

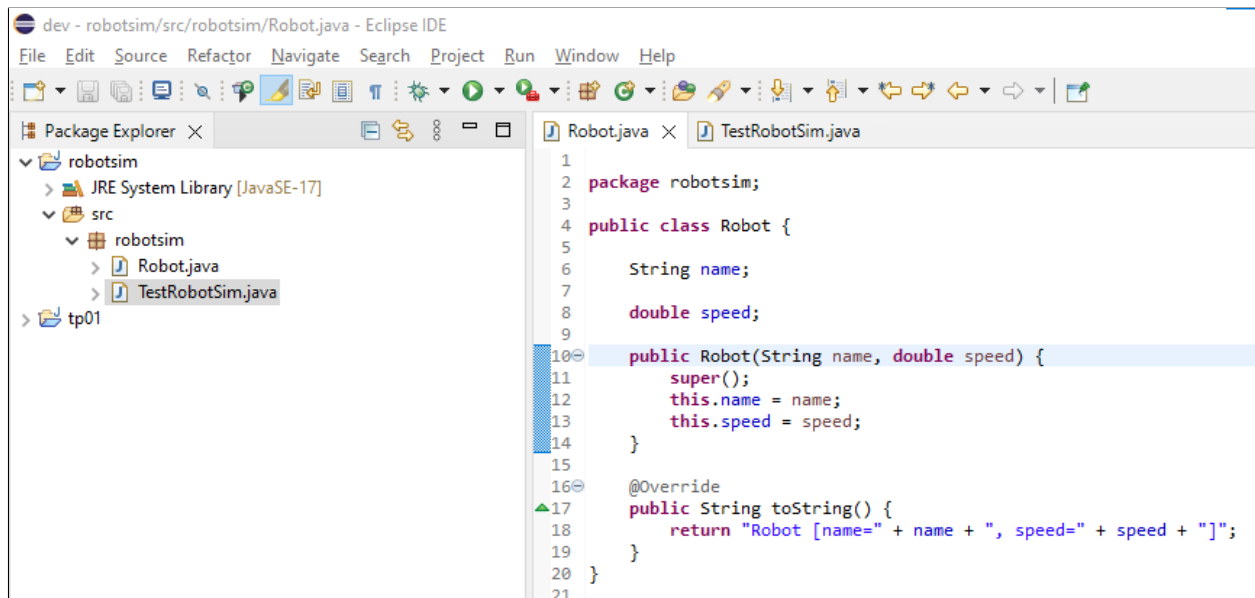
TP03: Modelling a factory containing robots

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

For this lab, you will continue working in the same project as the previous lab, TP02 (named `robotsim`), which will eventually contain **all the sources** for your development project in this course. **Remember :** This project will be submitted at the end of the course.

To model a goods production factory, you will add a new class named `Factory`.

1. But first, launch Eclipse to work on the same `robotsim` project.

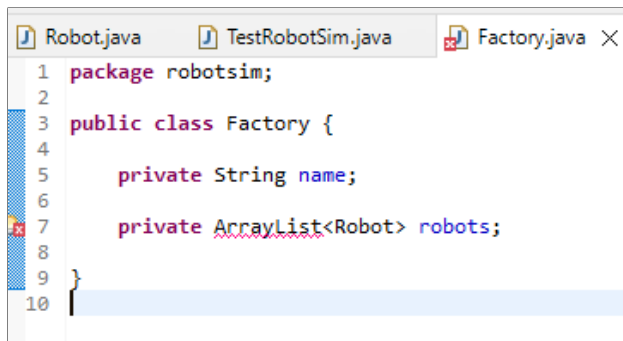


1 Encapsulating the Robot class data

We have seen in class the importance of encapsulating data. This is done using the `private` visibility keyword in Java. Modify the visibility of the attributes in your `Robot` class, then generate accessors for these attributes, considering that a robot's name cannot change during its existence, which is not the case for its `speed`. Just like for the generation of the `toString()` method in the previous lab, the IDE can automatically generate these accessors for you.

