Datalogisk Institut
Aarhus Universitet

dIntProg-E04

Introduktion til Programmering

Opgaver

E2004

August 2004

Dette er en samling af opgaver til kurset Introduktion til Programmering.

I nogle opgaver henvises til filer med JAVA-programmer. Dette eksempelmateriale kan enten findes på nettet:

`http://www.daimi.au.dk/dIntProg/eksempler/`

eller direkte i alle studerendes katalog `~/eksempler/`

Indhold

1 Introduktion

1.1 Programmeringsomgivelser

Opgave 1.1.1

a) Installer JAVA SDK og BlueJ på din computer.

b) Kig på JAVA-dokumentationen og forsøg at få et overblik/indtryk af strukturen og indholdet af (den ret overvældende) dokumentation.

c) Læs "The BlueJ Tutorial" og løs opgaverne deri (du kan finde den på `http://www.daimi.au.dk/dIntProg/bluej/tutorial.pdf`).

1.2 Grafik

Opgave 1.2.1

Den opgave går ud på at arbejde med objekter i form af farveblyanter.

JAVA stiller et omfattende grafikbibliotek til rådighed. Det vil senere i kurset blive brugt direkte. Her vil vi i stedet nøjes med at anvende en simpel klasse `Crayon` der beskriver tilstand og metoder for en farveblyant. Hvert objekt der laves ud fra klassen bliver en selvstændig blyant af en angivet farve og stiftbredde. Alle blyanterne tegner på det samme ark papir. Papiret har et koordinatsystem svarende til den første kvadrant i et sædvanligt koordinatsystem, med en lille ændring: y-aksen er vendt på hovedet og peger nedad, så $(0, 0)$ er i øverste venstre hjørne (dette kan synes underligt, men er blevet en standard i grafikverdenen).

Hvert farveblyant-objekt har en tilstand, der består af:

- en farve
- en stiftbredde (fra 1 og opefter)
- en aktuel position (et koordinatpunkt)
- en aktuel tegneretning (vinkel i grader)

Man kan kommunikere med hvert af disse objekter gennem følgende metoder:

- `move(d)`: tegn en linje der er d enheder lang i den aktuelle tegneretning ud fra den aktuelle position.
- `moveto(x,y)`: tegn en linje fra den aktuelle position hen til punktet med koordinater (x, y).

- `jump(d)`: flyt blyanten d enheder i den aktuelle tegneretning ud fra den aktuelle position (uden at tegne).
- `jumpto(x,y)`: flyt blyanten fra den aktuelle position hen til punktet med koordinater (x, y) (uden at tegne).
- `turn(a)`: læg a grader til den aktuelle tegneretning.
- `turnto(a)`: den aktuelle tegneretning ændres til at være a grader.

Man laver en ny farveblyant af farve c og tegnebredde w med konstruktørkaldet `new Crayon(c,w)`.

Følgende JAVA-program tegner et rødt kvadrat inden i et blåt kvadrat:

```
public class CrayonSquares
{
    public static void main(String[] args)
    {
        Crayon blue_pencil = new Crayon(Color.blue,1);
        Crayon red_pencil = new Crayon(Color.red,1);

        blue_pencil.jumpto(100,100);
        blue_pencil.turnto(0);
        blue_pencil.move(100);
        blue_pencil.turn(90);
        blue_pencil.move(100);
        blue_pencil.turn(90);
        blue_pencil.move(100);
        blue_pencil.turn(90);
        blue_pencil.move(100);
        blue_pencil.turn(90);

        red_pencil.jumpto(120,120);
        red_pencil.turnto(0);
        red_pencil.move(60);
        red_pencil.turn(90);
        red_pencil.move(60);
        red_pencil.turn(90);
        red_pencil.move(60);
        red_pencil.turn(90);
        red_pencil.move(60);
        red_pencil.turn(90);
  }
```

```
}
```

a) Prøv at oversætte og udføre ovenstående program. For at gøre dette må du først tage en kopi af projektet `opg1.2.1/crayon/`

b) Find ud af hvordan du redigerer i programmet, og lad det tegne en ligesidet trekant inden i en ligesidet sekskant i stedet for at tegne to kvadrater.

c) `main`-metoden i klassen `CrayonTree` bruger farveblyanter til at tegne en buket (prøv det). Lav et nyt program, der laver en kunstnerisk tegning efter dit eget valg.

