Classes

Jon Sporring

November 26, 2019

1 Lærervejledningn

Emne Classes, Objects, Methods Attributes

Sværhedsgrad Let

2 Introduktion

3 Opgave(r)

1. Implement a class `student`, which has 1 property `name` and an empty constructor. When objects of the `student` type are created (instantiated), then the individual name of that student must be given as an argument to the default constructor. Make a program, which creates 2 `student` objects and afterward print the name stored in each object using the "`.`"-notation.

2. Change the class in Exercise 1 such that the value given to the default constructor is stored in a mutable field called `name`. Make 2 methods `getValue` and `setValue`. `getValue` must return the present value of an object's mutable field, and `setValue` must take a name as an argument and set the object's mutable field to this new value. Make a program, which creates 2 `student` objects and afterward print the name stored in each object using `getValue`. Use `setValue` to change the value of one of the object's mutable fields, and print the object's new field value using `getValue`.

3. Implement a class `Counter`. The class must have 3 methods:

   - The constructor must make a counter field whose values initially is 0,
   - `get` which returns the present value of the counter field, and
   - `incr` which increases the counter field by 1.

   Write a white-box test that tests the class.

4. Implement a class `Car` with the following properties: A car has

   (a) a specific fuel economy measured in km/liter

(b) a variable amount of fuel in liters in its tank

The fuel economy for a particular `Car` object must be specified as an argument to the constructor, and the initial amount of fuel in the tank should be set to 0.

`Car` objects must have the following methods:

- `addGas`: Add a specific amount of fuel to the car.
- `gasLeft`: Return the present amount of fuel in the car.
- `drive`: Let the car drive a specific length in km, reducing the amount of fuel in the car. If there is too little fuel then cast an exception.

Make a white-box test class `CarTest` to test `Car` and run it.

5. Implement a class `Moth`, which represents a moth that is attracted to light. The moth and the light live in a 2-dimensional coordinate system with axes (x, y), and the light is placed at $(0, 0)$. The moth must have a field for its position in a 2-dimensional coordinate system of floats. Objects of the `Moth` class must have the following methods:

   - The constructor must accept the initial coordinates of the moth.
   - `moveToLight` which moves the moth in a straight line from its position halfway to the position of the light.
   - `getPosition` which returns the moth's initial position.

   Make a white-box test class and test the `Moth` class.

6. Write a class `Car` that has the following data properties:

   - `yearOfModel`: The car's year model.
   - `make`: The make of the car.
   - `speed`: The car's current speed.

   The `Car` class should have a constructor that accepts the car's year model and make as arguments. Set the car's initial speed to 0. The `Car` class should have the following methods:

   - `accelerate`: The `accelerate` method should add 5 to the `speed` attribute each time it is called.
   - `brake`: The `brake` method should subtract 5 from the `speed` attribute each time it is called.
   - `getSpeed`: The `getSpeed` method should return the current speed.

   Design a program that instantiates a `Car` object, and then calls the `accelerate` method five times. After each call to the `accelerate` method, get the current speed of the car and display it. Then call the `brake` method five times. After each call to the `brake` method, get the current speed of the car and display it.

   Extend class `Car` with the attributes ăddGas, gasLeft from exercise 4, and modify methods `accelerate, break` so that the amount of gas left is reduced when the car accelerates or breaks. Call `accelerate, brake` five times, as above, and after each call display both the current speed and the current amount of gas left.

   Test all methods. Create an object instance that you know will not run out of gas, and another object instance that you know will run out of gas and test that your `accelerate, brake` methods work properly.

7. In a not-so-distant future will drones be used for delivery of groceries. Imagine that the drone-traffic has become intense in your area and that you have been asked to decide if drones collide. Assume that all drones fly at the same altitude and that drones fly with different speeds measured in meters/minute and in different directions. If 2 drones are less 5 meters from each other, then they collide. When a drone reaches its destination, then it lands and can no longer collide with any drone. Create an implementation file `simulate.fs`, and add to it a `Drone` class with properties and methods:

   - The constructor must take start-position, -destination, and -speed.
   - `position` (property): returns the drone's position in (x, y) coordinates.
   - `speed` (property): returns the drone's present speed in meters/minute.
   - `destination` (property): returns the drone's present destination in (x, y) coordinates. If the drone is not flying, then its present position and its destination are the same.
   - `fly` (method): Set the drone's new position after 1 minutes flight.
   - `isFinished` (method): Returns true or false depending on whether the drone has reached its destination or not.

   Extend your implementaiton file with a class `Airspace`, which contains the drones and as a minimum has the properties and methods:

   - `drones` (property): The collection of drones.
   - `droneDist` (method): The distance between two given drones.
   - `flyDrones` (method): Advance the position of all flying drones in the collection by 1 minute.
   - `addDrone` (method): Add a new drone to the collection of drones.
   - `willCollide` (method): Given a time interval, determine which drones will collide.

   Write a white-box test class `testSimulate.fsx` that tests both the above classes.