2 Creating a **Factory** class

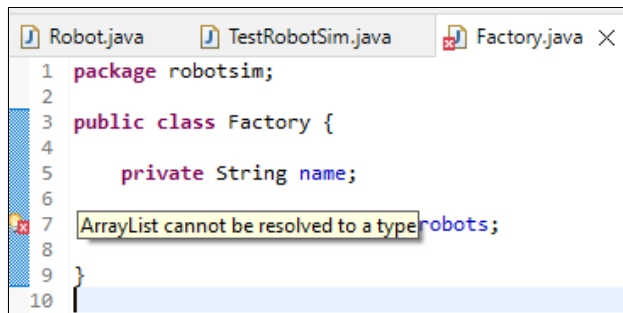
2. By right-clicking on the `robotsim` package and then selecting the `New >> Class` menu, create a class named `Factory`.
3. Since a factory has a name and must contain multiple robots, declare in this class a `name` attribute as for your `Robot` class and an attribute named `robots` of type `ArrayList<Robot>` to contain the factory's robots. Don't forget to encapsulate these attributes using the `private` qualifier. You should get this:



```
1 package robotsim;
2
3 public class Factory {
4
5     private String name;
6
7     private ArrayList<Robot> robots;
8
9 }
10
```

Import declaration

A red mark indicating an error has appeared in the left margin of the code editor. When hovering over this marker with the mouse, the error is displayed:

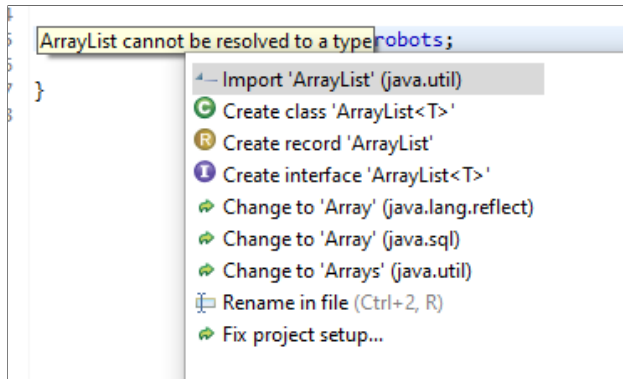


```
1 package robotsim;
2
3 public class Factory {
4
5     private String name;
6
7     ArrayList cannot be resolved to a type robots;
8
9 }
10
```

4. Indeed, the `ArrayList` class is not known by default. As seen in class, you must import this class using the declaration:

```
import java.util.ArrayList;
```

This declaration must be placed at the beginning of the class file. Here again, the IDE can help you. By clicking on the red error mark, the IDE will offer you different possible solutions:



The first solution is of course the correct one. Move the mouse and click on `Import 'ArrayList' (java.util)`. The import declaration is then automatically added to the class.

Writing the constructor

As we have seen in class, by default the robots attribute is initialized with the value `null`. We need to specify a constructor to initialize the attributes of a class with suitable values. Consult the `Javadoc` of the `ArrayList` class to know the constructors of this class.

5. Write a constructor for your `Factory` class.

Adding a robot to the factory

6. Write a method in the `Factory` class that will allow adding robots to the factory. It will have the following signature:

```
public boolean addRobot(String name)
```

This method should first verify that the name of the new robot named `name` is unique among the names of robots that have already been added to the production factory. If the `name` is unique, then the robot will be added to the factory and the method will return the boolean value `true`. Otherwise, the robot will not be added to the factory and the boolean value `false` will be returned. The `speed` of the created robot will be `0.0`.

Verifying the uniqueness of robot names

7. Write a method in the `Factory` class that will verify the uniqueness of a robot name passed as a parameter. This method, which will be called by the `addRobot()` method, will have the signature:

```
private boolean checkRobotName(String name)
```

It should iterate through the list of robots to verify that none of them has the same name as the one passed as a parameter.

Displaying the factory and its robots to the console

8. Write a method in the Factory class with the following signature:

```
public void printToConsole()
```

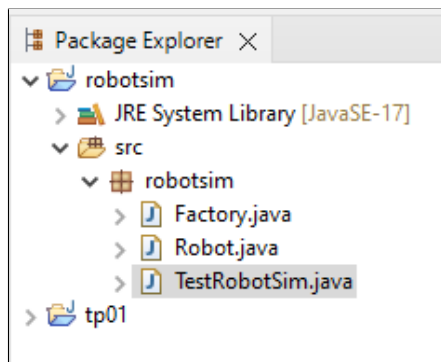
This method will display the name of the factory as well as the list of its robots to the `Console`.