Opgave 1.2.2

a) Lav et grafikprogram, der tegner en ottekant.

b) Lav et grafikprogram, der tegner et (simpelt) hus.

c) Lav et grafikprogram, der tegner en høj bygning (med mange etager).

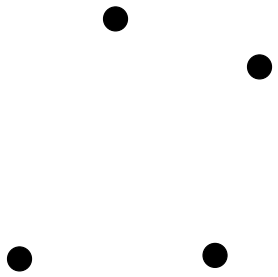
d) Lav et grafikprogram, der flytter en ottekant frem og tilbage over skærmen.

e) Lav et grafikprogram, der tegner en "slange", der bevæger sig hen over skærmen.

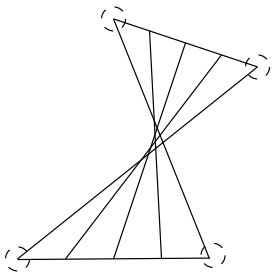
Opgave 1.2.3

Denne opgave arbejder videre med farveblyanterne fra opgave 1.2.1.

Man kan lade de fire hjørner i en firkant udspænde et "neg" som her:



4 hjørner i firkant



"neg"

a) Skriv et program der som input tager en firkant (f.eks. i form af koordinaterne for hjørnerne), og tegner et modsvarende "neg". Kør programmet på nogle eksempler.

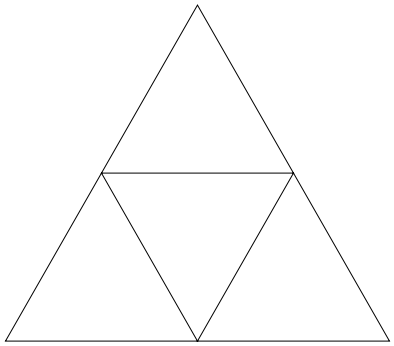
Opgave 1.2.4
Denne opgave arbejder videre med farveblyanterne fra opgave 1.2.1, og går ud på at konstruere objekter, der har metoder til at tegne trekanter og grupper af trekanter.

Skriv en klasse `Triangles` med:

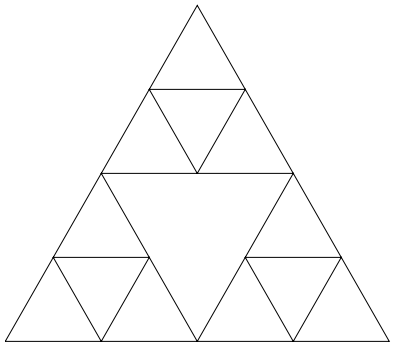
- en konstruktør, der som parameter tager en farveblyant (type `Crayon`), og gemmer en reference hertil i en tilstandsvariabel (field) `c`
- en metode `triangle`, der som parameter tager et heltal (type `int`) og bruger farveblyanten til at tegne en ligesidet trekant med sidelængde angivet af parameteren.
- en metode `tritriangle` der er defineret således:

```
/**
 * tritriangle draws a tritriangle with the left base point at
 * the current position of the crayon, and with the base line
 * in the current direction of the crayon. When the drawing is
 * finished the crayon is left in the same current position
 * and direction as it had initially
 * @param l Side length of tritriangle
 */
public void tritriangle(int l)
{
    triangle(l/2);
    c.move(l/2);
    triangle(l/2);
    c.turn(-120); c.move(l/2); c.turn(120);
    triangle(l/2);
    c.turn(120); c.move(l/2); c.turn(-120);
}
```

Afprøv klassen, idet det forventes at et kald af `tritriangle` resulterer i:



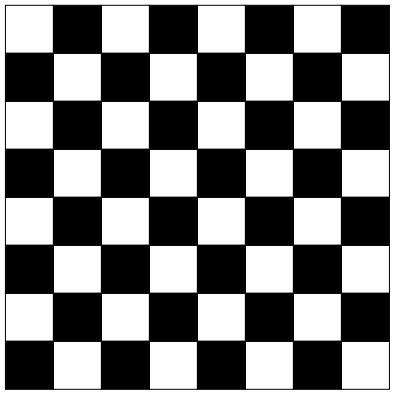
Tilsvarende ser en tritritriangle således ud:



Tilføj en metode, der kan tegne en *tritritriangle* efter samme princip, d.v.s. en tri^{n}angle fremkommer ved at placere en tri^{n-1}angle ovenpå to tri^{n-1}angles. Afprøv den resulterende klasse.

