TP02: Programming a Robot class

Dominique Blouin, Télécom Paris, Institut Polytechnique de Paris
dominique.blouin@telecom-paris.fr

In this assignment, we will learn to code a first class used to model a robot, which will ultimately serve to model the production factory in the simulator you must produce as part of this course's project.

Because programming software is more familiar with English than other languages, it can be complicated to work with `class`, `method`, or `attribute` names written in French. This is why you will always use English words in your code.

The internet is a reference for Java programming. You will find programming examples for everything you want. Just use a search engine with the right keywords. You will also find tutorials on using Eclipse, the vast majority of which are in English. If your version of Eclipse is in French, it may require some effort to find your way around.

1 Getting started with the `robotism` project

1. Launch Eclipse from your operating system's menus.
2. Using the `File` >> `New` >> `Java Project` menu, create a project named `robotsim`.

Note 1: The sources for this project should be archived in a Git repository that will be provided to you and will constitute your development project to be submitted at the end of the course.

Creating a `Robot` class

3. In the `src` folder of the `robotsim` project, as done for the `HelloWorld` class in the previous assignment, create a class named `Robot`. This class will be used to model the concept of a robot within a washer production factory, similar to the one you will develop for the course project.
4. Among the attributes of the `Robot` class, we must have:

- An attribute named `name` of type `String`.
- An attribute named `speed` of type `double`.

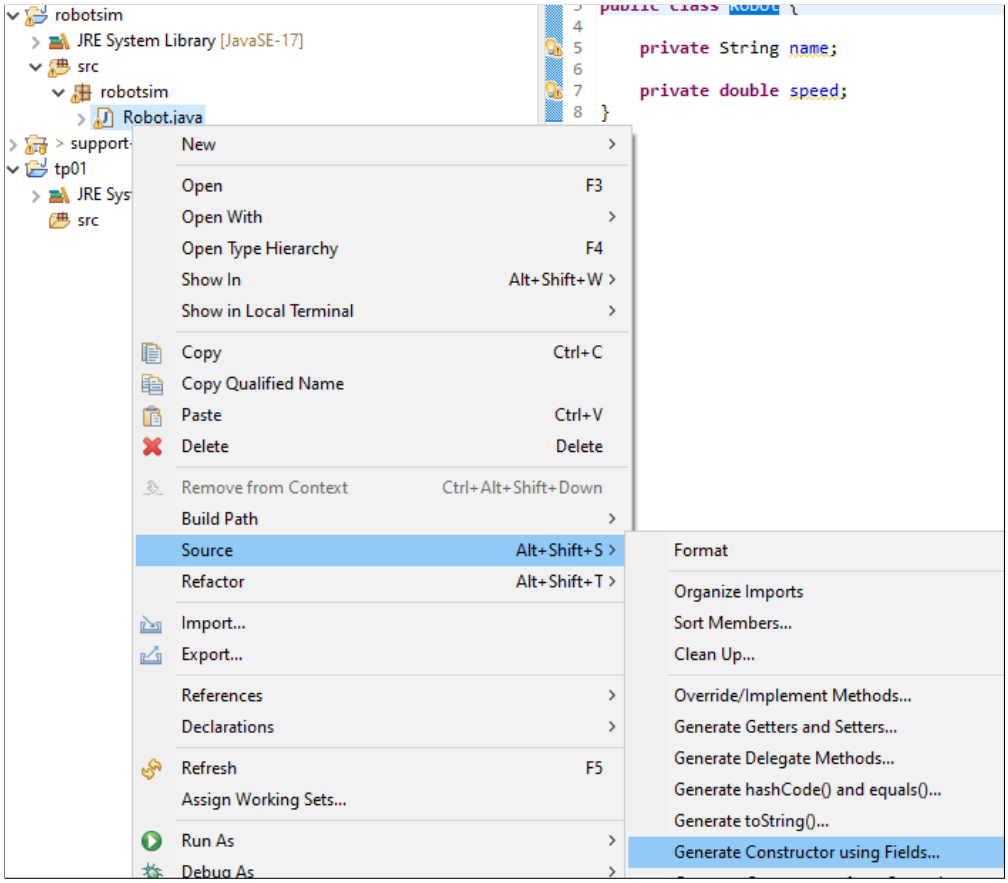
Use the Java code editor to declare these attributes in the `Robot` class.

