département Côte d'or(21) :
1 Dijon, 40.41 kilomètres carrés, pour 375831 habitants.
2 Quetigny, 8.19 kilomètres carrés, pour 9690 habitants.
3 Beaune, 31.3 kilomètres carrés, pour 52741 habitants.


Villes du département Saône-et-Loire(71) :
1 Mâcon, 27.0 kilomètres carrés, pour 100172 habitants.
2 Chalon-sur-Saône, 15.22 kilomètres carrés, pour 133557 habitants.
```

FIGURE 9: Affichage du main

6 Attributs et méthodes de classe *vs* attributs et méthodes d'instance

6.1 Méthode statique

Une méthode statique appartient à une classe et une méthode non statique appartient à un objet d'une classe. Les méthodes statiques sont utiles si on n'utilise qu'une seule fois la méthode et qu'on n'a pas besoin de plusieurs objets. Les méthodes non statiques sont utilisées si on va utiliser notre méthode pour créer plusieurs objets. De ceci, si on ajoute le mot static lors de la déclaration de l'attribut nom, l'attribut **nom** deviendra static donc il sera unique pour toutes les instances de notre classe. En d'autre termes, l'attribut **nom** de n'importe quelle instance sera identique à toutes les autres :

```
Villes du département Côte d'or(21) :
1 Chalon-sur-Saône, 40.41 kilomètres carrés, pour 375831 habitants.
2 Chalon-sur-Saône, 8.19 kilomètres carrés, pour 9690 habitants.
3 Chalon-sur-Saône, 31.3 kilomètres carrés, pour 52741 habitants.


Villes du département Saône-et-Loire(71) :
1 Chalon-sur-Saône, 27.0 kilomètres carrés, pour 100172 habitants.
2 Chalon-sur-Saône, 15.22 kilomètres carrés, pour 133557 habitants.
```

FIGURE 10: Ajout du mot **static** à l'attribut nom de la classe Ville

6.2 *estIdentiqueA* et *sontIdentiques*

```java
public boolean estIdentiqueA(Ville ville){
    if(this.nom == ville.nom &&
            this.population == ville.population &&
            this.superficie == ville.superficie )
        return true;
    return false;
}
public static boolean sontIdentiques(Ville ville1, Ville ville2){
    if(ville1.nom == ville2.nom &&
            ville1.population == ville2.population &&
            ville1.superficie == ville2.superficie )
        return true;
    return false;
}
```

FIGURE 11: Méthodes pour vérifier si deux Villes sont identiques

Afin de vérifier si deux villes sont identiques, il faudra vérifier les attributs de chaque classe. Comme expliquer ci-dessus, la méthode statique est une méthode qui peut être utiliser sans créer un objet et l'instancier. Donc la méthode **sontIdentiques** pourrait être statique alors que la méthode **estIdentiqueA** ne peut pas être statique parcequ'elle utilise les attributs d'une instance, l'instance qui appelle la méthode.

7 Conclusion

En conclusion, à travers ce TP, nous avons découvert les méthodes de classe et les méthodes d'instance. De plus, on a découvert la fonctionalité du mot **static** devant un object ou une méthode.