Hvor mange små trekanter er der inden i en *tritritriangle*? Hvor mange små trekanter er der inde i en tri^{n}angle ? (udtryk tallet som en funktion $T(n)$ af n)

Opgave 1.2.5
Denne opgave går ud på at tegne et skakbræt med JAVA's grafikpakke:



Appletten på næste side er næsten færdig. Den mangler blot indmaden til metoden `draw8x8Board`. Observer at et 8×8 bræt kan opfattes som sammensat af fire 4×4 brætter.

a) Skriv signaturen for en metode `draw4x4Board`, der tænkes at kunne tegne et 4×4 bræt (analogt med den givne signatur for `draw8x8Board`). Udfyld derpå kroppen til `draw8x8Board`)

b) Overvej, hvordan du analogt kan skrive `draw4x4Board`. Færdiggør og afprøv programmet.

c) Overvej alternativ strukturering af programmet. Kan du f.eks basere tegningen af et skakbræt på en klasse `Board8x8`, der indeholder 4 fields af type `Board4x4`?

```
import java.applet.Applet;
import java.awt.*;

public class ChessBoard extends Applet
{
    public void paint(Graphics g)
    {
        draw8x8Board(50,50,25,g);
    }

    /**
     * draw8x8Board
     * @param x First coordinate of upper left corner of board
     * @param y Second coordinate of upper left corner of board
     * @param f Side length of a single field on the board
     * @param g Graphics pen to be used for drawing
     */
    private void draw8x8Board(int x, int y, int f, Graphics g)
    { ... }
}
```

Opgave 1.2.6
Udarbejd passende JAVA-dokumentation for klassen `Crayon` fra opgave 1.2.1.

1.3 Diverse

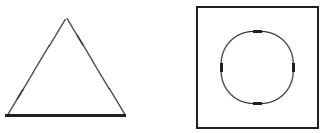
Opgave 1.3.1
En *figurtegning* kan beskrives på følgende måde:

- **trekant**, **firkant** og **cirkel** beskriver tegninger af de tilsvarende simple figurer.
- T_1 **til venstre for** T_2 beskriver en tegning af T_1 ved siden af T_2, centreret vandret.
- T_1 **oven på** T_2 beskriver en tegning af T_1 oven på T_2, centreret lodret.
- T **inde i** S beskriver en tegning af T inde i den simple figur S.

For eksempel ser

trekant til venstre for (cirkel inde i firkant)

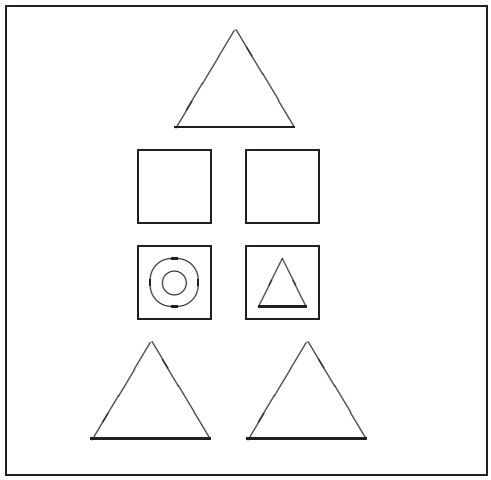
ud som følger:



a) Angiv tegningen beskrevet af

((((firkant inde i firkant) oven på cirkel) til venstre for ((cirkel oven på trekant) inde i trekant)) til venstre for (firkant inde i cirkel)) oven på (trekant inde i trekant)

b) Angiv beskrivelsen af følgende tegning



Er der mere end én beskrivelse, der passer?

c) Hvorfor er det nødvendigt med parenteser i sådanne beskrivelser?

