

TP 2 : Programmer une classe Robot

Dans ce TP, nous allons apprendre à coder une première classe servant à modéliser un robot et qui à terme, servira à modéliser l'usine de production du simulateur que vous devez produire dans le cadre du projet de ce cours.

Parce que les logiciels de programmation connaissent mieux la langue anglaise que les autres langues, il peut être compliqué de travailler avec des noms de classe, de méthodes ou d'attributs écrits en français. C'est pourquoi vous utiliserez toujours des mots anglais dans votre code.

Internet est une référence pour la programmation en Java. Vous y trouverez des exemples de programmation pour tout ce que vous voudrez. Il suffit d'utiliser un moteur de recherche en utilisant les bons mots clé. Vous y trouverez également des tutoriels sur l'utilisation d'Eclipse dont la grande majorité sont en anglais. Si votre version d'Eclipse est en français, cela vous demandera parfois quelques efforts pour vous y retrouver.

Lancer Eclipse à partir des menus de votre système d'exploitation. A l'aide du menu *File>>New>>Java Project*, créer un projet nommé *robotsim*. Les sources de ce projet seront à archiver dans le répertoire Git qui vous sera fourni et constitueront votre projet de développement à rendre en fin de cours.

Dans le dossier *src* du projet *robotsim*, tel que fait pour la classe *HelloWorld* du TP précédent, créer une classe nommée *Robot*. Cette classe servira à modéliser la notion de robot contenu dans une usine de production de rondelles telle que celle du projet que vous développerez pour le cours.

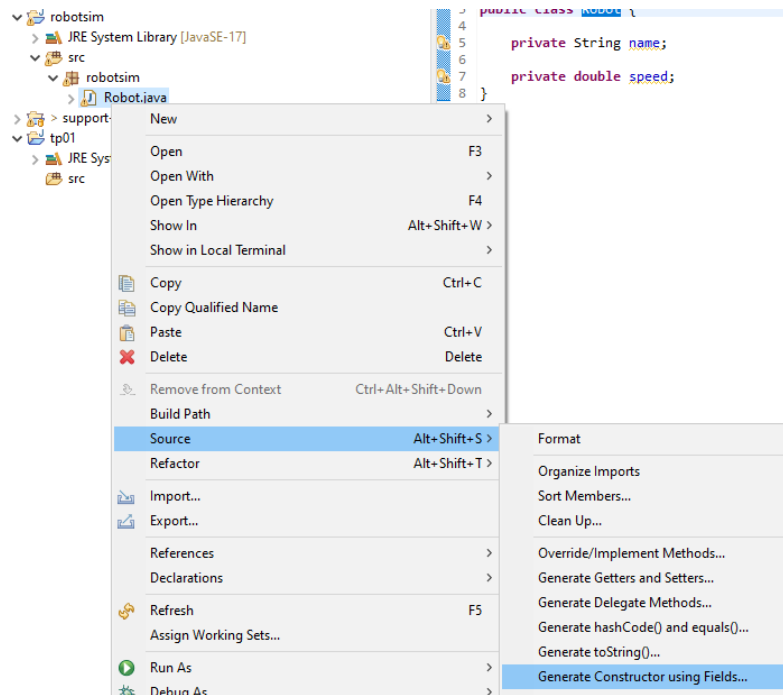
Parmi les attributs de la classe *Robot*, on doit avoir :

- Un attribut nommé **name** de type **String**.
- Un attribut nommé **speed** de type **double**.

