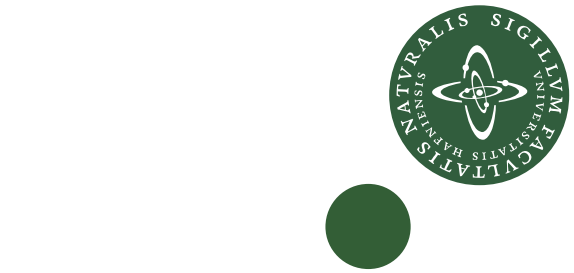
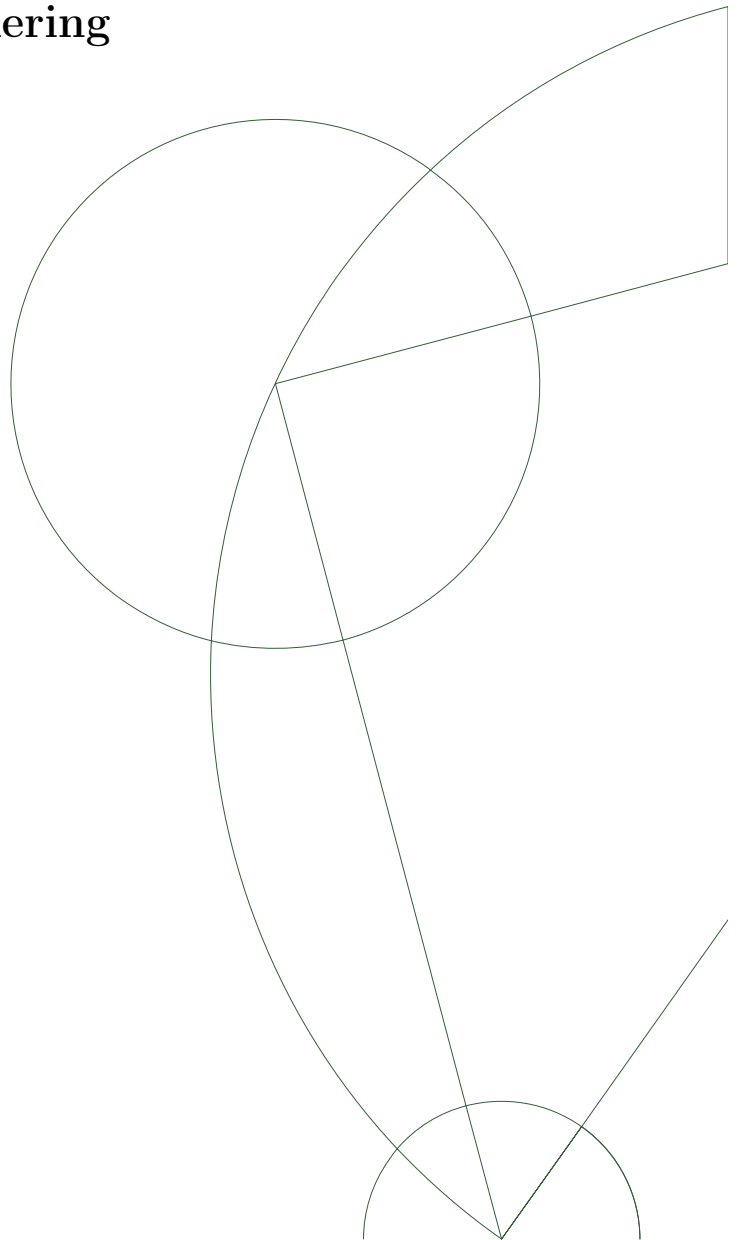


Programmering og Problemløsning

Aflevering 9i - Objekt Orienteret Programmering



Adam Ingwersen

Datalogisk Institut
Københavns Universitet

December 14, 2016

1

Formålet med denne øvelsesopgave er at anvende OOP i FSharp. Hertil bør opgaven tage udgangspunkt i et eksempel, hvor et objekt - en bil - skal have forskellige karakteristika og egenskaber. Bilen betragtes som en klasse i FSharp og kreeres ved brug af `type`, hvis karakteristika defineres a priori for populeringen af et f.eks. bil-objekt. Enhver klasse skal have karakteristika - et såkaldt `member`. Et `member` kan være en variabel, funktion mv., som udelukkende er defineret i konteksten af selve klassen.

1.1 En bil som klasse

Bil-klassen skal have en række data-attributter. Herunder `yearOfModel, make, speed`. Disse attributer kan skrives eksplicit ind i type-definitionens konstruktor. Her vælges det at bruge `yearOfModel` samt `make` i konstruktoren - ydermere tilføjes konstruktorne `efficiency` og `fuelCap`, da disse bliver brugbare senere. Klassen Car har således 4 konstruktorer.
Dertil skal konstruktorerne kunne anvendes udenfor klasse-definitionen og visse af dem skal interagere. Hertil tilføjes konstruktorerne som `members`. `Car` bør ligeledes have en række egeneskaber - også kaldet metoder. Bilen skal kunne accelerere samt bremse.

