Programmering og Problemløsning
Datalogisk Institut, Københavns Universitet
Uge(r)seddel 11 - group opgave

Jon Sporring and Christina Lioma

16. december – 20. december.
Afleveringsfrist: onsdag d. 21. december kl. 22:00

I denne periode skal I arbejde i grupper. Formålet er at arbejde med:

- Inheritance
- UML diagrams

Opgaverne for denne uge er delt i øve- og afleveringsopgaver.

Øve-opgaverne er:

10ø.0 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.

10ø.1 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.

10ø.2 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.

10ø.3 **(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.

10ø.4 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.

Produce a UML diagram representing the following.

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).

10ø.5 Du skal implementere en udvidelse til Simple Jacksom indeholder en omstrukturering af nogle af klasserne, samt indførelse af en række nye strategier. Du skal simulere nogle Simple Jackspil hvor du afprøver forskellige strategier, for at afgøre hvilken strategi som lader til at være den bedste.

Implementér super-klassen `Player`, og klasserne `Dealer`, `Human` og `AI` som nedarver fra `Player`. `Player` skal indeholde attributter og metoder som implementerer den fælles funktionalitet som alle tre typer "spillere" har, f.eks. en metode som vælger "Hit" eller "Stand".

Implementér super-klassen `Strategy`, samt en klasse for hver af følgende strategier, som alle nedarver fra `Strategy`

(a) Vælg altid "Hit", medmindre summen af egne kort kan være 15 eller over, ellers vælg "Stand".

(b) Vælg altid "Hit", medmindre summen af egne kort kan være 17 eller over, ellers vælg "Stand".

(c) Vælg altid "Hit", medmindre summen af egne kort kan være 19 eller over, ellers vælg "Stand".

(d) Vælg tilfældigt mellem "Hit" og "Stand". Hvis "Hit" er valgt, trækkes et kort og der vælges igen tilfældigt mellem "Hit" os "Stand" osv.

(e) Følg strategi 2. hvis ét af egne kort er et Es, ellers følg strategi 1.

Simulér 3000 spil Simple Jackmed 5 `AI` spillere som følger de 5 ovenstående strategier. Dealer skal følge strategi 2. Konkludér hvilken af strategierne som lader til at være bedst.

Du skal også

- Opdatere dit UML-diagram
- Lave Unittests
- Kommentere ny kode jævnfør kommentarstandarden for F#

10ø.6 Produce a UML diagram for each of the above exercises.

Afleveringsopgaven er:

10g.0 We consider a simulation of a natural habitat as two groups of animals interact. One group is the prey, a population of animals that are the food source for the other population of animals, the predators. Both groups have a fixed birthrate. The prey usually procreate faster than the predators, allowing for a growing prey population. But as the population of prey increases, the habitat can support a higher number of predators. This leads to an increasing predator population, and, after some time, a decreasing prey population. Around that time, the predator population grows so large as to reach a critical point, where the number of prey can no longer support the present predator population, and the predator population begins to wane. As the predator population declines, the prey population recovers, and the two populations continue this interaction of growth and decay.

An actual example of studying predator-prey relationships is the one between wolves and moose on Isle Royale in Lake Superior (http://www.isleroyalewolf.org/). Its population of wolves and moose are isolated on the island. We can simulate this, with the following rules:

(a) The habitat updates itself in units of time called clock ticks. During one clock tick, every animal in the island gets an opportunity to do something.

(b) All animals are given an opportunity to move into an adjacent space, if an empty adjacent space is found. One move per clock tick is allowed.

(c) Both the predators and prey can reproduce. Each animal is assigned a fixed breed time. If the animal is still alive after breed time ticks of the clock, it will reproduce. The animal does so by finding an unoccupied adjacent space and fills that space with the new animal and its offspring. The animal's breed time is then reset to zero. An animal can breed at most once in a clock tick.

(d) The predators must eat. They have a fixed starve time. If they cannot find a prey to eat before starve time ticks of the clock, they die.

(e) When a predator eats, it moves into an adjacent space that is occupied by prey (its meal). The prey is removed and the predator's starve time is reset to zero. Eating counts as the predator's move during that clock tick.

(f) At the end of every clock tick, each animal's local event clock is updated. All animal's breed times are decremented and all predator's starve times are decremented.

Krav til opgavebesvarelsen

I skal lave et program, som kan simulere rov- og byttedyrene som beskrevet ovenfor og skrive en lille rapport. Afleveringen skal bestå af en pdf indeholdende rapporten, et katalog med et eller flere fsharp programmer som kan oversættes med Monos fsharpc kommando og derefter køres i mono, og en tekstfil der angiver sekvensen af oversættelseskommandoer nødvendigt for at oversætte jeres program(mer). Kataloget skal zippes og uploades som en enkelt fil. Kravene til programmeringsdelen er:

1. Man skal kunne angive antal af tiks (clock ticks), som simuleringen skal køre, formeringstid (breeding time) for begge racer og udsultningstid for rovdyrene ved programstart.
2. Antallet af dyr per tik skal gemmes i en fil.
3. Programmet skal benytte klasser og objekter
4. Der skal være mindst en (fornuftig) nedarvning
5. Programmets klasser skal bla. beskrives ved brug af et UML diagram
6. Programmet skal kommenteres ved brug af fsharp kommentarstandarden
7. Programmet skal unit-testes

Kravene til rapporten er:

8. Rapporten skal skrives i LaTeX.
9. I skal bruge `rapport.tex` skabelonen
10. Rapporten skal som minimum i hoveddelen indeholde afsnittene Introduktion, Problemanalyse og design, Programbeskrivelse, Afprøvning, og Diskussion og Konklusion. Som bilag skal I vedlægge afsnittene Brugervejledning og Programtekst.
11. Rapporten må maximalt være på 10 sider alt inklusivt.

Afleveringsopgaven skal afleveres som et antal fsx tekstfiler navngivet efter opgaven, som f.eks. `10g0a.fsx`. Tekstfilerne skal kunne oversættes med fsharpc, og resultatet skal kunne køres med mono. Funktioner skal dokumenteres ifølge dokumentationsstandarden. Hvis der er mere end 1 fsx-fil, så skal de samles i en zip-fil. Der skal også laves en kort beskrivelse af løsningen for hver opgave i Latex som afleveres ved siden af zip-filen i pdf format. Begge filer uploades til Absalon.

God fornøjelse.