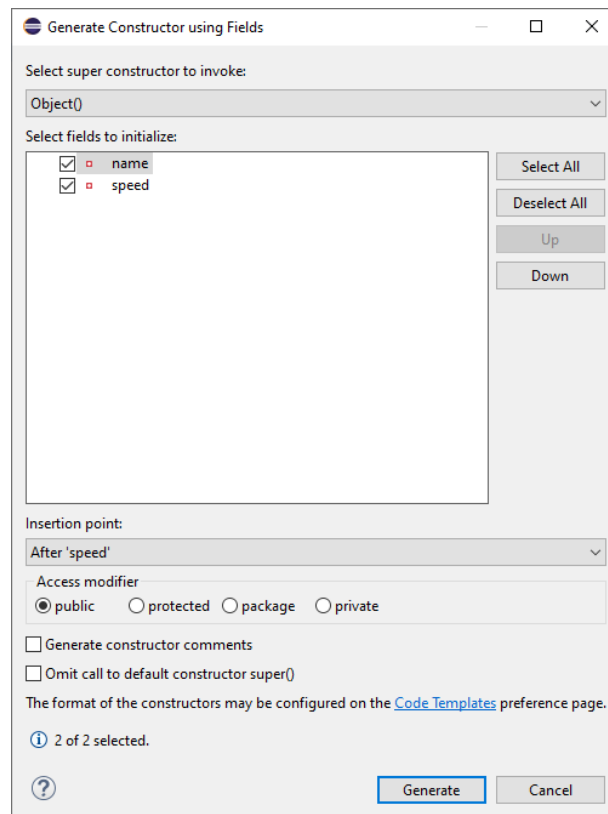
Utiliser l'éditeur de code Java pour déclarer ces attributs dans la classe *Robot*.

Nous allons maintenant écrire un **constructeur** pour cette classe. Ce constructeur devra initialiser tous les champs. Vous pouvez coder ce constructeur directement dans l'éditeur de la classe ou alors utiliser l'IDE :

1. Faire un clic droit dans la fenêtre d'édition de la classe *Robot* ou sur le fichier *Robot.java* dans l'explorateur de package.
2. Un menu apparaît ; amenez la souris sur *Source*.
3. Un second menu apparaît ; cliquez sur *Generate Constructor using Fields* (ne sélectionnez pas *Generate Constructor from Superclass* ; cette notion n'a pas encore été abordée dans le cours).

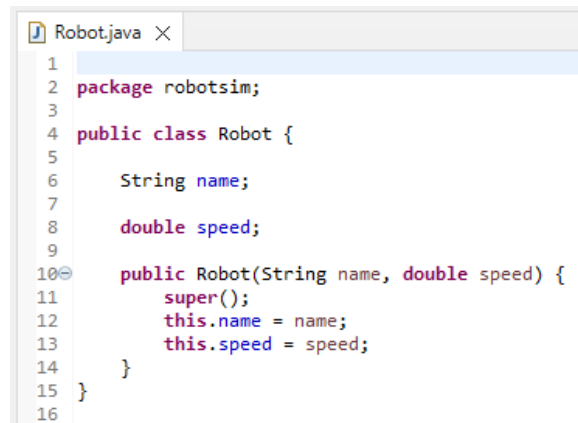


Une fenêtre apparaît qui vous propose de personnaliser le constructeur :



Vérifier que les deux attributs sont sélectionnés. Le point d'insertion du constructeur dans la classe peut être laissé tel quel ou sélectionné comme étant par exemple après la déclaration des attributs. Cliquer sur le bouton *Generate*.

On obtient alors une classe modifiée :



```
1 package robotsim;
2
3
4 public class Robot {
5
6     String name;
7
8     double speed;
9
10    public Robot(String name, double speed) {
11        super();
12        this.name = name;
13        this.speed = speed;
14    }
15 }
16
```

Avant de quitter une classe pour aller en éditer une autre, il est fortement conseillé de sauvegarder la classe en cours d'édition. Le bouton de sauvegarde est situé dans la barre de menu.

A présent, générons une seconde classe appelée *TestRobotSim* et contenant une méthode *main* comme dans la classe *HelloWorld* du premier TP. Dans cette méthode *main*, créer un objet de type *Robot* avec l'instruction :

```
Robot myRobot = new Robot("Robot 1", 5);
```

Puis demandez l'affichage de l'objet dans la console avec l'instruction :

```
System.out.println(myRobot);
```

Puis exécutez le programme. Que remarquez-vous à l'affichage de la classe ?