Opgave 1.3.2
Det følgende er eksempler på romertal og deres decimale værdier

III	=	3
XIX	=	19
LVII	=	57
D	=	500
MCMXCII	=	1992

Derimod er nedenstående bogstavfølger *ikke* romertal

ABC
IIII
MLL
XCX
CCM

a) Giv en præcis beskrivelse af, hvilke bogstavfølger der er romertal.

b) Beskriv, hvordan man udregner den decimale værdi af et romertal.

c) Byt svar på a) og b) med en anden på dit øvelseshold. Er svarene præcise nok til at kunne bruges af andre? Er de korrekte?

d) De "rigtige" romertal har den egenskab, at enhver talværdi kan udtrykkes på netop én måde. Kunne man med fordel definere nogle "generaliserede" romertal, der ville gøre det lettere at løse a) og b)?

2 Basale begreber

2.1 Syntaksbeskrivelse

Opgave 2.1.1
Et tog består af et lokomotiv, eventuelt efterfulgt af en kulvogn der kan efterfølges af en postafdeling (en postvogn, eventuelt efterfulgt af brevvogne) eller en almindelig afdeling (person- og/eller godsvogne i vilkårlig rækkefølge). Postafdelinger og almindelige afdelinger kan følge vilkårligt efter hinanden.

a) Med *Tog*, *Postafdeling* og *Almindelig afdeling* som syntaktiske kategorier og *lokomotiv*, *kulvogn*, *postvogn*, *brevvogn*, *personvogn* og *godsvogn* som syntaktiske entiteter, lav da en EBNF-definition for lovlige tog.

b) Omskriv EBNF-definitionen til syntaksdiagrammer.

Opgave 2.1.2
Betragt følgende EBNF-definition:

$$S ::= \text{"1"}\ S\ \text{"0"} \mid \text{"0"}\ S\ \text{"1"} \mid \text{"1"}$$

a) Hvilke af følgende tegnfølger (strenge) er syntaktisk korrekte?

(1) 1

(2) 1 0

(3) 0 1 1

(4) 1 2 0

(5) 1 1 0 0

(6) 1 0 0 1 1 0 1 1 0

(7) 1 0 0 0 1 1 0 1 0

b) Omskriv EBNF-definitionen til et syntaksdiagram.

Opgave 2.1.3
Lav en EBNF-definition for et Tour de France-felt.

Et Tour de France felt med "das ganze molevitten" består af en reklamekaravane efterfulgt af feltet. Ryttere kan gå i udbrud, og det betyder at feltet kan bestå af et antal udbrydergrupper, et hovedfelt og et antal agtergrupper. Der findes en løbslederbil og et antal servicevogne. Hvis der er udbrydergrupper følger løbslederbilen 'course de la tete' (den førende rytter). Servicevogne kan følge ryttere der er gået i udbrud (og gør det når udbrydergruppen har fået et tilpas stort forspring). Til sidst kommer det berygtede fejeblad (en bil der kan samle de ryttere op som vælger at stå af).

Lav syntaksdefinitionen i termer af følgende syntaktiske entiteter:

- r-bil (reklamebil, fra reklamekaravanen)
- rytter
- ll-bil (løbslederbil)
- s-vogn (servicevogn)
- fejeblad

Find selv ud af hvilke syntaktiske kategorier det er hensigtsmæssigt at gøre brug af.

Opgave 2.1.4
Betragt følgende EBNF-definition for figurudtryk:

$$
\begin{aligned}
E &::= E\ op\ E \mid \text{"("}\ E\ \text{")"} \mid E\ \textbf{"inden i"}\ F \mid F \\
op &::= \textbf{"til venstre for"} \mid \textbf{"oven på"} \\
F &::= \text{"T"} \mid \text{"F"} \mid \text{"C"}
\end{aligned}
$$

a) Udvid EBNF-definitionen med operatorerne **uden om**, **til højre for** og **neden under**.

b) Omskriv EBNF-definitionen til et syntaksdiagram.

c) Lav to eksempler på syntaktisk lovlige figurudtryk, og godtgør at de er lovlige. Tegn også de tilhørende figurer.