5. We will now write a **constructor** for this class. This constructor should initialize all fields. You can code this constructor directly in the class editor or use the IDE:

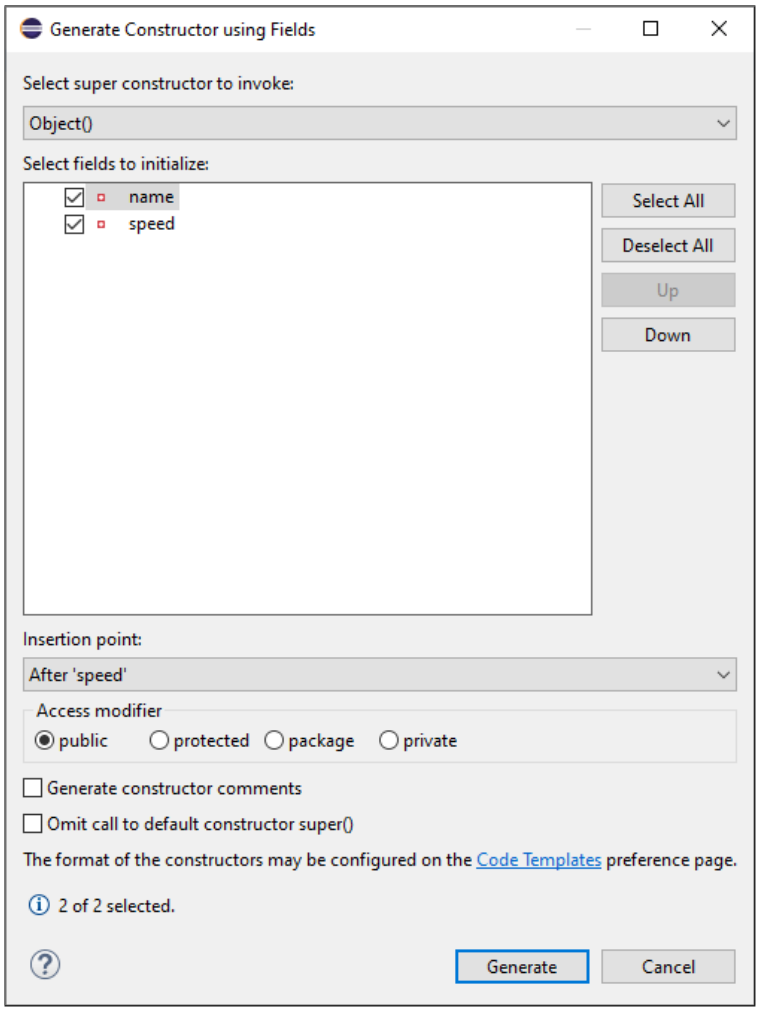
   (a) Right-click in the editing window of the `Robot` class or on the `Robot.java` file in the `package explorer`.

   (b) A menu appears; move the mouse over `Source`.

   (c) A second menu appears; click on `Generate Constructor using Fields` (do not select `Generate Constructor from Superclass`; this concept has not yet been covered in the course).

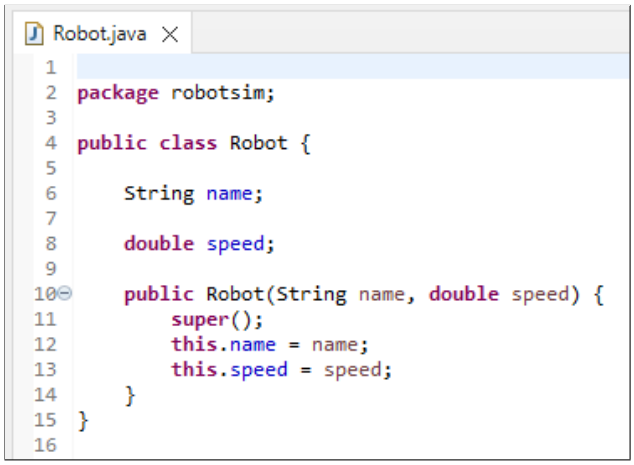


6. A window appears that allows you to customize the constructor:



7. Verify that both attributes are selected. The `Insertion point` of the constructor in the class can be left as is or selected to be, for example, *after* the declaration of the attributes.