1.1.1 accelerate

Accelerations-metoden bør forøge bilens hastighed med 5.0, hver gang den kaldes. Metoden bør også reducere gas-beholdningen for bilen, hver gang den kaldes. Ydermere, bør metoden ikke være i stand til at resultere i, at bilen får en negativ gas-beholdning. Derfor introduceres et pattern-match på gas-variablen, således at hvis den næste acceleration ville have resulteret i en negativ gas-beholding; smid exception.

Her er funktionen der dekrementerer gas-variablen forholdsvist arbitrært bestemt. Mængden af gas der skal fratrækkes ved hver acceleration stiger jo højere fart bilen befinder sig i samt det overordnede brændstofsforbrug (`efficiency`).

1.1.2 brake

Bremse-metoden skal formindske bilens hastighed med udgangspunkt i den nuværende `speed`. For hver gang denne metode kaldes, skal hastigheden formindskes med 5.0. Metoden bruger samme arbitrære funktion for reduktion af gas-beholdningen, men eftersom `brake` reducerer hastigheden, er gas-forbruget for hvert kald ligeledes aftagende i `brake`-metoden. Hvis bilens nuværende hastighed er 0.0, er det umiddelbart ikke muligt, at bremse yderligere - derfor promptes brugeren med nedenstående besked.

Et oplagt problem med denne metodes konstruktion er, at den afhænger af, at `speed` altid er et heltal. Hvis det skulle ønskes, at inkrementerne for `accelerate` og `brake` skulle afhænge af f.eks. bilens vægt, ville der hurtigt opstå problemer, da pattern-matche't ikke nødvendigvis ville blive mødt. Der er indført yderligere en undtagelse; da det var et kriterie, at `brake` skulle formindske `gas` ved hvert kald, måtte der indføres en undtagelse, hvis der ikke er mere benzin tilbage til at bremse med.

1.1.3 getSpeed og getGas

Disse returnerer simpelthen bare til den umiddelbare værdi i `speed` hhv. `gas`. Disse anvendes i sektionen nedenfor.

1.2 Instantiering

For at bruge klasser, anvendes begrebet instantiering. Forud for instantiering, defineres en to funktioner, der begge printer; bilens fart samt bilens fart og gasbeholdning. Print-funktionerne returnerer ligeledes bilens `make` og `yearOfModel`.

For at instantiere klassen, defineres en bil med en række karakteristika, der opfylder type-definitionerne i konstruktorerne for `Car`, som f.eks:

```
let aCar = new Car(2016, "VW Up!, 1.0 TDI", 16.9, 35.0)
```

Figure 1: Instantiér klasse

Således haves et af mange mulige objekter; en VW Up! fra 2016, der kører 16.9 Km/l ved bykørsel og har en tank-kapacitet på 35.0 liter. Nu kan metoderne, der blev konstrueret i forrige sektion anvendes.
For simplicitetens skyld anvendes et for-loop til at kalde både accelerate, brake og dertilhørende print-funktion 5 gange.

1.3 Testing

I .fsx-filen er der udført testing fra linje 82-97. Tests er blevet konstrueret, således at `accelerate` og `brake` bliver udsat for grænsetilfælde ifht. gasbeholdningen. Udfordringen har her været, at undgå scenarier, hvor beholdningen antager negativ værdi. De undtagelser, der er anvendt i metoderne for `acclerate` samt `brake` tager højde for netop dette.

1.4 Diskussion og konklusion

Trods mange af koncepterne i OOP er nye - er mange af de syntaktiske elementer i FSharp-sproget bibeholdt, og den hidtidige træning i sproget har givet gode forudsætninger for at gennemføre øvelsesopgaven.
Opgaven kunne på visse områder have været løst mere elegant; eksempelvist kunne man have konstrueret en brugerinteraktion, der tillader brugeren at definere et Car-objekt og bestemme, hvordan dette skulle opføre sig. Ligeldes ifht. testing, ville dette have givet bedre forudsætninger for at afprøve ufordsete scenarier.
Afslutningsvist kan det siges, at de umiddelbare fejl, man kunne begå ved denne opgave er blevet rettet og i det mindste italesat. Det kunne i en mere avanceret udgave være spændende at opbygge en højere grad af kompleksitet for Car-objektet, således at vægt, dækbredede, gearskift og brændstofsforbrug ville blive inkoorporeret iht. at beregne gas-tankens beholdning ved acceleration og deceleration.