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

Jon Sporring and Christina Lioma

2. januar – 9. januar.
Afleveringsfrist: onsdag d. 9. januar 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.

Øveopgave(r)

11ø.0 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.

11ø.1 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.

11ø.2 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`.

Afleveringsopgave(r)

Sporring, "Learning to program with F#", 2017, Chapter 21.4 describes a simplified version of Chess with only Kings and Rooks, and which we here will call Simple Chess, and which is implemented in 3 files: `chess.fs`, `pieces.fs`, and `chessApp.fsx`. In this assignment you are to work with this implementation.

11g.0 The implementation of `availableMoves` for the King is flawed, since the method will list a square as available, even though it can be hit by an opponents piece at next turn. Correct `availableMoves`, such that threatened squares no longer are part of the list of vacant squares.

11g.1 Extend the implementation with a class `Player` and two derived classes `Human` and `Computer`. The derived classes must have a method `nextMove`, which returns a legal movement as a codestring or the string "quit". A codestring is a string of the name of two squares separated by a space. E.g., if the white king is placed at a4, and a5 is an available move for the king, then a legal codestring for moving the king to a5 is "a4 a5". The player can be either a human or the computer. If the player is human, then the codestring is obtained by a text dialogue with the user. If the player is the computer, then the codestring must be constructed from a random selection of available move of one of its pieces.

11g.2 Extend the implementation with a class `Game`, which includes a method `run`, and which allows two players to play a game. The class must be instantiated with two player objects either human or computer, and `run` must loop through each turn and ask each player object for their next move, until one of the players quits by typing “quit”.

11g.3 Make an extended UML diagram showing the final design including all the extending classes.

Afleveringen skal bestå af en zip-fil og en pdf-fil. Zip-filen skal indeholde en mappe med fsharp koden. Koden skal kunne oversættes med fsharpc og køres med mono. I skal tilføje en README.txt fil, hvor I beskriver, hvordan man oversætter og kører programmet. Pdf-filen skal indeholde jeres LaTeX rapport oversat til pdf.

God fornøjelse.