Une première classe

Java est langage orienté objet. Cela signifie que (presque) tout est un objet. La définition d'un objet s'appelle une classe. Donc programmer en Java revient à **déclarer** des classes, à **instancier** des objets à partir des classes déclarées ou fournies et à effectuer des opérations sur ces objets.

Déclarer une classe

Dans ce chapitre, nous allons ébaucher l'implémentation d'une classe Voiture. La classe Voiture sera une représentation abstraite d'une voiture pour les besoins de notre application.

En Java, une classe est déclarée dans son propre fichier qui **doit** porter le même nom que la classe avec l'extension *.java*. Il nous faut donc créer le fichier **Voiture.java** :

```
/**
* Une première classe représentant une voiture
*
* @author David Gayerie
*/
public class Voiture {


}
```

Anatomie d'une classe

En Java une classe est déclarée par le mot-clé **class** suivi du nom de la classe. Nous reviendrons plus tard sur le mot-clé **public** qui précède et qui permet de préciser la portée (**scope**) de la définition de cette classe. Ensuite, on ouvre un bloc avec des accolades pour déclarer le contenu de la classe.

La déclaration d'une classe peut contenir :

des attributs
: Les attributs représentent l'état interne d'un objet. Par exemple, notre voiture peut avoir un attribut pour mémoriser sa vitesse.

des méthodes

Les méthodes représentent les opérations que l'on peut effectuer sur un objet de cette classe.

des constantes
: Les constantes sont un moyen de nommer des valeurs particulières utiles à la compréhension du code.

des énumérations
: Les énumérations sont des listes figées d'objets. Nous y reviendrons dans un chapitre ultérieur.

des classes internes
: Un classe peut contenir la déclaration d'autres classes que l'on appelle alors classes internes (**inner classes**). Nous y reviendrons dans un chapitre ultérieur.

L'ordre dans lequel apparaissent ces éléments dans la déclaration de la classe est sans importance en Java. Pour des raisons de commodité de lecture, les développeurs adoptent en général une convention : d'abord les constantes, puis les énumérations, puis les attributs et enfin les méthodes.

Ajouter des méthodes

Ajoutons quelques méthodes à notre classe **Voiture**. Nous allons commencer par ajouter la méthode **getVitesse** qui permet de connaître la vitesse actuelle d'une voiture en km/h.

```
/**
* Une première classe représentant une voiture
*
* @author David Gayerie
*/
public class Voiture {

  /**
   * @return La vitesse en km/h de la voiture
   */
  public float getVitesse() {

  }

}
```

Une méthode est identifée par sa **signature**. La signature d'une méthode est de la forme :

```
[portée] [type de retour] [identifiant] ([liste des paramètres]) {
  [code]
}
```

Pour la méthode que nous venons de déclarer:

Signature de la méthode

portée	public
type de retour	float (nombre à virgule flottante)
identifiant	getVitesse
liste des paramètres	aucun

Le code source précédent ne compilera pas, en effet, Java étant un langage fortement typé, nous sommes obligé d'indiquer le type de retour de la méthode *getVitesse* et donc le compilateur s'attend à ce que cette méthode retourne un nombre à virgule flottante. La vitesse de la voiture est typiquement une information qui correspond à l'état de la voiture à un moment donné. Il est donc intéressant de stocker cette information comme attribut de la classe :

```
/**
 * Une première classe représentant une voiture
 *
 * @author David Gayerie
 */
public class Voiture {

  private float vitesse;

  /**
   * @return La vitesse en km/h de la voiture
   */
  public float getVitesse() {
    return vitesse;
  }

}
```

Un attribut est identifé par :

```
[portée] [type] [identifiant];
```

Pour l'attribut *vitesse*, nous spécifions le type de portée **private**. En Java, un attribut a toujours une valeur par défaut qui dépend de son type. Pour le type **float**, la valeur par défaut est 0.

Nous pouvons maintenant enrichir notre classe avec des méthodes supplémentaires :

```
/**
 * Une première classe représentant une voiture
 *
 * @author David Gayerie
 */
public class Voiture {

  private float vitesse;

  /**
   * @return La vitesse en km/h de la voiture
   */
  public float getVitesse() {
    return vitesse;
  }

  /**
   * Pour accélérer la voiture
   * @param deltaVitesse Le vitesse supplémentaire
   */
  public void accelerer(float deltaVitesse) {
    vitesse = vitesse + deltaVitesse;
  }

  /**
   * Pour décélérer la voiture
   * @param deltaVitesse Le vitesse à soustraire
   */
  public void decelerer(float deltaVitesse) {
    vitesse = vitesse - deltaVitesse;
  }

  /**
   * Freiner la voiture.
   */
  public void freiner() {
```

```
    vitesse = 0;
  }

  /**
   * Représentation de l'objet sous la forme
   * d'une chaîne de caractères.
   */
  public String toString() {
    return "La voiture roule actuellement à " + vitesse + " km/h.";
  }
}
```

Les méthodes *Voiture.accelerer(float)* et *Voiture.decelerer(float)* prennent toutes les deux un paramètre de type **float**. Comme ces méthodes ne retournent aucune valeur, nous sommes obligés de l'indiquer avec le mot-clé **void**.