On voit que Java ne sait pas afficher un objet de la classe *Robot*. Il sait afficher des chaînes de caractères, des nombres, etc. Mais il ne connaît pas la classe *Robot* qui est notre invention. Quand Java ne sait pas afficher un objet, il affiche le nom de la classe de l'objet, ici *Robot*, suivi du symbole *@* (arobase), suivi de l'adresse en mémoire de l'objet exprimée en hexadécimal (base 16).

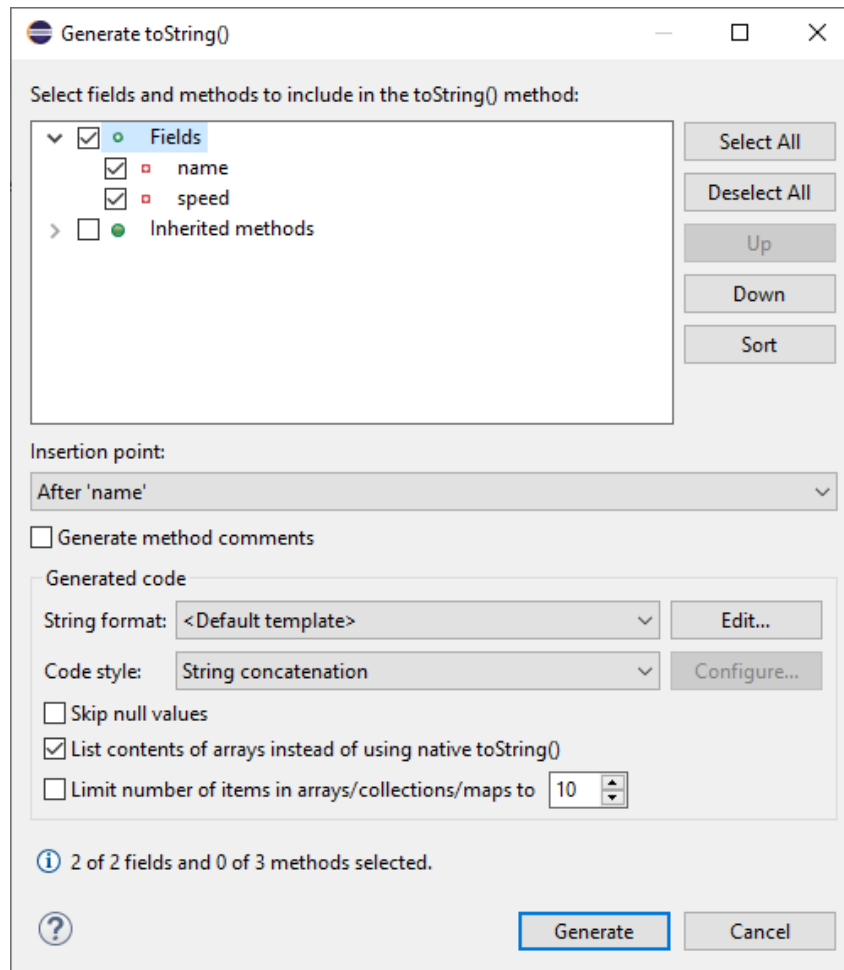
Si l'on veut que Java soit capable d'afficher un objet de la classe correctement, il faut lui fournir une méthode qui donne l'affichage sous la forme d'une chaîne de caractères que la fonction *System.out.println(...)* pourra appeler pour afficher l'objet. Cette méthode a l'en-tête suivante :

```
public String toString()
```

Pour générer cette méthode dans la classe, vous pouvez utiliser l'IDE :

1. Clic droit sur la fenêtre d'édition de la classe *Robot*.
2. Un menu apparaît. Amenez la souris sur *Source*.
3. Un second menu apparaît ; cliquez sur *Generate toString()*.

