Programmering og Problemløsning
Datalogisk Institut, Københavns Universitet
Arbejdsseddel 11 - gruppeopgave

Jon Sporring

9. december - 20. december.
Afleveringsfrist: fredag d. 20. december kl. 17:00.

Emnerne for denne arbejdsseddel er:

- UML diagrams
- Inheritance

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

Øveopgaver

11ø0 Write a `Person` class with data attributes for a person's name, address, and telephone number. Next, write a class named `Customer` that is a subclass of the `Person` class. The `Customer` class should have a data attribute for a customer number and a Boolean data attribute indicating whether the customer wishes to be on a mailing list. Demonstrate an instance of the `Customer` class in a simple program.

11ø1 (a) Write an `Employee` class that keeps data attributes for the following pieces of information:

- Employee name
- Employee number

Next, write a class named `ProductionWorker` that is a subclass of the `Employee` class. The `ProductionWorker` class should keep data attributes for the following information:

- Shift number (an integer, such as 1 or 2)
- Hourly pay rate

The workday is divided into two shifts: day and night. The shift attribute will hold an integer value representing the shift that the employee works. The day shift is shift 1 and the night shift is shift 2. Write the appropriate methods for each class.

Once you have written the classes, write a program that creates an object of the `ProductionWorker` class and prompts the user to enter data for each of the object's data attributes. Store the data in the object and then use the object's methods to retrieve it and display it on the screen.

(b) Extend the previous exercise as follows: Let a shift supervisor be a salaried employee who supervises a shift. In addition to salary, the shift supervisor earns a yearly bonus when his or her shift meets production goals. Write a `ShiftSupervisor` class that is a subclass of the `Employee` class you created in the previous exercise. The `ShiftSupervisor` class should keep a data attribute for the annual salary and a data attribute for the annual production bonus that a shift supervisor has earned. Demonstrate the class by writing a program that uses a `ShiftSupervisor` object.

(c) **(Extra difficult).** Considering that production during night shifts is reduced by 5% compared to production during day shifts, and that the hourly pay rate during night shifts is double the hourly pay rate during day shifts, compute the best possible worker & shift allocation over the period of 12 months. You need to think how to measure productivity and salary cost, and then find their best tradeoff in the period of 12 months.

11ø2 Cheetahs, antelopes and wildebeests are among the world's fastest mammals. This exercise asks you to simulate a race between them. You are not asked to simulate their movement on some plane, but only some of the conditions that affect their speed when running a certain distance.

Your base class is called `Animal` and has these attributes:

- The amount of food needed daily (measured in kilograms)
- The weight of the animal (measured in kilograms)
- The maximum speed of the animal (measured in kilometres per hour)
- The current speed of the animal (measured in kilometres per hour)

The `Animal` class should have a primary constructor that takes two arguments: the animal's weight and the animal's maximum speed. The `Animal` class should also have an additional constructor that takes as input only the animal's maximum speed and generates the animal's weight randomly within the range of 70 - 300 kg. The `Animal` class should have two methods:

- The first method should set the current speed of the animal proportionately to its food intake and maximum speed as follows: if the animal eats 100% of the amount of food it needs daily, the animal's current speed should be its maximum speed; if the animal eats 50% of the amount of food it needs daily, the animal's current speed should be 50% of its maximum speed, and so on.
- The second method should set the amount of food needed daily proportionately to the animal's weight as follows: the animal should eat half its own weight in food every day (if the animal weighs 50 kg, it should eat 25kg of food daily).

Create a subclass `Carnivore` that inherits everything from class `Animal`, and modifies the second method as follows: the animal should eat 8% of its own weight in food every day.

Create a subclass `Herbivore` that inherits everything from class `Animal`, and modifies the second method as follows: the animal should eat 40% of its own weight in food every day.

