Assignment 5 Requirement

Late homework assignments will not be accepted, unless you have a valid written excuse (medical, etc.). You must do this assignment alone. No team work or "talking with your friends" will be accepted. No copying from the Internet. Cheating means zero.

Create a new Eclipse workspace named "**`Assignment5_1234567890`**" on the desktop of your computer (replace **`1234567890`** with your alcohol ID number). For each question below, create a new project in that workspace. Call each project by its question number: "**`Question1`**", "**`Question2`**", etc. Answer all the questions below. At the end of the assignment, create a ZIP archive of the whole workspace folder. The resulting ZIP file must be called "**`Assignment5_1234567890.zip`**" (replace **`1234567890`** with your alcohol ID number). Upload the ZIP file on iSpace.

Here are a few extra instructions:

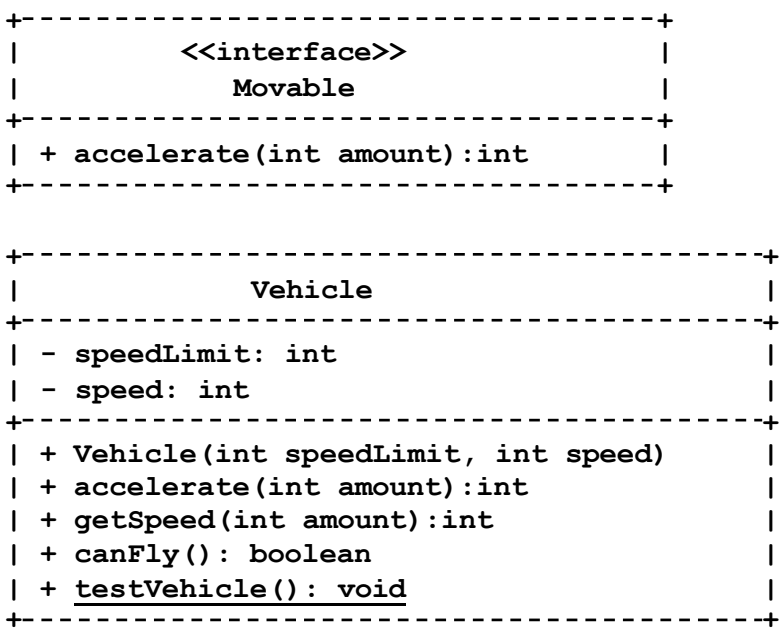
- Do not forget to write tests for *all* the code of *all* the classes.
- Give meaningful names to your variables so we can easily know what each variable is used for in your program.
- Put comments in your code (in English!) to explain WHAT your code is doing and also to explain HOW your program is doing it.
- Make sure all your code is properly indented (formatted). Your code should be beautiful to read.

Failure to follow these instructions will result in you losing points.

Due Date: 23:50 on May 4 (Wednesday)

Question 1

Create a Movable interface and a **`Vehicle`** class with the following UML specifications:



The **`Vehicle`** class implements the **`Movable`** interface, which contains a method of **`accelerate`** that has an amount as argument and will change the speed accordingly. The vehicle speed will plus the given amount and returns the new speed as result.

- If the amount given as argument to the **`accelerate`** method is greater than or equals to 0, the vehicle speed will increase. However, if increase the given amount, the speed will be greater than the **`speedLimit`**, then the accelerate method will not change the speed and must throw an **`ExceedSpeedLimit`** exception with the

message "**Current speed is XXX. Accelerate XXX will exceed the speed limit!**"(with XXX replaced with the speed and amount value respectively).

- If the amount given as argument to the **accelerate** method is less than 0, the vehicle speed will decrease (which is equivalent to decelerate). However, if the result of amount plus current speed is strictly smaller than 0, then the **accelerate** method must throw a **NotEnoughSpeed** exception with the message **"Current speed is XXX, not enough speed to decelerate!"** (with **XXX** replaced with the current speed).

The **Vehicle** method is the constructor which takes two arguments as input and perform initialization of instance variables accordingly.

