JAVA – HOWTO 1
Concevoir et implémenter une classe

Université de Mons-Hainaut

En Java, vous allez devoir (très!) souvent concevoir et implémenter une classe. Par exemple, nous allons concevoir une classe `Car` pour représenter une voiture.

Etape 1 *Déterminer ce que vous devez faire avec un objet de la classe*

Par exemple, pour la classe `Car`, vous n'allez pas modéliser chaque fonctionnalité d'une voiture réelle, il y en a beaucoup trop. Vous devez choisir *quels aspects* d'une voiture doivent être simulés par votre classe. Faites une liste, en français, des opérations qu'un objet de votre classe doit pouvoir exécuter, comme celle-ci :

- Ajouter de l'essence.
- Rouler : effectuer une certaine distance.
- Vérifier le niveau d'essence du réservoir.

Etape 2 *Trouver des noms pour vos méthodes*

Choisissez des noms pour vos méthodes et appliquez-les sur un objet de test :

```
Car monTacot = new Car(. . .);
monTacot.addGas(20);           // ajouter 20 litres d'essence
monTacot.drive(100);           // rouler 100 kilometres
monTacot.getGas();             // vérifier le niveau du reservoir
```

Etape 3 *Documenter l'interface publique*

Voici la documentation, avec les commentaires qui décrivent la classe et ses méthodes.

```
/**
   Classe Car
   Une voiture roule et consomme de l'essence
*/
public class Car
{
    /**
       Ajouter de l'essence
       @param amount le montant (en litres) d'essence a ajouter
    */
    public void addGas(double amout)
    {
    }
    /**
       Rouler sur une certaine distance, consommant de l'essence
       @param distance la distance parcourue
    */
    public void drive(double distance)
    {
    }
    /**
       Connaitre le niveau du reservoir
       @return le nombre de litres d'essence restante
    */
    public double getGas()
    {
    }
}
```

Etape 4 *Déterminer les variables (champs) d'instance*

Demandez-vous quelle information un objet doit retenir pour effectuer son travail. N'oubliez pas que les méthodes peuvent être appelée dans n'importe quel ordre! Un objet a besoin de suffisament de "mémoire interne" pour être capable d'exécuter chaque méthode, en utilisant ses variables d'instance et les paramètres des méthodes. Regardez chaque méthode, peut-être en commençant par la plus simple ou la plus intéressante, et demandez-vous ce que vous avez besoin pour effectuer correctement cette méthode. Créer des champs d'instance pour stocker l'information dont une méthode a besoin.

Dans notre exemple, nous avons besoin de connaître (ou de calculer) le nombre de litres d'essence dans le réservoir (méthode `getGas`). Cela a donc du sens pour un objet `Car` de conserver cette information :

```
public class Car
{
   . . .
   private double gas;
}
```

La méthode `addGas` va simplement ajouter quelque chose à cette valeur. La méthode `drive` devra réduire cette valeur. De combien ? Cela dépend de la consommation de la voiture. Si vous roulez 100 kilomètres, et que la voiture peut rouler 20 kilomètres avec un litre d'essence, alors 5 litres sont consommés. Nous n'avons pas mis la consommation comme paramètre de la méthode `drive`, la voiture (du moins un objet `Car`) doit donc la stocker :

```
public class Car
{
   . . .
   private double gas;
   private double consumption;
}
```

Etape 5 *Déterminer les constructeurs*

Demandez-vous ce dont vous avez besoin pour construire un objet. Souvent, vous pouvez simplement mettre tous les champs d'instance à 0 ou à une valeur constante. Parfois, vous avez besoin d'information importante. Dans ce cas, vous avez besoin de mettre cette information dans un constructeur. Vous voudrez parfois deux constucteurs : un pour mettre tous les champs à une valeur par défaut, l'autre pour laisser la possibilité à l'utilisateur de leur donner des valeurs.

Dans l'exemple de la classe `Car`, nous pouvons commencer avec une voiture dont le réservoir est vide, mais nous avons besoin de connaîatre sa consommation. Il n'y a pas de bonne valeur par défaut pour celle-ci, cela devra donc être un paramètre de construction. Il est fréquent de préfixer les noms des paramètres de construction par "un" ou "une" ("a" ou "an" en anglais), de telle sorte qu'il n'y ait pas de conflit avec les noms des variables d'instance.

```
/**
   Construit une voiture avec une consommation donnee
   @param aConsumption la consommation de la voiture
*/
public Car(double aConsumption)
{
}
```

Etape 6 *Implémenter les méthodes*

Implémentez les méthodes et les constructeurs de votre classe, un par un, commençant par les plus faciles. Si vous avez certaines difficultés lors de l'implémentation, il faudra peut-être revenir à une étape précédente. Peut-être votre ensemble de méthodes de la première à la troisième étape n'étaient pas bon ? Peut-être n'avez-vous pas les bonnes variables d'instance ? Il est courant pour un débutant d'avoir quelques problèmes qui nécessitent de revenir en arrière.

Etape 7 *Tester votre classe*

Ecrivez un petit programme de test et exécuter-le. Le programme de test peut simplement faire appel aux méthodes que vous avez trouvée à l'étape 2.

```
/**
  Test de la classe Car
*/
public class CarTest
{
   public static void main(String[] args)
   {
      Car monTacot = new Car(20);          // 20 kilometres par litres
      monTacot.addGas(20);                 // ajouter 20 litres d'essence
      monTacot.drive(100);                 // rouler 100 kilometres
      double gasLeft = monTacot.getGas();  // vérifier le niveau du reservoir
      System.out.println(gasLeft);
   }
}
```