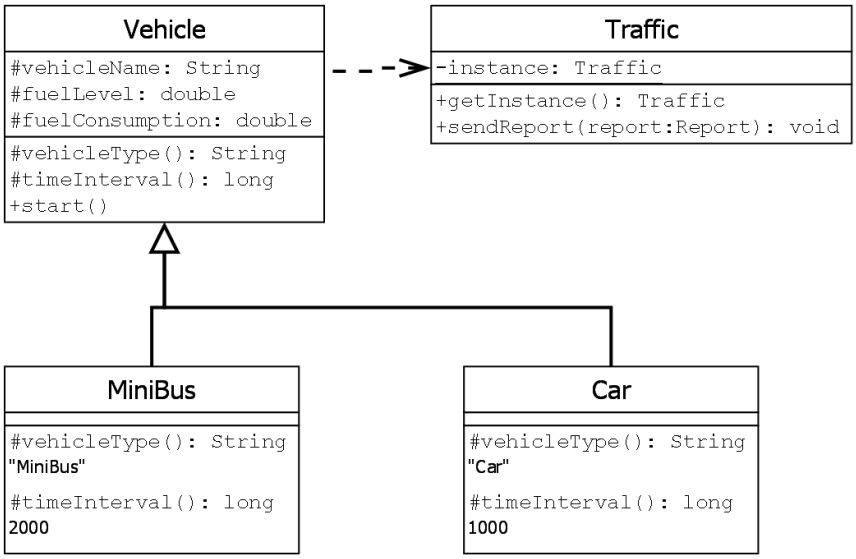
**Exercise 1.**

There are two types of vehicles (car, minibus). Each vehicle has a name, fuel consumption and fuel level. Simulate the vehicles as they stand in a traffic jam. The fuel level of the vehicles decreases corresponding to the fuel consumption. When a vehicle runs out of fuel, it stalls, and finishes its simulation. Assign a thread to each vehicle, which periodically updates the fuel level and writes it out to the console.

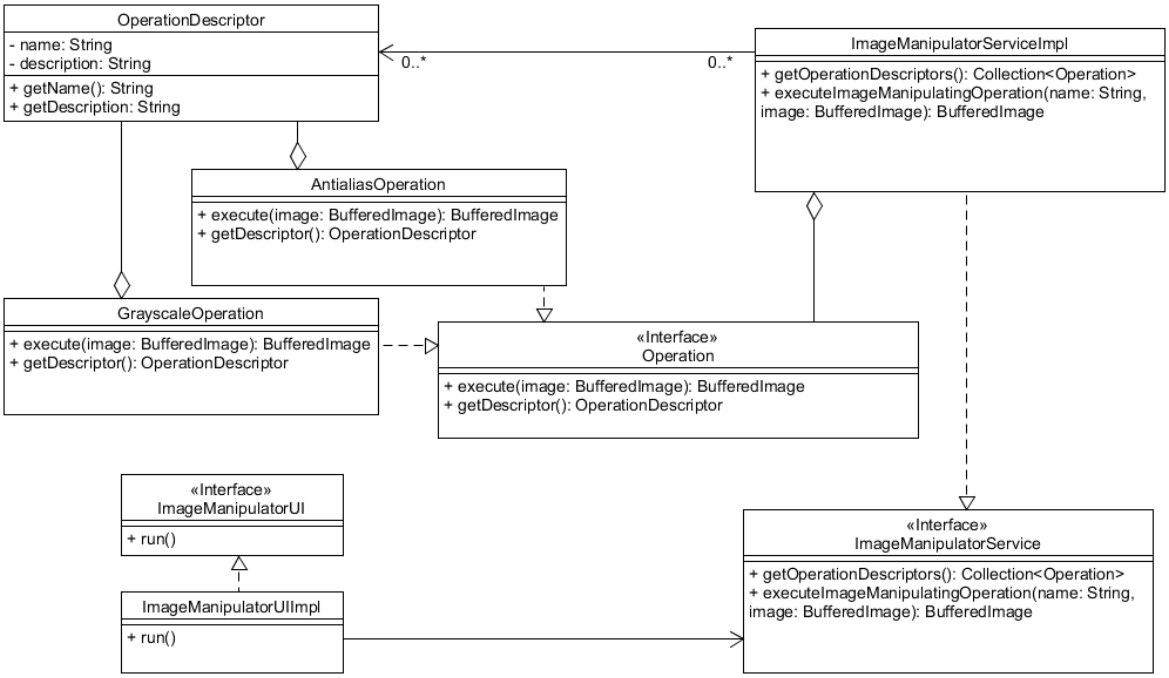
**Class diagram**



**Exercise 2.**

Implement an image manipulation service, where several image manipulations are registered. We can query the descriptors (name and description) of the registered manipulations, and execute a selected one on an image. Create some methods (e.g. blur filter, grayscale filter), and a console user interface, which asks the path of the image, writes out the available manipulations, and then asks the name of the method the user wants to apply, and finally saves the result to a desired place. Implement the application so that the service, manipulation and console interfaces will be put in a separate project, and they also implemented in separate projects.

**Class diagram**



**Exercise 3.**

**Create a class hierarchy, that reflects to the class diagram below. Use ArgoUML to draw the diagram and create the Java code skeleton.**

We have two types of rooms: kitchen, living room
The living room has an owner, but the kitchen doesn't.
Later, we want to add window(s) and door(s) to the rooms.

The diagram below shows the *decorator design pattern*. The idea of the decorator is that it stores a reference to the room, and the additional property, which is used to decorate the room. The decoration now is just a simple message printing to the console (it prints out the decorations: doors, windows). Because the decorator is also a kind of room, we can continue this decoration as long as we want. The only thing we have to take care of is that we have to pass the last decorated room to the new decorator. In order to make the draw() of the decorator work properly, its implementation should call first the draw() method of the decorated room, and then print out the decoration.