Create an instance of `Carnivore` called `cheetah` and two instances of `Herbivore` called `antelope, wildebeest`. Set their weight and maximum speed to:

- cheetah: 50kg, 114km/hour

- antelope: 50kg, 95km/hour
- wildebeest: 200kg, 80km/hour

Generate a random percentage between 1 - 100% (inclusive) separately for each instance. This random percentage represents the amount of food the animal eats with respect to the amount of food it needs daily. E.g., if you generate the random percentage 50% for the antelope, this means that the antelope will eat 50% of the amount it should have eaten (as decided by the second method).

For each instance, display the random percentage you generated, how much food each animal consumed, how much food it should have consumed, and how long it took for the animal to cover 10km. Repeat this 3 times (generating different random percentages each time), and declare winner the animal that was fastest on average all three times. If there is a draw, repeat and recompute until there is a clear winner.

Test all methods.

Optional extra: repeat the race without passing as input argument the weight of each animal (i.e. letting the additional constructor generate a different random weight for each instance).

11ø3 Draw the UML diagram for the following programming structure: A `Person` class has data attributes for a person's name, address, and telephone number. A `Customer` has data attribute for a customer number and a Boolean data attribute indicating whether the customer wishes to be on a mailing list.

11ø4 Make an UML diagram for the following structure:

A `Employee` class that keeps data attributes for the following pieces of information:

- Employee name
- Employee number

A subclass `ProductionWorker` that is a subclass of the `Employee` class. The `ProductionWorker` class should keep data attributes for the following information:

- Shift number (an integer, such as 1 or 2)
- Hourly pay rate

A class `Factory` which has one or more instances of `ProductionWorker` objects.

11ø5 Write a UML diagram for the following:

A class called `Animal` and has the following attributes (choose names yourself):

- The amount of food needed daily (measured in kilograms)
- The weight of the animal (measured in kilograms)
- The maximum speed of the animal (measured in kilometres per hour)
- The current speed of the animal (measured in kilometres per hour)

The `Animal` class should have two methods (choose appropriate names):

- The first method should set the current speed of the animal proportionately to its food intake and maximum speed as follows: if the animal eats 100% of the amount of food it needs daily, the animal's current speed should be its maximum speed; if the animal eats 50% of the amount of food it needs daily, the animal's current speed should be 50% of its maximum speed, and so on.
- The second method should set the amount of food needed daily proportionately to the animal's weight as follows: the animal should eat half its own weight in food every day (if the animal weighs 50 kg, it should eat 25kg of food daily).

A subclass `Carnivore` that inherits everything from class `Animal`.

A subclass `Herbivore` that inherits everything from class `Animal`, and modifies the second method as follows: the animal should eat 40% of its own weight in food every day.

A class called `Game` consisting of one or more instances of `Carnivore` and `Herbivore`.

Afleveringsopgaver

I Lake Superior på grænsen mellem USA og Canada ligger en øde ø kaldet Isle Royale. Her har man over en lang årrække fulgt populationen af ulve og elge (http://www.isleroyalewolf.org/). Bestanden af de 2 dyrarter er tæt knyttet til hinanden som rov- og byttedyr.

Denne opgave omhandler simulering af populationen af ulve og elge i et lukket miljø. Elge spiser planter og i denne opgave vil vi antage at ulve kun spiser elge. Både ulve og elge formerer sig, hvilket medfører at populationstørrelserne svinger. Typiske mønstre er, at hvis elgbestanden bliver stor, så vokser ulvebestanden efterfølgende, da der nu kan brødfødes flere ulve. Når ulvebestanden er stor, så falder elgbestanden efterfølgende, da ulvene nedlægger mange elge. Når der er få elge, så falder ulvebestanden pga. manglende føde, hvorefter elgbestanden igen vokser.

11g0 I det følgende skal der simuleres et lukket miljø med ulve og elge. Simuleringen skal benytte følgende regler:

(a) Et miljø består af $n \times n$ felter.

(b) Alle levende dyr har en coordinat i miljøet, og der kan højst være et dyr per felt. Når et dyr dør, fjernes det fra miljøet. Hvis et dyr fødes, tilføjes det i et tomt felt. Ved simuleringens begyndelse skal der være u ulve og e elge som placeret tilfældigt i tomme felter.

