Programmering og Problemløsning
Datalogisk Institut, Københavns Universitet
Arbejdsseddel 10 - gruppeopgave

Jon Sporring and Christina Lioma

4. december – 13. december.
Afleveringsfrist: onsdag d. 13. december kl. 22:00

I denne periode skal I arbejde individuelt. Regler for individuelle afleveringsopgaver er beskrevet i "'Noter, links, software m.m."'→"'Generel information om opgaver"'. Formålet er at arbejde med:

- Classes
- Objects
- Methods
- Attributes

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

Øve-opgaverne er:

10ø.0 Implementer en klasse `Counter`. Objekter (variable) af typen `Counter` skal være tællere, og den skal have 3 metoder (funktioner): Konstruktoreren, 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.

10ø.1 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`.

10ø.2 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.

10ø.3 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 kollidere 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.

10ø.4 Write a class `Car` that has the following data attributes:

- `yearOfModel` (attribute) : The car's year model.
- `make` (atttribute) : The make of the car.
- `speed` (atttribute) : 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 9ø.0, 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.

Afleveringsopgaven er:

10g.0 Du skal implementere en forsimplet udgave af kortspillet Blackjack som vi kalder Simple Jack. I Simple Jackspiller man ikke om penge/jetoner men blot om sejr/tab mellem en spiller og dealer. Reglerne for Simple Jacker som følger:

Regler Spillet består af en dealer, 1-5 spillere samt et normalt kortspil (uden jokere). Ved spillets start får dealer og hver spiller tildelt 2 tilfældige kort fra bunken som placeres med billedsiden opad foran spilleren, så alle kan se dem. I Simple Jackspilles der med åbne kort dvs. alle trukne kort til hver en tid er synlige for alle spillere. Kortene har værdi som følger:

(a) Billedkort (knægt, dame og konge) har værdien 10

(b) Es kan antage enten værdien 1 eller 11

(c) Resten af kortene har den påtrykte værdi

For hver spiller gælder spillet om at ende med en korthånd hvis sum af værdier er højere en dealers sum af værdier, uden at summen overstiger 21, i hvilket tilfælde spilleren er "bust". Spillerne får nu en tur hver, hvor de skal udføre en af følgende handlinger:

(a) "Stand": Spilleren/dealeren vælger ikke at modtage kort og turen går videre.

(b) "Hit": Spilleren/dealeren vælger at modtage kort fra bunken et ad gangen indtil han/hun vælger at stoppe og turen går videre.

Det er dealers tur til sidst efter alle andre spillere har haft deres tur. Når dealer har haft sin tur afsluttes spillet. Ved spillets afslutning afgøres udfaldet på følgende måde: En spiller vinder hvis ingen af følgende tilfælde gør sig gældende:

(a) Spilleren er "bust"

(b) Summen af spillerens kort-værdier er lavere end, eller lig med dealers sum af kort-værdier

(c) Både spilleren og dealer har SimpleJack (SimpleJack er et Es og et billedkort)

Bemærk at flere spillere altså godt kan vinde på en gang. Et spil Simple Jacker mellem en spiller og dealer, så med 5 spillere ved bordet, er det altså 5 separate spil som spilles.

Implementation I skal designe og implementere et program som kan simulere Simple Jackved brug af klasser. Start med grundigt at overveje hvilke aspekter af spillet som giver mening at opdele i klasser. I skal implementere spillet sådan, at en spiller enten kan være en bruger af Simple Jackprogrammet, som foretager sine valg og ser kortene på bordet via terminalen, eller en spiller kan være en AI som skal følge en af følgende strategier:

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

(b) Vælg tilfældigt mellem "Hit" og "Stand". Hvis "Hit" vælges trækkes et kort og der vælges igen tilfældigt mellem "Hit" og "Stand" osv.

Dealer skal følge strategi nummer 1. Du skal også lave:

- En rapport (maks 20 sider)
- Et UML-diagram af din implementation
- Unit-tests
- Din implementation skal kommenteres jævnfør kommentarstandarden for F#

Hint: Man kan generere tilfældige tal indenfor et interval (f.eks. fra og med 1 til og med 100) ved brug af følgende kode:

```
let gen = System.Random()
let ran_int = gen.Next(1, 101)
```

You should comment your code and describe in max. 2 pages what your program does and how you have tested the methods.

Afleveringsopgaven skal afleveres som et antal fsx tekstfiler navngivet efter opgaven, som f.eks. `10g0.fsx`. Tekstfilerne skal kunne oversættes med fsharpc, og resultatet skal kunne køres med mono. Funktioner skal dokumenteres ifølge dokumentationsstandarden, og udover selve programteksten skal der vedlægges en kort rapport i pdf format kaldet `10g.pdf`, som indholder de dele af besvarelsen, som ikke naturligt vil være i en programtekst. Det hele skal samles i en zip fil og uploades på Absalon.

God fornøjelse.