

# Programmering og Problemløsning

## Datalogisk Institut, Københavns Universitet

### Arbejdsseddel 10 - individuel opgave

Jon Sparring

2. december - 6. december.  
Afleveringsfrist: fredag d. 6. december kl. 17:00.

Emnerne for denne arbejdsseddel er:

- Classes
- Objects
- Methods
- Attributes

Opgaverne er delt i øve- og afleveringsopgaver. I denne periode skal I arbejde individuelt med jeres afleveringsopgaver. Regler for gruppe- og individuelle afleveringsopgaver er beskrevet i ”Noter, links, software m.m.” → ”Generel information om opgaver”.

## Øveopgaver

10ø0 Implementer en klasse Counter. Objekter (variable) af typen Counter skal være tællere, og den skal have 3 metoder (funktioner): Konstruktoren, som laver en tæller hvis start værdi er 0; get, som returnerer tællerens nuværende værdi; incr, som øger tællerens værdi med 1. Skriv et unit-test program, som afprøver klassen.

10ø1 Implementér en klasse Car med følgende egenskaber. En bil har en specifik benzin effektivitet (målt i km/liter) og en bestemt mængde benzin i tanken. Benzin effektiviteten for en bil er specificeret med konstruktoren ved oprettelse af et Car objekt. Den indledende mængde benzin er 0 liter. Implementer følgende metoder til Car klassen:

- addGas: Tilføjer en specificeret mængde benzin til bilen.
- gasLeft: Returnerer den nuværende mængde benzin i bilen.
- drive: Bilen køres en specificeret distance, og bruger tilsvarende benzin. Hvis der ikke er nok benzin på tanken til at køre hele distancen kastes en undtagelse.

Lav også en klasse `CarTest` som tester alle metoder i `Car`.

10ø2 Implementér en klasse `Moth` som repræsenterer et møl der flyver i en lige linje fra en bestemt position mod et lys således at møllets nye position er halvvejs mellem dets nuværende position og lysets position. En position er to float tal som angiver x og y koordinater. Møllets indledende position gives ved oprettelse af et `Moth` objekt vha. konstruktoren. Implementér metoderne:

- `moveToLight` som bevæger møllet i retning af et lys med specificeret position som beskrevet ovenfor.
- `getPosition` som returnerer møllets nuværende position.

Test alle metoder i `Moth` klassen.

10ø3 I en ikke-så-fjern fremtid bliver droner massivt brugt til levering af varer købt på nettet. Drone-trafikken er blevet så voldsom i dit område, at du er blevet bedt om at skrive et program som kan afgøre om droner flyver ind i hinanden. Antag at alle droner flyver i samme højde og at 2 droner rammer hinanden hvis der på et givent tidspunkt (kun hele minutter) er mindre end 5 meter imellem dem. Droner flyver med forskellig hastighed (meter/minut) og i forskellige retninger. En drone flyver altid i en lige linje mod sin destination, og når destinationen er nået, lander dronen og kan ikke længere kolliderer med andre droner. Ved oprettelse af et `Drone` objekt specificeres start positionen, destinationen og hastigheden. Implementér klassen `Drone` så den som minimum har attributterne og metoderne:

- `position` (attribut) : Angiver dronens position i (x, y) koordinater.
- `speed` (attribut) : Angiver distancen som dronen flyver for hvert minut.
- `destination` (attribut) : Angiver positionen for dronens destination i (x, y) koordinater.
- `fly` (metode) : Beregner dronens nye position efter et minuts flyvning.
- `isFinished` (metode) : Afgør om dronen har nået sin destination eller ej.

og klassen `Airspace` så den som minimum har attributterne og metoderne:

- `drones` (attribut) : En samling droner i luftrummet.
- `droneDist` (metode) : Beregner afstanden mellem to droner.
- `flyDrones` (metode) : Lader et minut passere og opdaterer dronernes positioner tilsvarende.
- `addDrone` (metode) : Tilføjer en ny drone til luftrummet.
- `willCollide` (metode) : Afgør om der sker en eller flere kollisioner indenfor et specificeret tidsinterval givet i hele minutter.

Test alle metoder i begge klasser. Opret en samling `Drone` objekter som du ved ikke vil medføre kollisioner, samt en anden samling som du ved vil medføre kollisioner og test om din `willCollide` metode virker korrekt.

10ø4 Write a class `Car` that has the following data attributes:

- `yearOfModel` (attribute) : The car's year model.
- `make` (attribute) : The make of the car.
- `speed` (attribute) : 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` (method) : The `accelerate` method should add 5 to the `speed` attribute each time it is called.
- `brake` (method) : The `brake` method should subtract 5 from the `speed` attribute each time it is called.
- `getSpeed` (method) : 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 `addGas`, `gasLeft` from exercise 10ø1, 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.

## Afleveringsopgaver

10i0 Write a class `Car` that has the following data attributes:

- `yearOfModel` (attribute) : The car's year model.
- `make` (attribute) : The make of the car.
- `speed` (attribute) : 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` (method) : The `accelerate` method should add 5 to the `speed` attribute each time it is called.
- `brake` (method) : The `brake` method should subtract 5 from the `speed` attribute each time it is called.
- `getSpeed` (method) : 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 `addGas`, `gasLeft` from exercise 10ø1, 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.

Afleveringen skal bestå af

- en zip-fil
- en pdf-fil

Zip-filen skal indeholde en `src` mappe og filen `README.txt`. Mappen skal indeholde fsharp koden, der skal være en fsharp tekstfil per fsharp-opgave, og de skal navngives `10i0.fsx` osv. De skal kunne oversættes med `fsharpc` og den oversatte fil skal kunne køres med `mono`. Funktioner skal dokumenteres ifølge dokumentationsstandard. Filen `README.txt` skal ganske kort beskrive, hvordan koden oversættes og køres. Pdf-filen skal indeholde jeres rapporten oversat fra  $\text{\LaTeX}$ .

God fornøjelse.