Opgaver uge 7

Formål med opgaverne

Efter disse øvelser skal du kunne kunne benytte begreber om programdesign, især arv, subklasser og subtyper. Forkortelsen "B&K" henviser til lærebogen Barnes og Kölling: *Objects First with Java*, tredje udgave.

Hvis du vil have feedback på dine besvarelser, så udskriv dem på papir og aflevér dem senest ved tirsdagsøvelserne kl 13.00. Så retter og kommenterer hjælpelæreren dem og giver dig dem tilbage ugen efter.

Afleveringsopgave GC

Opgaverne 7.2 og 7.3 afleveres som besvarelse til afleveringsopgave GC. I må naturligvis også gerne aflevere 7.1 hvis i vil have yderligere feedback fra TA's. På grund af efterårsferie skal GC afleveres tirsdag efter efterårsferien kl 13.

Opgave 7.1

Denne opgave handler om at beskrive forskellige typer boliger: hus, lejlighed og telt, hvor hus og lejlighed har det fællestræk at de har en fast adresse, og alle tre har det fællestræk at de har et areal. Det kan modelleres med en superklasse `FastEjendom` for `Hus` og `Lejlighed`, og en superklasse `Bolig` for `FastEjendom` og `Telt`.

(i) Start med at tegne (i hånden) klassehierarkiet for klasserne `Bolig`, `FastEjendom`, `Telt`, `Hus` og `Lejlighed`.

(ii) Lav klassen `Bolig` med et enkelt felt `areal` af type heltal, en konstruktor til at initialisere feltet, og en metode `public void show()` der udskriver arealet.

(ii) Lav klassen `FastEjendom` som en subklasse af `Bolig` med yderligere et felt `adresse` af type `String`, en konstruktor, og en metode `public void show()`.

Vink: Konstruktoren skal tage arealet og adressen som argument, kalde superklasse-konstruktoren `super(areal)` for at initialisere `areal`-feltet som vist side 260–261 i B&K, og skal derefter initialisere `adresse`-feltet.

Vink: Metoden `show()` skal udskrive adressen og skal derefter udføre `super.show()` for at kalde superklassens udgave af `show()`, der jo udskriver arealet. Metodekaldet `super.show()` udfører superklassens `show()`-metode; se B&K afsnit 9.5. På den måde tager superklassen `Bolig` sig af alt der har med `areal` at gøre, og subklassen `FastEjendom` tager sig af alt der har med `adresse` at gøre.

(iii) Lav på samme måde klassen `Telt` som en subklasse af `Bolig` med yderligere et felt `vandtæt` af type `boolean` der viser om teltet holder regnen ude, en konstruktor, og en metode `public void show()`. Metoden `show()` skal udskrive teksten "`Telt`" og værdien af `vandtæt` og derefter kalde `super.show()`.

(iv) Lav klassen `Hus` som en subklasse af `FastEjendom` med yderligere et felt `grundareal` af type heltal der viser hvor stor husets grund er, en konstruktor, og en metode `public void show()`. Metoden `show()` skal udskrive teksten "`Hus`" og grundarealet og derefter kalde `super.show()`.

(v) Lav på samme måde klassen `Lejlighed` som en subklasse af `FastEjendom` med yderligere et felt `etage` af type heltal der viser hvilken etage den ligger på, en konstruktor, og en metode `public void show()`. Metoden `show()` skal udskrive teksten "`Lejlighed`" og etagen og derefter kalde `super.show()`.

(vi) Lav en klasse `Boliger` med en metode `public void initAndShow()` der opretter en arrayliste med `Bolig`-objekter og bruger en foreach-løkke til at skrive dem ud. Resultatet kunne se sådan ud:

```
Hus, grundareal = 845
Adresse = Kirkevej 29
Areal = 210 kvm

Lejlighed, etage = 4
```

```
Adresse = Falkoner alle 110
Areal = 140 kvm

Telt, vandtæt = false
Areal = 4 kvm
```

Opgave 7.2

Denne opgave handler om simulering af træer, i fortsættelse af opgave GA.1 fra uge 2 og opgave 4.2 fra uge 4. Men nu skal der ud over klassen `Tree` laves to subklasser `Ash` og `Beech`, svarende til asketræer og bøgetræer. Ideen er at asketræer og bøgetræer har forskellige vækstforløb, som bedst beskrives med forskellige `growOneYear()` metoder.

(i) Lav en klasse `Tree` med to `public` felter:

- `age` af type `int` som er træets alder.
- `height` af type `double` som er træets højde.

(ii) Lav en metode `void growOneYear()` som forøger `age` med 1 og ellers ikke gør noget.

(iii) Lav en metode `void show()` som udskriver træets alder og højde, fx i dette format: `alder = 17 år og højde = 3.66 meter`.

(iv) Lav en klasse `Ash` som en subklasse af `Tree`. Klassen skal ikke erklære nogen felter selv, den arver jo `age` og `height` fra superklassen `Tree`. Klassen skal have en konstruktor `Ash()` uden parametre som sætter `age` til 1 og `height` til 1.0.

(v) Klasse `Ash` skal have en ny version af metoden `public void growOneYear()`. Dels skal denne metode udføre `super.growOneYear()`, som kalder superklassens `growOneYear`-metode der tæller `age` op med 1. Dels skal den lægge 20 % til højden ved at gange `height` med 1.20, forudsat at *højden* er mindre end 15.