Opgave 2.1.5
Betragt følgende EBNF-definition:

$$\begin{aligned} \textit{Unique} &::= \textit{Left Right} \\ \textit{Left} &::= \textit{Bit Left Bit} \mid \text{"0"} \\ \textit{Right} &::= \textit{Bit Right Bit} \mid \text{"1"} \\ \textit{Bit} &::= \text{"0"} \mid \text{"1"} \end{aligned}$$

a) Hvilke af nedenstående følger genereres af *Unique*?

(1) 0 1

(2) 0 1 1 0 0 1

(3) 1 1 0 0 1 1 0 0 1 0

(4) 1 0 0 0 1 0 0 1

(5) 0 1 0 1 1 0 1 1 0

b) Omskriv EBNF-definitionen til et syntaksdiagram.

c) Beskriv kort og præcist den mængde af følger der genereres af *Unique*.

2.2 Klasser og objekter

Opgave 2.2.1
I denne opgave skal vi modellerer begrebet person. I modellen — som i enhver anden model vi laver — skal kun medtages de egenskaber ved personer der har interesse i den givne sammenhæng. I dette tilfælde drejer det sig først og fremmest om personens navn og alder. Der kan tages udgangspunkt i vedlagte projekt.

a) Lav en klasse `Person` der modellerer en person og erklær attributter for navn (`String`) og alder (`int`) samt `get-` og `set`-metoder for disse attributter.

I stedet for at lave en standard `set`-metode, lav så en `foedselsdag`-metode som når den kaldes øger en persons alder med én; kald af denne metode er dermed den eneste måde en persons alder kan ændres på.

Lav en konstruktør så det er muligt at oprette `Person`-objekter; lad navnet være en parameter til konstruktøren (alder behøver ikke at være en parameter, alder skal initialiseres til 0).

b) Vi vil gerne kunne afgøre om en person er barn (0-17 år) eller pensionist (67- år).

Lav to metoder, `erBarn` og `erPensionist`, der afgør om en person er barn henholdsvis pensionist. Tag udgangspunkt i følgende metode der afgør om en person er teenager:

```
/**
 * Returnerer om personen er teenager
 * @return true, hvis personen er teenager, false ellers
 */
public boolean erTeenager()
{
    return (13 <= age  &&  age <= 19);
}
```

Som det fremgår har metoden returtype `boolean` hvilket betyder at metoden returnerer en sandhedsværdi (`true` eller `false`). Operatoren && betegner logisk 'og', så det metoden returnerer, er sandhedsværdien af om alder er mindst 13 og højst 19, hvilket præcis er definitionen på begrebet teenager.

Opgave 2.2.2
Denne opgave bygger videre på opgave 2.2.1.

a) Udvid `Person`-klassen med en ekstra attribut (field) som beskriver personens intelligenskvotient (et heltal). Modificér konstruktøren ved at tilføje en parameter for intelligenskvotient, og benyt denne parameter til i kroppen af metoden at initialisere attributten, der repræsenterer en persons intelligenskvotient. Lav endvidere en accessor-metode, `getIntelligens`, og en mutator-metode, `setIntelligens` til at kunne spørge om og opdatere en persons intelligenskvotient.

b) En person er *menzaner* hvis intelligenskvotienten er 140 eller derover. Skriv en metode der afgør om en person er menzaner (d.v.s. returnerer `true` eller `false` afhængig af om personen er menzaner eller ej).

c) Udvid `run`-metoden i `Driver`-klassen til at oprette fire `Person`-objekter, og derefter beregne og udskrive gennemsnittet af deres intelligenskvotient.

d) Udvid `run`-metoden i `Driver`-klassen til for hver af de oprettede personobjekter at udskrive om den pågældende person er menzaner eller ej.

Opgave 2.2.3
Denne opgave bygger videre på opgave 2.2.1.

Vi er i denne opgave interesseret i en persons relationer til andre personer, specielt er vi interesseret i hvem en person eventuelt er forelsket i.

a) Udvid `Person`-klassen med en attribut elsker (af typen `Person`) der holder styr på hvem (om nogen) personen er forelsket i. Tilføj get- og set-metoder for denne attribut. Hvis get-metoden hedder `getElsker`, skal den erklæres som følger:

```
/**
 * Returnerer personens elsker
 * @return personens elsker (null, hvis der ikke er nogen)
 */
public Person getElsker() { ... }
```

b) Nu vil vi også holde styr på hvem (om nogen) en persons kæreste er.