La méthode *toString()* est une méthode particulière. Elle s'agit de la méthode que le compilateur doit appeler s'il veut obtenir une représentation de l'objet sous la forme d'une chaîne de caractères. Notez, que l'opérateur **+** est utilisé en Java pour concaténer les chaînes de caractères et qu'il est possible de concaténer des chaînes de caractères avec d'autres types. Dans notre exemple, nous concaténons une chaîne de caractères avec le nombre à virgule flottante représentant la vitesse de la voiture.

> Java ne supporte pas la notion de fonction. Il n'est donc pas possible de déclarer des méthodes en dehors d'une classe.

La méthode main

Si nous voulons utiliser notre classe dans un programme, il nous faut déterminer un point d'entrée pour l'exécution du programme. Un point d'entrée est représenté par la méthode **main** qui doit avoir la signature suivante :

```
public static void main(String[] args) {
}
```

Une classe ne peut déclarer qu'une seule méthode **main**. En revanche, toutes les classes peuvent déclarer une méthode **main**. Cela signifie qu'une application Java peut avoir plusieurs points d'entrée (ce qui peut se révéler très pratique). Voilà pourquoi la commande **java** attend comme paramètre le nom d'une classe qui doit déclarer une méthode **main**.

Ajoutons une méthode **main** à la classe Voiture pour réaliser une programme très simple :

```
/**
 * Une première classe représentant une voiture
 *
 * @author David Gayerie
 */
public class Voiture {

  private float vitesse;

  /**
   * @return La vitesse en km/h de la voiture
   */
  public float getVitesse() {
    return vitesse;
  }

  /**
   * Pour accélérer la voiture
   * @param deltaVitesse Le vitesse supplémentaire
   */
  public void accelerer(float deltaVitesse) {
    vitesse = vitesse + deltaVitesse;
  }

  /**
   * Pour décélérer la voiture
   * @param deltaVitesse Le vitesse à soustraire
   */
  public void decelerer(float deltaVitesse) {
    vitesse = vitesse - deltaVitesse;
  }

  /**
   * Freiner la voiture.
   */
  public void freiner() {
    vitesse = 0;
  }

  /**
```

```
    * Représentation de l'objet sous la forme
    * d'une chaîne de caractères.
    */
   public String toString() {
     return "La voiture roule actuellement à " + vitesse + " km/h.";
   }

   public static void main(String[] args) {
     Voiture voiture = new Voiture();

     System.out.println(voiture);

     voiture.accelerer(110);
     System.out.println(voiture);

     voiture.decelerer(20);
     System.out.println(voiture);

     voiture.freiner();
     System.out.println(voiture);
   }
}
```

À la ligne 49, le code commence par créer une **instance** de la classe Voiture. Une classe représente une abstraction ou, si vous préférez, un schéma de ce qu'est une Voiture pour notre programme. Évidemment, une voiture peut aussi être définie pas sa couleur, sa marque, son prix, les caractéristiques techniques de son moteur. Mais faisons l'hypothèse que, dans le cadre de notre programme, seule la vitesse aura un intérêt. Voilà pourquoi notre classe Voiture n'est qu'une abstraction du concept de Voiture.

Si dans notre programme, nous voulons interagir avec une voiture nous devons créer une instance de la classe Voiture. Cette instance (que l'on appelle plus simplement un objet) dispose de son propre espace mémoire qui contient son état, c'est-à-dire la liste de ses attributs. Créer une instance d'un objet se fait grâce au mot-clé **new**.

> Remarquez l'utilisation des parenthèses avec le mot-clé **new** :

```
Voiture voiture = new Voiture();
```

Ces parenthèses sont obligatoires.

En Java, l'opérateur **.** sert à accéder aux attributs ou aux méthodes d'un objet. Donc si on dispose d'une variable *voiture* de type *Voiture*, on peut appeler sa méthode *accelerer* grâce à cet opérateur :

```
voiture.accelerer(90);
```

Aux lignes 51, 54, 57 et 60, nous utilisons la classe System pour afficher du texte sur la sortie standard. Notez que nous ne créons pas d'instance de la classe System avec l'opérateur **new**. Il s'agit d'un cas particulier sur lequel nous reviendrons lorsque nous aborderons les méthodes et les attributs de classe. Nous utilisons l'attribut de classe **out** de la classe System qui représente la sortie standard et nous appelons sa méthode println qui affiche le texte passé en paramètre suivi d'un saut de ligne. Cependant, nous ne passons pas une chaîne de caractères comme paramètre mais directement une instance de notre classe Voiture. Dans ce cas, la méthode println appellera la méthode *Voiture.toString()* pour obtenir une représentation textuelle de l'objet.

Exécuter le programme en ligne de commandes

Dans un terminal, en se rendant dans le répertoire contenant le fichier Java, il est possible de le compiler

```
$ javac Voiture.java
```

et de lancer le programme

```
$ java Voiture
```

Ce qui affichera sur la sortie suivante :

```
La voiture roule actuellement à 0.0 km/h
La voiture roule actuellement à 110.0 km/h
La voiture roule actuellement à 90.0 km/h
La voiture roule actuellement à 0.0 km/h
```

Created 2025-02-10 11:45:20 UTC by Nicolas
Updated 2025-02-10 12:13:14 UTC by Nicolas