

ASSIGNMENT (AFLEVERINGSOPGAVE) 6

GENERAL INFORMATION

This assignment is made public on Friday, March 12th, 1 PM. The assignment is due on

Friday, March 19th, 1PM.

Hand in your assignment to the teaching assistant running your lab session.

The first page of your (written) assignment has to contain at least the following information:

- the course name (Grundlæggende Programmering)
- name and student number of the fellow student(s) in your group (max two)
- assignment number

Please staple your assignment!

You will get back the graded assignment one week after submission deadline.

WARM-UP AND SUGGESTIONS FOR FURTHER EXERCISES

As a warm-up I suggest the review questions from Chapter 6 on Objects: Data Abstraction (pp. 227/228 of my copy of the textbook).

If you want to do further programming exercises I suggest the following ones from the book (pp. 228–230):

- Exercises 4–6 modifying the `Counter` class used in class.
- Exercise 9 for the mathematically inclined.
- Exercises 13 and 14 about implementing a class representing a clock.
- Exercise 16 which asks you to implement a data type to represent data about a person.
- Exercise 15 (mathematical) implementing a data type to represent the complex numbers.

It is up to you whether or not you want to work on these exercises. Those exercises will not be marked.

ØVELSER

Arbejd i små grupper.

- Klasser: Opgaver 4-6 fra kapitel 6.
- Klasser: Opgaver 16 fra kapitel 6.

ASSIGNMENT: CLASS LIGHT BULB

The questions for this week exercise are from last years IPBR course. In style, the questions represent typical exam questions, the first asking to implement a class with a given specification, the second to use a class where you know the specification, but not the implementation. *Don't be discouraged if you find these questions hard.* It is the first time that you are implementing classes!

1. Class implementation: light bulb

In this question we will implement a simple class that will model a light bulb (*pære*). The UML class diagram of our class will be as follows:

Bulb
-state: boolean ON: boolean OFF: boolean
Bulb() Bulb(boolean)
isOn(): boolean isOff(): boolean getState(): boolean stateToString(): String change(): void

A bulb has two states, either on or off, which we represent by a boolean variable. The value true represents the state on, false represents the state off. There are four methods that question the state of the bulb:

- `boolean isOn()` returns true if the bulb is on, false otherwise.
- `boolean isOff()` returns true if the bulb is off, false otherwise.
- `boolean getState()` simply returns the current state of the bulb.
- `String stateToString()` returns either the strings `on` or `off` depending on the state of the bulb.

Finally, the method `void change()` changes the state of the bulb, from on to off, or from off to on.

There are also two constructor methods:

- `Bulb()` simply sets the state to false, i.e., the bulb is off.
- `Bulb(boolean)` sets the state to the value which is passed as parameter.

Finally, there are also two *static* variables `ON` and `OFF` that represent the two states a bulb can be in.

- (a) Download the files `Bulb.class`, `TestBulb.java`, `BulbGUI.class`, `BulbGUIListener.class` and `TestBulbGUI.class` from the labs homepage. They contain a compiled version of the class `Bulb`, plus (compiled) version of some test programs using class `Bulb`.

Study the code of file `TestBulb.java`. Compile and run the program. Modify the code to get a feel for the use of objects. Observe how we make use of the static variables to set or modify the state of a bulb. Thus, it is not necessary to know that states are represented internally by boolean values. It is not even important to know that the return type of some of the methods is boolean!

Run the class file `TestBulbGUI.class` which contains the compiled version of a simple visualization which represents a bulb and a switch to turn the bulb on or off. Do not worry if there are a number of messages printed on screen.

- (b) Provide your own implementation of class `Bulb` according to the above specification. Compile the file to check for syntax errors.
- (c) Modify the class `TestBulb` so that you test the functionality of your implementations. When you now run the program `TestBulb` it will use your implementation of class `Bulb`.

(d) Why are the two variables `ON` and `OFF` best declared as `static` and as `final`?

HVAD SKAL AFLEVERES: Aflever koden af klassen `Bulb`, koden af din nye klasse `TestBulb`, og en udskrift hvor du viser hvordan dit program kører. Aflever også svaret til det sidste spørgsmål.

2. Class implementation: traffic light

Note: This question can (and should) be answered independently of the previous one.

You are giving the specification of a class `Bulb` modeling a light bulb (*pære*). The UML class diagram is

Bulb
-state: boolean ON: boolean OFF: boolean
Bulb() Bulb(boolean)
isOn(): boolean isOff(): boolean getState(): boolean stateToString(): String change(): void

A bulb has two states, either on or off, which are represented by a boolean variable. The value `true` represents the state on, `false` represents the state off. There are two static variables `ON` and `OFF` for the two different states, and the following methods:

- `Bulb()` simply sets the state to `false`, i.e., the bulb is off.
- `Bulb(boolean)` sets the state to the value which is passed as parameter.
- `boolean isOn()` returns `true` if the bulb is on, `false` otherwise.
- `boolean isOff()` returns `true` if the bulb is off, `false` otherwise.
- `boolean getState()` simply returns the current state of the bulb.
- `String stateToString()` returns either the strings `on` or `off` depending on the state of the bulb.
- The method `void change()` changes the state of the bulb, from on to off, or from off to on.

A compiled version of this class can be downloaded from the course page.

You are to use this class to implement a simple (pedestrian) traffic light which consists of two bulbs, one red one green. When building a traffic light it should be in the state with red bulb on and green one off. There should also be a method which switches to green bulb on, red bulb off, and back to the original state.

The UML diagram and precise specification are as follows:

PTrafficLight
-red: Bulb; -green: Bulb;
PTrafficLight()
change(): void getRed(): Bulb getGreen(): Bulb

The two private attributes are two bulbs, one representing the red, the other the green bulb. The methods are as follows:

- `PTrafficLight()`, the constructor, creates the two bulbs and sets the red one to on and the green one to off.
- `void change()` changes the state of both bulbs.

- `Bulb getRed()` returns a reference to the red bulb.
 - `Bulb getGreen()` returns a reference to the green bulb.
- (a) Download the file `Bulb.class` from the course homepage.
 - (b) Implement the class `PTrafficLight` according to the above specification.
 - (c) Write a test program which will test your pedestrian traffic light. For example, you could create one, print out the state of both bulbs, change the state of the traffic light, and print out again the state of the two bulbs.

HVAD SKAL AFLEVERES: Aflever koden af klassen `PTrafficLight`, koden af dit test program, og en udskrift som viser hvad dit test program laver.