8. Click on the `Generate` button. You then get a modified class:



9. Before leaving a class to edit another, it is strongly recommended to save the class currently being edited. The `save` button is located in the menu bar of Eclipse.

Creating a `TestRobotSim` class

10. Now, let's generate a second class called `TestRobotSim` containing a `main` method, like in the `HelloWorld` class from the first lab (TP01).

11. In this `main` method, create an `object` of type `Robot` with the instruction:

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

12. Then request the display of the `object` in the console with the instruction:

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

13. Then run the program. What do you notice about the output of the class?

Customizing object display with `toString()`

We can see that Java does not know how to display an object of the `Robot` class. It knows how to display *strings*, *numbers*, etc., but it does not know the `Robot` class, which is our invention. When Java does not know how to display an object, it displays the name of the object's class, here `Robot`, followed by the @ symbol (*at* sign), followed by the memory address of the object expressed in hexadecimal (base 16).

If we want Java to be able to display an object of the class correctly, we need to provide it with a method that returns the display in the form of a string, which the `System.out.println(...)` function can call to display / print the object. This method has the following header:

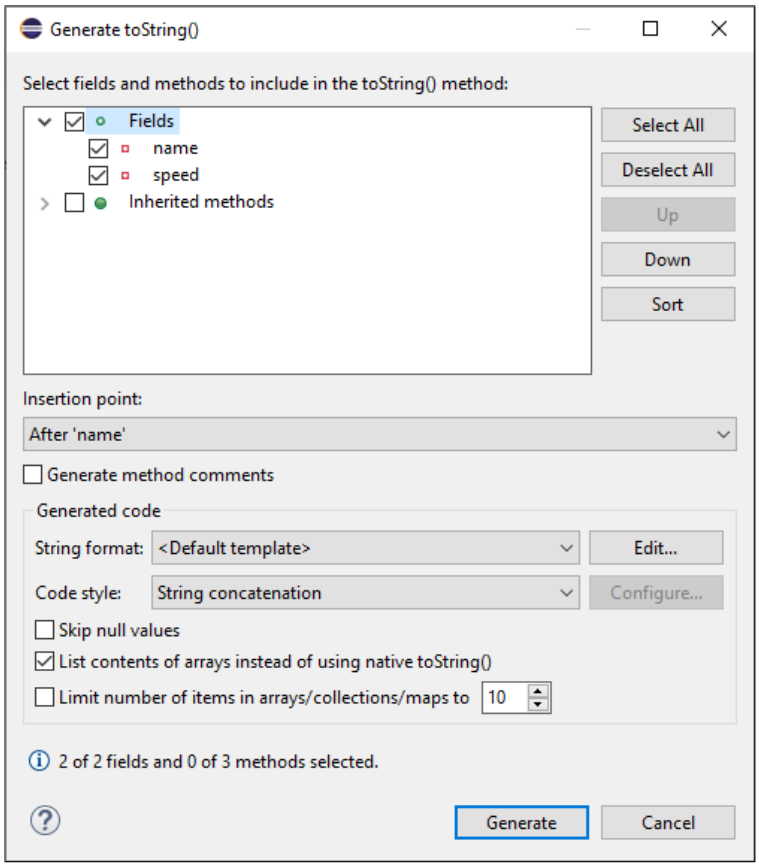
```
public String toString();
```

To generate this method in the class, you can use the IDE:

14. Right-click on the editing window of the `Robot` class.

15. A menu appears. Move the mouse over `Source`.

16. A second menu appears; click on `Generate toString()`. A window appears to offer you the generation of a `toString()` method:



17. The method that will be generated will calculate a string that includes the name of the class followed by the value of the attributes. In this window, it is possible to add other properties to the display. Click on the `Generate` button.

