Découverte

Gestion des objets POO : Trains

Rédigé par

David ROUMANET
Professeur BTS SIO

Changement

Date	Révision

Sommaire

Nomenclature :

- **Assimiler** : cours pur. Explication théorique et détaillée (globalement supérieur à 4 pages).
- **Décoder** : fiche de cours, généralement inférieure à 5 pages.
- **Découvrir** : Travaux dirigés. Faisable sans matériel.
- **Explorer** : Travaux pratiques. Nécessite du matériel ou des logiciels.
- **Mission** : Projet encadré ou partie d'un projet.
- **Voyager** : Projet en autonomie totale. Environnement ouvert : Vous êtes le capitaine !

A Introduction

La compréhension des notions de programmation orientée objet implique de pratiquer et de se heurter à la réalité de fonctionnement des objets.

A.1 Présentation

Nous allons programmer la gestion d'un système ferroviaire, avec des trains, des conducteurs et des gestionnaires.

L'application sera très simple, pour que la notion pratique soit la compréhension des objets et pas autre chose.

A.2 Cahier des charges

Le principe est de créer des trains, de les conduire d'un point A (source) à un point B (destination) par un conducteur (nom et matricule).

Seuls les matricules supérieurs à 1000 sont des gestionnaires.

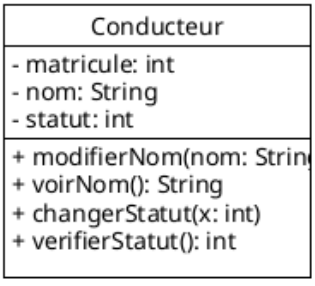
Le conducteur renseigne les passagers du train sur le pourcentage de trajet effectué (dans un système réel, ce serait fait par l'ordinateur de bord, mais ici, nous simplifions).

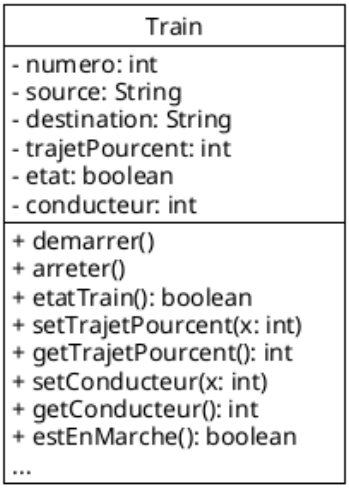
Les passagers du train pourront consulter l'avancement du trajet, mais pas le modifier.

Un train peut être à l'arrêt, ou en fonctionnement, c'est le conducteur qui gère cela.

A.3 Vision UML du cahier des charges

Les diagrammes de classe d'UML sont très pratiques pour voir les interactions entre les classes.





La représentation UML est, pour le moment, simplifié.

B Création de l'application

Dans l'éditeur IntelliJ, vous allez créer un nouveau projet Java "Train" et vous allez créer un package "model" dans lequel nous mettrons les deux classes "Train" et "Conducteur".

B.1 Création du projet

Dans le menu **File** → **New** → **Project...** puis choisir **Java** → **Java Projects**

Saisir le nom du projet : **StationProject**

Dans le gestionnaire de projet, cliquer sur **src** et faire un clic droit **New** → **Package**

Nommez le package "**model**" et validez.

Sélectionnez maintenant votre package "model" et faites un clic droit, **New** → **Class** pour créer la classe "**Train**".

B.1.1 Classe Train

> Complétez le code ci-dessous avec les éléments du cahier des charges. Essayez de compléter certaines méthodes vous-même. Respectez les informations dans le diagramme UML.

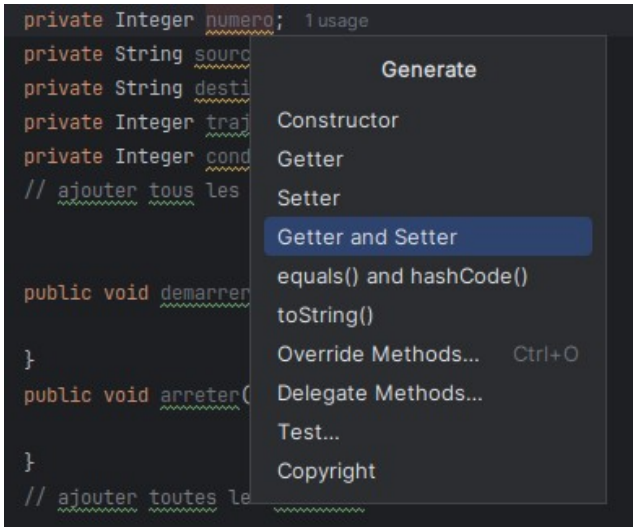
```
package model;

public class Train {
    private Integer numero;
    private String source;
    private String destination;
    private Integer trajetPourcent;
    // ajouter tous les attributs


    public void demarrer() {

    }
    public void arreter() {

    }
    // ajouter toutes les méthodes
}
```