Testing the methods

9. In the `main()` method of the `TestRobotSim` class created in the previous lab, add instructions that will:
- Create an object of the `Factory` class.
 - Add a few robots to this object. Make different trials with some robots with identical names to verify that your program works correctly.
 - Display the `Factory` class object to the `Console`.
10. Execute your program with different sets of robots and verify that what is displayed on the `Console` is correct.

3 Organizing classes into packages

You have probably noticed when you created the `Robot` and `Factory` classes with Eclipse that it put them in a package named `robotsim`, which has the same name as the project.



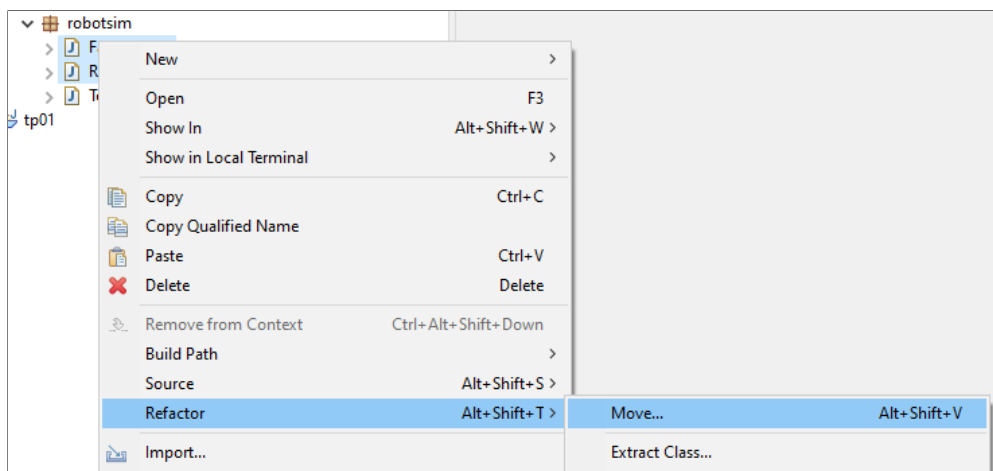
This organization of classes is not ideal because, as seen in class, it is preferable to group classes that perform a common functionality of the application within the same package.

The `Factory` and `Robot` classes perform the function of modelling the production factory. They will therefore be grouped within a package named `fr.tp.inf112.robotsim.model`. What about the `TestRobotSim` class?

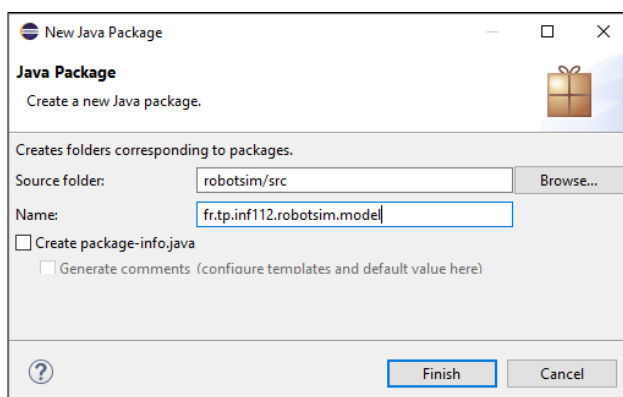
To change the package of a class, it is not enough to rename the package declaration in the class. Indeed, Java requires that the class file be located in a directory tree of the file system equivalent to the class package name, after converting the `"."` characters of the package name to `"/"` characters.

Again, the IDE can automatically change the package declaration of the class and move its file to the correct corresponding subdirectory.

11. To do this, select the class(es) in the package explorer and right-click. In the menu that appears, select `Refactor` >> `Move...`



12. In the dialogue box that appears, enter the desired package name and click `Finish`.



Do not forget to also change the package of the `TestRobotSim` class.

13. Run the `TestRobotSim` class again to verify that your program still works correctly after reorganizing your classes.