Une fenêtre apparaît pour vous proposer la génération d'une méthode *toString* :



La méthode qui sera générée calculera une chaîne de caractères qui comprendra le nom de la classe suivi de la valeur des attributs. Sur cette fenêtre, il est possible d'ajouter d'autres propriétés à l'affichage. Cliquer sur le bouton *Generate*. La classe est modifiée :

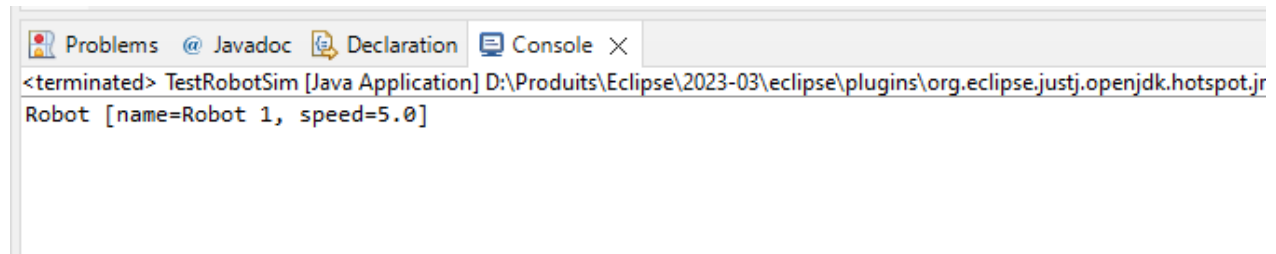
```

Robot.java x
1 package robotsim;
2
3 public class Robot {
4
5     private String name;
6
7     private double speed;
8
9     public Robot(String name, double speed) {
10         super();
11         this.name = name;
12         this.speed = speed;
13     }
14
15     @Override
16     public String toString() {
17         return "Robot [name=" + name + ", speed=" + speed + "]";
18     }
19 }
20

```

L'annotation **@Override**, placée juste avant la méthode, indique que cette méthode est la redéfinition d'une méthode héritée. Nous verrons cela en cours plus tard.

Exécuter à nouveau le programme. Est-ce que l’affichage de la classe est plus compréhensible et utile ? Vous devriez obtenir ceci dans la vue *Console* :



```
<terminated> TestRobotSim [Java Application] D:\Produits\Eclipse\2023-03\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jr
Robot [name=Robot 1, speed=5.0]
```

Redéfinir l’affichage des élèves

Exécutez votre programme du TP précédent pour vérifier que tout fonctionne encore bien.

Modifiez la méthode **toString()** de la classe pour qu’elle renvoie la chaîne de caractères ainsi formée :