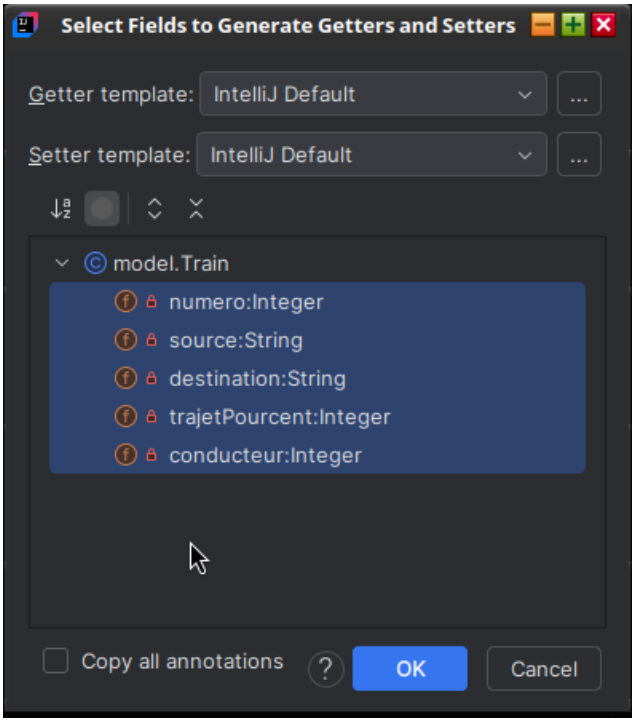
Nous allons également créer un constructeur pour instancier un train rapidement : les éléments importants sont le **numéro** du train, la ville **source** et la ville **destination** et le **conducteur** affecté au trajet.

> Générez automatiquement le constructeur en suivant les instructions : cliquez sur un attribut, puis clic droit Generate... > Constructor. Sélectionnez les attributs concernés puis cliquez sur le bouton [Ok].

Vous pouvez notamment définir les accesseurs et les mutateurs. Un moyen simple pour gagner du temps, est de le faire faire par IntelliJ : normalement, les attributs sont soulignés en jaune (pour indiquer qu'ils ne sont pas utilisés).

Un clic gauche sur un attribut, puis **Refactor... > Getter and Setter**. Dans la fenêtre de dialogue qui s'affiche, sélectionner tous les attributs, puis [OK].





> Renommez les méthodes avec les noms choisis dans le diagramme UML.
>
> Précisément, les mutateurs `demarrer()` et `arreter()` changent la valeur de l'attribut 'etat' à **true** (démarrer) ou **false** (arrêter). etatTrain() est un accesseur de cet attribut.

Note : pour donner un état booléen, IntelliJ préfère utiliser un accesseur ayant un nom **isEtat()**. Cela représente une question "is etat true ?" (est-ce que etat est vrai) ? Ainsi, la réponse est false s'il est faux et true s'il est vrai. Le nom de la méthode étant très expressif, on peut conserver ce nom (et ne pas respecter à 100 % le cahier des charges fournit).

Pour rappel, les attributs commencent par une minuscule, ainsi que les méthodes. Seules les classes ont des noms commençant par une majuscule. Les constantes s'écrivent entièrement en majuscules.

Dans la méthode setTrajetPourcent(), nous allons ajouter un commentaire un peu spécifique, que l'éditeur saura repérer facilement :

```
// TODO rédiger la gestion par le conducteur uniquement
```

> Observez la couleur de ce commentaire : est-il similaire aux autres commentaires ?

> Dans l'IDE IntelliJ, cliquez sur l'icône représentant trois points (à gauche, sous l'icône dossier et sous l'icône avec trois cubes) et choisissez TODO : que vous affiche l'IDE ?

Ainsi, il est pratique pour un développeur de mettre des "pense-bête" dans son code, s'il n'a pas le temps de rédiger un code ou s'il doit y revenir plus tard. Un autre mot-clé similaire à TODO est FIXME qui signifie qu'il y a un code bogué à cet endroit.

> Cliquez sur l'icône ToDo en bas à gauche (au-dessus de l'icône du terminal) pour faire disparaître la fenêtre.

B.1.2 Classe Conducteur

De la même manière que nous avons créé la classe Train, il faut créer la classe Conducteur.

> Réalisez la classe Conducteur, en vous inspirant des astuces vues pour la création de la classe Train, en particulier, l'automatisation des constructeurs et des accesseurs et mutateurs.
>
> Le constructeur le plus important ici, prendra comme paramètre numero et nom.