(c) Miljøet opdateres i tidsenheder, som kaldes tiks, og simuleringen udføres T tiks. Indenfor et tik kan dyrene gøre et af følgende: Flytte sig, formere sig, og for ulvenes vedkommende spise en elg. Kun et dyr handler ad gangen og rækkefølgen er tilfældig.

(d) Dyr kan flytte sig et felt per tik til et af de 8 nabofelter, som er tomme.

(e) Alle dyr har en artsspecifik formeringstid f angivet i antal tiks, og som tæller ned. Når formeringstiden når nul (for et levende dyr), og der er et tomt nabofelt, så fødes der et nyt dyr af samme type ved at det nye dyr tilføres i et tomt nabofelt. Moderdyrets formeringstid sættes til startværdien, hhv. f_{elg} og f_{ulv}.

(f) Ulve har en sulttid s angivet i antal tiks, og som tæller ned. Hvis sulttiden når nul, så dør ulven, og den fjernes fra miljøet.

(g) Ulve kan spiser elge. Hvis der er en elg i et nabofelt vil ulven spise elgen, elgen fjernes fra miljøet, ulven flytter til elgens felt, og ulvens sulttid sættes til startværdien, s.

(h) I hvert tik reduceres alle formerings- og sulttællere for levende dyr med 1.

Lav et program, som kan simulere dyrene som beskrevet ovenfor og skrive en rapport. Til opgaven udleveres følgende kildefiler:

`animalsSmall.fsi`, `animalsSmall.fs`, og `testAnimalsSmall.fs`.

Opgaven er at tage udgangspunkt i disse filer og programmere følgende regler:

(a) Der skal laves et bibliotek som implementerer klasser for miljø, ulve og elge. Det er ikke et krav at der bruges nedarvning.

(b) Man skal kunne starte simuleringen med forskellige værdier af T, n, u, e, f_{elg}, f_{ulv} og s

(c) Der skal laves en white-box test af biblioteket.

(d) Der skal laves en applikation, som kører en simulering, og tidsserien over antallet af dyr per tik skal gemmes i en fil. Filnavn og parametrene T, n, e, f_{elg}, u, f_{ulv} og s skal angives som argumenter til det oversatte program fra komandolinjen. Eksempelvis kunne:

```
mono experimentWAnimals.exe 40 test.txt 10 30 10 2 10 4
```

starte et eksperiment med $T = 40$, $n = 10$, $e = 30$, $f_{\text{elg}} = 10$, $u = 2$, $f_{\text{ulv}} = 10$ og $s = 4$ og hvor tidsserien skrives til filen `test.txt`.

(e) Der skal laves et antal eksperimenter, hvor simuleringen køres med forskellige værdier af simuleringens parametre. For hvert eksperiment skal der laves en graf (ikke nødvendigvis i F#), der viser antallet af ulve og elge over tid.

(f) Koden skal kommenteres ved brug af F# kommentarstandarden.

Kravene til rapporten er:

(g) Rapporten skal skrives i LaTeX og tage udgangspunkt i `rapport.tex` skabelonen

(h) Rapporten skal som minimum indeholde afsnittene Introduktion, Problemanalyse og design, Programbeskrivelse, Afprøvning, Eksperiment og Konklusion. Som bilag skal I vedlægge afsnittene Brugervejledning og Programtekst.

(i) Eksperimentafsnittet skal kort diskutere hvert eksperiments udfald.

(j) Rapporten minus bilag må maximalt være på 10 A4 sider alt inklusivt.

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 `11g0.fsx` osv. De skal kunne oversættes med fsharpc og den oversattte fil skal kunne køres med mono. Funktioner skal dokumenteres ifølge dokumentationsstandarden. Filen `README.txt` skal ganske kort beskrive, hvordan koden oversættes og køres. Pdf-filen skal indeholde jeres rapporten oversat fra LaTeX.

God fornøjelse.