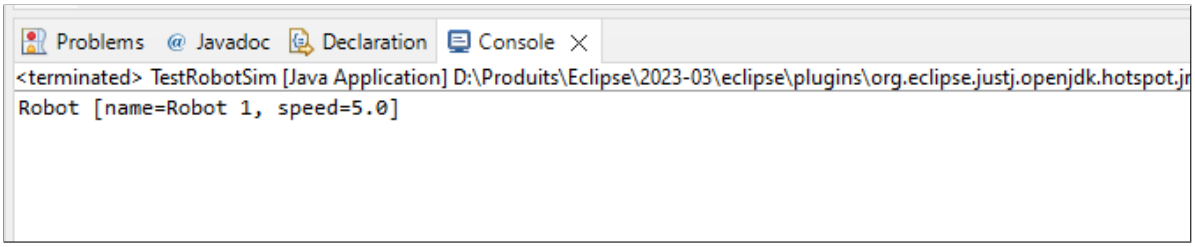
18. The class is modified:

Robot.java

```
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
```

The `@Override` annotation, placed just before the method, indicates that this method is overriding an inherited method. We will cover this in the course later.

19. Run the program again. Is the class display more understandable and useful now?
    You should get this in the `Console` view:



2 Redefining the display of students

1. Execute your program from the previous lab (TP01) to verify that everything still works well.
2. Modify the `toString()` method of the class so that it returns the string formed as follows:

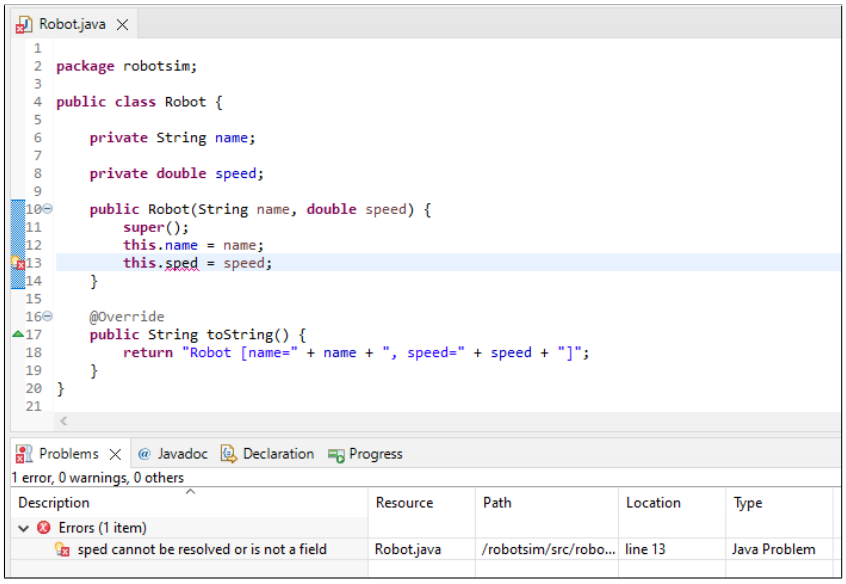
```
My name is Robot 1 and I move at 5.0 km/h.
```

3. Run your program and verify that everything works correctly.

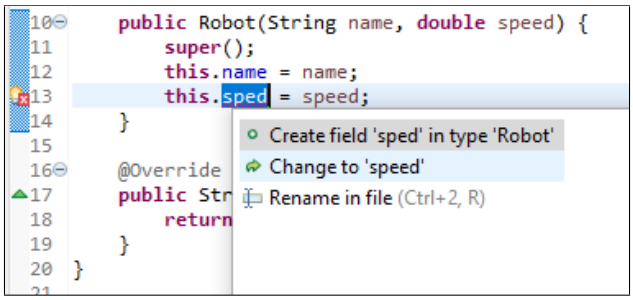
3 Utility of an IDE

In this lab, we ultimately wrote little code by ourselves. This will not always be the case. The development environment allows us to perform standard coding tasks (generation of `constructors`, `getters`, `setters`, `toString()`, etc.) with just a few mouse clicks.

The IDE also notifies you of compilation errors and warnings via the `Problems` view shown in the following screenshot. In this view, double-clicking on an error in the list will take you directly to the position of the error in the Java code editor.



Moreover, in the margin of the code editor, error correction suggestions can be proposed by the IDE by clicking on the light bulb, as illustrated by the screenshot of the following window. In this example, the `speed` attribute was not written correctly.



It is also possible to automatically rename code elements such as class or variable names (refactoring).

All these features are very useful in an industrial development environment as they greatly improve the programmer's productivity. You should not hesitate to use them, although you should always understand the generated code. Indeed, the IDE will not program algorithms for you, although extensions such as Copilot (a paid tool) using artificial intelligence could potentially be useful.