Astuce : pour rendre votre code plus compact et lisible (parce que simple), vous pouvez corriger les getters et setters pour qu'ils tiennent sur une seule ligne :

```
public Integer getMatricule() { return numero; }
public void setMatricule(Integer numero) { this.numero = numero;  }
```

B.1.3 Classe StationProject

Il faut vérifier que nous pouvons utiliser les deux classes dans notre programme principal. Pour le moment, la méthode statique main() ne contient rien d'utile, voici comment la modifier :

Avant la classe StationProject, il faut importer nos classes dans le paquetage model :

```
import model.Conducteur;
import model.Train;
```

Ensuite, il faut vérifier qu'un appel aux constructeurs de chaque classe fonctionne :

```
Conducteur Roger = new Conducteur(0001, "Roger");
Conducteur Bob = new Conducteur(1001, "Bob");

Train TGV = new Train(1394, "Paris", "Lille", Roger);
```

Il devrait y avoir une erreur sur le constructeur Train, au niveau de l'attribut Roger et voici l'explication :

Roger est ici un objet Conducteur, mais dans notre constructeur de la classe Train, le quatrième paramètre est un Integer !

> Que faut-il faire pour corriger ce code ?

Il faut corriger la partie `Integer conducteur` par le type `Conducteur conducteur` :

```
public Train(Integer numero, String source, String destination, Conducteur conducteur)
```

Cette fois, c'est le **this.conducteur** qui devient souligné en rouge.

> Comprenez-vous pourquoi ? Corrigez l'erreur sur l'attribut conducteur.

Il faut désormais corriger toutes les lignes où conducteur est utilisé comme étant un Integer et remplacer le type Integer par le type Conducteur. Par chance, la classe Conducteur est dans le même paquetage que la classe Train, Java retrouve rapidement le bon type dès que vous tapez "Cond"+[Tab].

Si votre travail est correct, vous devriez pouvoir compiler le programme, même si pour le moment, il n'affiche rien.

Précision : tant qu'il y a une alerte rouge dans les icônes en haut à droite du code de votre projet, il n'est pas possible de compiler le programme, sans erreurs.

B.2 Amélioration du projet

Pour le moment, n'importe qui peut modifier le conducteur d'un train : la méthode `setConducteur(Conducteur conducteur)` ne prend pour paramètre que le nouveau conducteur.

Imaginons que pour changer de conducteur, il faut fournir une authentification supplémentaire, en fournissant – par exemple – une personne ayant le rôle pour le faire… un gestionnaire.

Train.java

```
    public boolean setConducteur(Conducteur cheminot, Conducteur gest) {
         if (gest.getMatricule() > 1000) {
              this.conducteur = cheminot;
              return true;
         }
         return false;
    }
```

Ici, les gestionnaires sont reconnus par leur identifiant > 1000. La méthode setConducteur() renverra vrai si la modification a été effectuée et faux si l'objet **gest** n'est pas un conducteur-gestionnaire.

Voici le code pour tester un changement de conducteur :

StationProject.java

```
    System.out.println("-----------------------------");
    System.out.println("Le train N°"+TGV.getNumero()+ " au départ de "+TGV.getSource()+" et à
destination de "+TGV.getDestination());
    System.out.format("A pour conducteur %s%n",TGV.getConducteur().getNom());

    if (TGV.setConducteur(Roger, Bob)) {
         System.out.println("Le nouveau conducteur est "+TGV.getConducteur().getNom());
    } else {
         System.out.println("Erreur - Droit de modification interdit");
    }
```

Ajoutez deux conducteurs Patrick et Franck, et testez les combinaisons suivantes :

- TGV.setConducteur(Patrick, Bob)
- TGV.setConducteur(Franck, Roger)

B.2.1 Sécurisation

Il est temps de revenir sur nos oublis :

Recherchez la liste des TODO en cours et pour la méthode concernée, ajoutez un paramètre conducteur et vérifiez que le conducteur du paramètre est bien celui qui conduit le train.

De même, seul le conducteur du train doit pouvoir démarrer ou arrêter le train : effectuez les modifications.

Petite aide, les modifications portent sur la classe Train.

B.2.2 Distance à parcourir

Afin de préparer une application pratique de nos classes, nous allons ajouter un attribut **distance** (integer) à la classe Train. Nous supposerons que la saisie de la distance est automatique, même si nous utiliserons la méthode setDistance(Integer distance). Nous allons également prévoir un moment de départ, allant de 0h à 23h, attribut **heureDepart** de type Integer aussi.

Ajoutez les deux attributs à la classe Train, ainsi que les accesseurs et mutateurs associés.

Note : il est préférable de bien mettre vos méthodes après les attributs. Le fait de corriger le code s'appelle refactoriser son code. La refactorisation de code consiste à restructurer le code informatique sans modifier son comportement externe ou sa fonctionnalité.

C Jouer au train

On entend souvent dire que les enfants jouaient au train lorsqu'ils étaient petits. C'est le cas de Sheldon dans The Big Bang Theory.

Nous allons essayer de simuler un réseau de train avec quelques trains que nous initialiserons dans une table dynamique (arraylist) et une boucle qui fonctionnera tant qu'il y aura des trains en cours de déplacement (getTrajetPourcent entre 0 et 99%)