(vi) Klasse `Ash` skal desuden have en ny version af metoden `public void show()`, som skriver "`Ask:`" og dernæst udfører `super.show()` for at kalde superklassens `show`-metode.

(vii) Lav en klasse `Beech` som en subklasse af `Tree`. Klassen skal have en konstruktor `Beech()` uden parametre som sætter `age` til 1 og `height` til 0.5.

Klasse `Beech` skal også have en ny version af metoden `public void growOneYear()`. Dels skal denne metode kalde `super.growOneYear()`, dels skal den lægge 6 % til højden, forudsat at *alderen* er mindre end 65. (Så i denne model er bøgetræers vækst begrænset af deres alder, mens asketræers vækst er begrænset af deres højde. I virkeligheden er det mere indviklet.)

Klasse `Beech` skal desuden have en ny version af metoden `public void show()`, som skriver "`Bøg:`" og dernæst kalder superklassens `show`-metode.

(viii) Lav en klasse `Forest` med en arrayliste af `Tree`-objekter. Lav en metode `initialize()` der kommer to asketræer og to bøgetræer af forskellig alder og højde i arraylisten.

(ix) Lav en metode `public void growOneYear()` i klasse `Forest` der gennemløber arraylisten og kalder `growOneYear` på hvert træ. Lav en metode `public void show()` i klasse `Forest` der gennemløber arraylisten og kalder `show()` på hvert træ.

(x) Lav en metode `public void growManyYears(int n)` der simulerer at skoven vokser i `n` år, dvs. `n` gange kalder `growOneYear()` og `show()`.

Opgave 7.3

I opgaven ovenfor indeholder en `Forest` en arrayliste af `Tree`-objekter. Skoven "ved" altså ikke noget om hvordan træerne står i forhold til hinanden.

For at få en mere realistisk simulering af et træs vækst er man nødt til at vide om træet står i nærheden af andre (høje) træer der skygger; så vil træet jo vokse langsommere. For at lave sådan en simulering kan man repræsentere skoven som et to-dimensionalt array `Tree[][] trees` af pladser, hvor hver plads `trees[i][j]` kan indeholde et `Tree` eller ingenting (`null`). Træet på plads (i, j) har så op til fire nabotræer, på pladserne $(i-1, j)$ og $(i+1, j)$ og $(i, j-1)$ og $(i, j+1)$.

(i) Opret et nyt BlueJ-projekt til denne opgave. Du kan genbruge næsten alt fra dine `Tree`, `Ash` og `Birch`-klasser, men klassen `Forest` udskiftes med `NewForest`.

(ii) Lav en klasse `NewForest` som har et felt `trees` af type `Tree[][]` til at indeholde alle skovens træer, og et felt `n` der angiver hvor stor skoven er på hver led. Klassen skal have en konstruktor `NewForest()` som sætter `n` til 4 og opretter arrayet `trees` med størrelse 4×4. Konstruktoren skal også oprette mindst 8 aske- og birketræer og sætte dem på forskellige pladser i "skoven".

(iii) Lav en metode `Tree getTree(int i, int j)` i `NewForest` der returnerer træet på plads (i, j) hvis i og j er inden for skoven, og ellers returnerer `null`.

(iv) Lav en metode `double getHeight()` i `Tree` der returnerer træets højde.

(v) Lav en metode `double shadow(int i, int j, NewForest forest)` i klasse `Tree` som beregner hvor meget skygge træet på plads (i, j) i skoven `forest` er udsat for, som et tal mellem 0.0 og 1.0 hvor 0.0 betyder ingen skygge og 1.0 betyder komplet skygge. Du kan beregne skyggen som `k/10.0` hvor `k` er antallet af nabotræer der er højere end det givne træ. (Vi dividerer med 10.0 i stedet for 4.0 for selv hvis alle fire nabotræer er højere, er der jo ikke komplet skygge). Vink: Metoden skal benytte kald af formen `forest.getTree(...,...)` for at finde træets naboer.

Hvis træet står på plads (i, j) så har det op til fire nabotræer der står nord for træet ("over"), eller syd for ("under"), eller vest for ("til venstre"), eller øst for ("til højre"). De fire muligheder svarer til pladserne $(i-1, j)$ og $(i+1, j)$ og $(i, j-1)$ og $(i, j+1)$. Men læg mærke til at dels kan det være at der ikke er noget træ (men `null`) ved siden af, dels står nogle af træerne i randen af skoven og har derfor ikke fire naboer. Hvis for eksempel $i = 0$ så står træet i øverste række og så er der ikke nogen nabo over træet. Metoden skal altså tjekke for `null`.

(vi) Lav en metode `void growOneYear(int i, int j, NewForest forest)` i klasse `Tree` som modificerer et træs højdetilvækst ud fra hvor meget skygge det får. Parametrene `i` og `j` angiver træets position, og `forest` er skoven det står i. Metoden kan for eksempel aflæse træets aktuelle højde, kalde træets eksisterende `growOneYear`-metode, udregne højdetilvæksten, udregne skyggefaktoren, og så reducere højdetilvæksten ved at trække skyggefaktor gange højdetilvækst fra højden.

(vii) Lav en metode `void growOneYear()` i `NewForest` der kalder `tree.growOneYear(i,j,this)` for alle træer `tree=trees[i][j]` i skoven – men selvfølgelig kun når der er et træ på plads (i, j), ikke når der står `null`.