- If **speedLimit** or **speed** is negative, the constructor must throw a **BadSpeedSetting** exception with the message "**Speed cannot be negative!**",
- If the **speed** is greater than speed limit, the constructor must throw a **BadSpeedSetting** exception with the message "**Speed cannot be greater than speed limit!**"

The **getSpeed** method returns the current speed as the result.

The **canFly** method returns as a result a boolean indicating whether the vehicle can fly or not.

The **testVehicle** method is static and is used for testing the **Vehicle** class.

Modify the **Movable** interface as appropriate and create exception classes as required.

Also add to your program a **Start** class with a **main** method that calls the **testVehicle** method of the **Vehicle** class.

```
public class Start {
	public static void main(String[] args)
		{ Vehicle.testVehicle();
	}
}
```

Question 2

Add to your program a **Car** class. **Car** is a vehicle that cannot fly. Car class has two constructors, one constructor takes the speed limit and current speed as arguments, another one takes current speed as argument and the speed limit is set to 120 by default. Makes sure you correctly test for exceptions in the **testCar** method.

Change the rest of the code as appropriate.

Do not forget to change the **main** method of the **Start** class to run the unit tests of the new **Car** class.

Hint: when you want to compare strings, the **==** operator only works with strings that are constants written directly in your code. You must use the **equals** method to correctly compare strings that are created at runtime using the **+** operator.

Question 3

Add a class **Aircraft** to your program. Aircrafts are vehicles that can fly. The **Aircraft** class must have a private instance variable called **altitude** of type **int** that indicates the altitude at which the aircraft is flying. The constructor for the **Aircraft** class takes as argument a speed limit, a speed and an altitude.

- If the **altitude** is smaller than 0, the constructor must throw a **BadAltitudeSetting** exception with the message "**Altitude cannot be negative!**"

The **Aircraft** class has a public method called **getAltitude**.

The **Aircraft** class has a **testAircraft** method which is static and is used for testing the **Aircraft** class.

Change the rest of the code as appropriate.

Do not forget to create exception classes as required and change the **main** method of the **Start** class to run the unit tests of the new **Aircraft** class.

Question 4

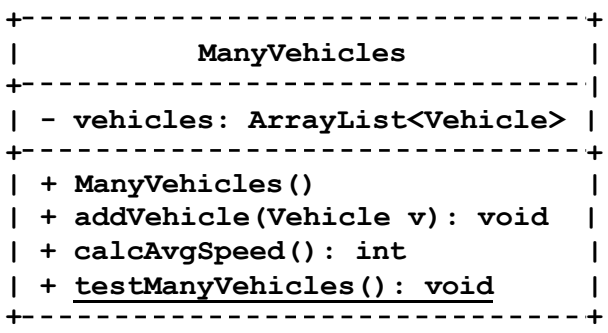
Add a class **Train** to your program. Train is a type of vehicle that cannot fly. The constructor takes a speed limit and a speed as arguments. In our setting, the train will keep the speed unchanged after started and cannot accelerate or decelerate. To prevent any speed changing, trying to accelerate the train must throw a **TrainSpeedChange** exception with the message **"Do not try to accelerate or decelerate the train!"** Makes sure you correctly test for exceptions in the **testTrain** method.

Change the rest of the code as appropriate.

Do not forget to change the **main** method of the **Start** class to run the unit tests of the new **Train** class.

Question 5

We now want to be able to manipulate many vehicles together, not just one courses at a time. So add a **ManyVehicles** class to your program with the following UML diagram:



In the **ManyVehicles** constructor you need to create a new **ArrayList** object and store it in the instance variable **vehicles**, like this: `this.vehicles = new ArrayList<Vehicles>();`

The **addVehicle** method takes any kind of vehicle as argument and adds it to the arraylist.

The **calcAvgSpeed** method calculate the average speed of all vehicles in the arraylist.

Change the rest of the code as appropriate.

Do not forget to change the **main** method of the **Start** class to run the unit tests of the new **ManyVehicles** class.