Lab 3

December 4th, 2017

Schedule

Date	Time	Room
December 4th, 2017	11:30 - 13:00	Q117 and Q120
December 4th, 2017	13:15 - 14:45	Q117 and Q120
December 8th, 2017	9:45 - 11:15	Q117 and Q120

Students assigned to Room Q120 - please, don't forget your notebooks!

Task 3.1 : Sailing a ship

In this exercise, you implement classes for simulating a ship regarding mileage and fuel consumption. The classes `Ship`, `Motor` and `Tank` represent their material equivalents. The UML class diagram given in Figure 1 shows the connections within the ship simulation.

The **ship** consists of `Motor` and `Tank`. The ship is described by the volume of its tank, its speed and the fuel consumption of its motor.

We recommend the following procedure for solving the task:

- Start implementing class `Tank` and add a simple class to test it.
- Then implement class `Motor` (that uses `Tank`) and test it as well.
- Finally, implement class `Ship` (that uses `Motor` and `Tank`) and test it with class `ShipTest`.

Tank

The **tank** is known by the ship and the motor and is described by its total volume and fill level. Use the following constructor for class `tank`:

```
1 public Tank(double totalVolume)
```

The ship queries the tank about its fill level and is able to fill the tank. The motor consumes fuel while moving the ship by using the method `consume()`, which removes the given amount in liters out of the tank. Since the required amount might be greater than the actual fill level, the method returns the amount which was actually removed. To fill a given amount of fuel in the tank, use the method `fill()`. Since the tank might be full, the method returns the actually filled amount.

Write a **test class** for `Tank` to check if your implementation works correctly.

A test class has to implement a `main()` method to be executed by the operating system and checks if your module and methods work as expected. The following could be a **simple example for a test** for class `Tank`. First, fill the empty tank with the full amount of fuel. Since the tank is now full, you expect

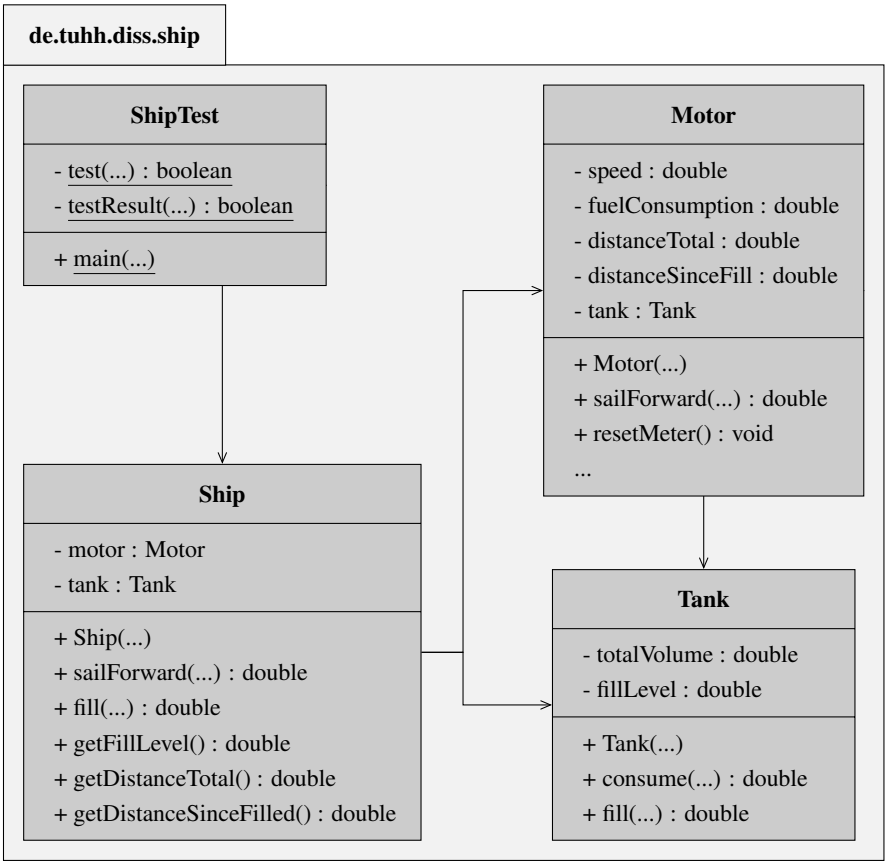

Figure 1: UML class diagram of ship simulation. Methods and constructors are shown without input parameters.

that a user can not fill anymore fuel in the tank. Thus, second, test if the return value of method `fill` is really zero if you query it with an input parameter greater than zero.

Motor

The **motor** is described by its fuel consumption (in liters per hour), the distance since last filling and the total distance (in nautical miles) as well as the speed (in knots = nautical miles/hour). Use the following constructor for class `Motor`:

```
1 public Motor(double fuelConsumption, double speed, Tank tank)
```

The counter for the distance since last filling can be reset by calling method `resetMeter()`. To move the ship, the motor provides the method `sailForward()`. Note that you need the speed of the ship to calculate the correct amount of needed fuel and that the tank might be empty before the specified distance is traveled.

The Ship

The ship can be queried about the actual fill level of its tank (in liters, method `getFillLevel()`), the distance covered by the ship since last filling (in nautical miles, method `getDistanceSinceFilled()`) and the total distance covered (in nautical miles, method `getDistanceTotal()`).

As indicated in Figure 1, the attributes for speed, fuel consumption and tank volume are not members of class `Ship`. Create separate objects of class `Motor` and `Tank` by calling their constructors and store the values where they belong.

To move the ship by a given distance in nautical miles, use the method `sailForward()` which queries the motor of the ship. Since the tank of the motor might run out of fuel while sailing, the method returns the actually traveled distance.

The method `fill()` increases the fill level of the tank by the given amount in liters. Since the tank might be full before the specified amount is filled in the tank, the method returns the actually filled amount.

To ensure compatibility to the `ShipTest` class, implement the following constructor and methods:

```
1 public Ship(double fuelConsumption, double tankVolume, double speed)
2 public double sailForward(double distance)
3 public double fill(double amount)
4 public double getFillLevel()
5 public double getDistanceTotal()
6 public double getDistanceSinceFilled()
```

Use the provided test class `ShipTest` available via StudIP to verify the expected behavior. Think about further tests and add them to the test class.

Add comments for the methods above: explain the behavior of each method briefly and name and explain input parameters as well as return values. Also include the units of return values and parameters in your documentation! You can follow the *Javadoc* syntax if you like, since it allows to simply generate a readable HTML file of your program structure afterwards. For more information, check this simple tutorial, the complete guide or check the comments in the given `ShipTest` class.

Task 3.2 : Turbo Motor (optional)

Please note: this task requires knowledge of inheritance which was not yet treated in the lecture. If you still want to try this task, you can find information in the lecture chapter 4.8 beginning on page 50.

Create a new subclass `TurboMotor` of class `Motor`. As well as the original motor, the turbo motor is described by its fuel consumption, the speed and its tank. Additionally, it offers a *Turbo Mode*, which can be turned on and off. When the *Turbo Mode* is turned on, the speed increases by 50% and the fuel consumption by 100%. If the *Turbo Mode* is turned off, the values of speed and fuel consumption decrease to their original values.

Implement class `TurboMotor` with corresponding constructor and methods for switching the *Turbo Mode* on and off. Keep in mind that you do not need to re-implement the behavior of `Motor` — make use of the inheritance concept!

Change your motor test class so that it is able to test the turbo motor as well.

Add the class `TurboMotor` to the UML diagram in Figure 1.