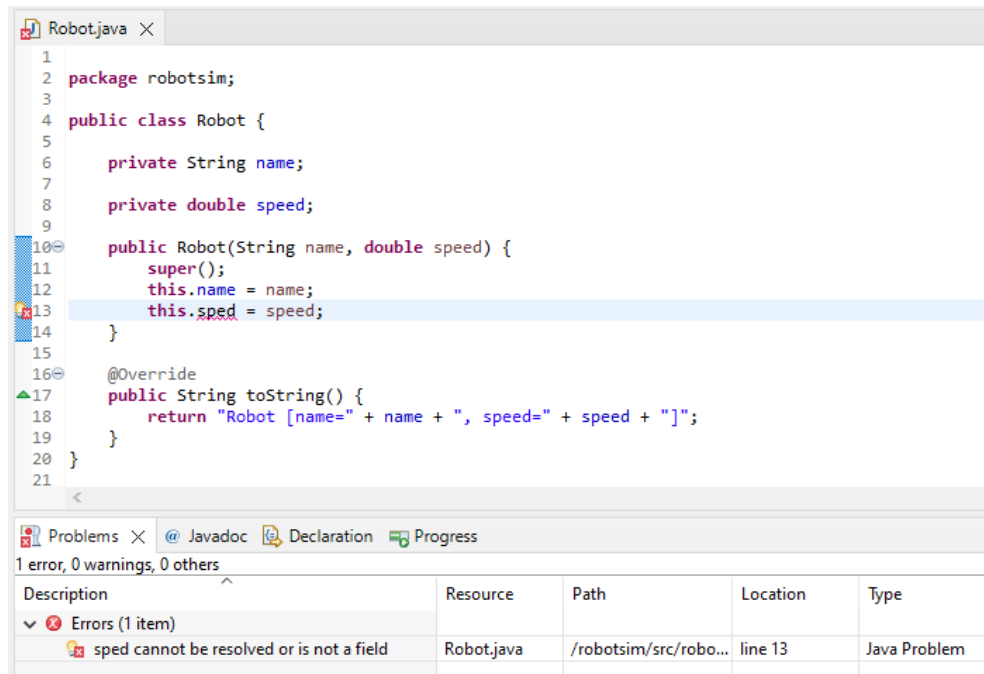
« Je m’appelle Robot 1 et j’avance à 5.0 km/h. »

Exécutez votre programme et vérifiez que le tout fonctionne correctement.

Utilité d’un IDE

Dans ce TP, nous n’avons finalement écrit que peu de code par nous-même. Ce ne sera pas toujours le cas. L’environnement de développement nous permet de réaliser des tâches de codage standards (génération de constructeurs, getters, setters, toString, etc.) en quelques clics de souris.

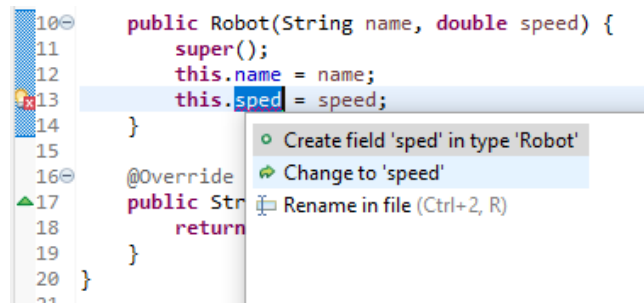
L’IDE vous signale également les erreurs et avertissements (warnings) de compilations via la vue *Problems* de la capture d’écran suivante. Dans cette vue, un double-clic sur une erreur de la liste vous amènera directement à la position de l’erreur dans l’éditeur de code Java.



```
1 package robotsim;
2
3
4 public class Robot {
5
6     private String name;
7
8     private double speed;
9
10    public Robot(String name, double speed) {
11        super();
12        this.name = name;
13        this.sped = speed;
14    }
15
16    @Override
17    public String toString() {
18        return "Robot [name=" + name + ", speed=" + speed + "]";
19    }
20 }
21
```

Description	Resource	Path	Location	Type
sped cannot be resolved or is not a field	Robot.java	/robotsim/src/robo...	line 13	Java Problem

Par ailleurs, dans la marge de l'éditeur de code, des suggestions de correction d'erreurs peuvent être proposées par l'IDE en cliquant sur l'ampoule tel qu'illustré par la capture d'écran de la fenêtre suivante. Dans cet exemple, l'attribut *speed* n'a pas été écrit correctement.



Il est également possible de renommer automatiquement des éléments de code tels que les noms de classes ou de variables (refactoring).

Toutes ces fonctionnalités sont très utiles en environnement de développement industriel car cela permet d'améliorer grandement la productivité du programmeur. Il ne faut pas hésiter à vous en servir, bien que vous deviez toujours comprendre le code généré. En effet, l'IDE ne programmera pas d'algorithmes pour vous, bien que des extensions telles que [Copilot](#) (outil payant) utilisant l'intelligence artificielle pourraient éventuellement être utiles.