Programmering og Problemløsning
Datalogisk Institut, Københavns Universitet
Opgave 10 - øvelser og gruppeopgave

Martin Petersen, Jon Sporring og Christina Lioma

Deadline 5. januar

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

- klasser og objekter

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

Øvelsesopgaver

Til øvelserne på alm. skema forventer vi at I arbejder efter nedenstående plan.

Mandag-Tirsdag 7-8/12:
Der arbejdes med 9i, og følgende opgaver:

10.0 Implementér 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.

10.1 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`

Fredag 11/12:
Feedback på opgave 9i og følgende:

10.2 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.

Mandag-tirsdag 14-15/12:

10.3 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.

Fredag 18/12:

Der arbejdes med 10g

Afleveringsopgave

10.4 Du skal implementere en forsimplet udgave af kortspillet Blackjack som vi kalder SimpleJack . I SimpleJack spiller man ikke om penge/jetoner men blot om sejr/tab mellem en spiller og dealer. Reglerne for SimpleJack er som følger:

Regler Spillet består af en dealer, 1-5 spillere samt ét 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 SimpleJack spilles der med åbne kort dvs. alle trukne kort til hver en tid er synlige for alle spillere. Kortene har værdi som følger:

1. Billedkort (knægt, dame og konge) har værdien 10
2. Es kan antage enten værdien 1 eller 11
3. 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 én tur hver, hvor de skal udføre en af følgende handlinger:

1. "Stand": Spilleren/dealeren vælger ikke at modtage kort og turen går videre.
2. "Hit": Spilleren/dealeren vælger at modtage kort fra bunken ét 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:

1. Spilleren er "bust"
2. Summen af spillerens kort-værdier er lavere end, eller lig med dealers sum af kort-værdier
3. Både spilleren og dealer har SimpleJack (SimpleJack er et Es og et billedkort)

Bemærk at flere spillere altså godt kan vinde på én gang. Et spil SimpleJack er mellem én 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 SimpleJack ved 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 SimpleJack programmet, 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:

1. Vælg altid "Hit", medmindre summen af egne kort kan være 17 eller over, ellers vælg "Stand"
2. 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)
```