Udvid `Person`-klassen med en attribut der holder styr på hvem (om nogen) personen er kæreste med. Tilføj get- og set-metoder for denne attribut.

c) Som vi alle ved er man ude i noget 'snavs' hvis ens elsker ikke er den man er kæreste med.

Udvid `Person`-klassen med en metode der kan afgøre om en person er ude i noget snavs. Hvis metoden hedder `isALousyGuy`, skal den erklæres som følger:

```
/**
 * Returnerer om personen er ``ude i noget snavs'' d.v.s.
 * om personens elsker er en anden end personens kaereste
 * @return om personen er ``ude i noget snavs''
 */
public boolean isALousyGuy() { ... }
```

Sørg også for at håndtere situationen hvor en person ikke har en kæreste eller ikke har en elsker.

d) I denne model vil vi definere begrebet lykkelig som følger: en person er lykkelig præcis hvis 1) personen ikke er forelsket, eller 2) den personen

er forelsket i også er forelsket i personen.

Udvid `Person`-klassen med en metode der kan afgøre om en person er lykkelig. Med andre ord skal metoden returnere sand (`true`) hvis personen er lykkelig jævnfør ovenstående definition, og ellers skal metoden returnere falsk (`false`). Kald metoden `isHappy`.

Hint: I denne opgave får du brug for objektreferencen `this` som i en metode på `Person`-klassen er en reference til den person som den aktuelle besked er sendt til.

Opgave 2.2.4
Denne opgave bygger videre på opgave 2.2.1.

Vi er i denne opgave interesseret i at holde styr på en persons vennekreds.

a) Udvid `Person`-klassen med tre metoder:

```
/**
 * Tilfoej p til denne persons vennekreds
 * @param p Personen, der skal tilfoejes
 */
public addFriend(Person p) { ... }

/**
 * Fjern p fra denne persons vennekreds
 * @param p Personen, der skal fjernes
 */
public removeFriend(Person p) { ... }

/**
 * Udskriv en liste over denne persons venner
 */
public printFriends() { ... }
```

b) Lav et testprogram (i en `Driver`-klasse) der aftester din løsning.

c) Lav en metode på `Person`-klassen, der kan afgøre om en person har en ven (person i sin vennekreds) med et bestemt navn. Metoden skal have følgende signatur:

```
/**
 * Har denne person en ven med navnet navn
 * @param navn Navnet paa personen
 * @return om personen har en ven med det navn
```

```
 */
public boolean harVen(String navn) { ... }
```

Metoden skal implementeres ved hjælp af en `Iterator`.

Opgave 2.2.5
Denne opgave bygger videre på opgave 2.2.4.

a) For at kunne lave en hurtigere søgning er det besluttet at vennregistret skal implementeres som en `HashMap`. `HashMap`'en skal have vennens navn som nøgle og vennen (objektet af typen `Person`) som værdi.

   Ret i metoderne `addFriend` og `removeFriend` samt konstruktøren, så denne ændring gennemføres.

b) Ret metoden `harVen`, så den tager højde for, at vennekredsen er realiseret som en `HashMap`

c) Lav et testprogram (i en `Driver`-klasse) der aftester din løsning.

Opgave 2.2.6
Denne opgave skal laves med en instruktor som "tastaturfører" idet der er mange ting de studerende ikke har forudsætningerne for at lave på egen hånd.

Opgaven tager udgangspunkt i projektet actors.

a) Lav en metode `getAntal` som returnerer det antal tricks en `Aktoer` har lavet til dato.

   Hvordan skal denne metode implementeres i de forskellige klasser? Kan alle benytte standardimplementationen fra `Aktoer`-klassen?

b) Modificér metoden `skrivAntalTricks` så den benytter `getAntal` til at foretage udskrivningen.

c) Definér et nyt skuespil (en ny metode i `Instruktoer`-klassen) hvor der oprettes tre koreografer, `a`, `b` og `c`, samt en til fire skuespillere af vilkårlig slags. Lad `a` være knyttet til `b` og `c`, og lad `b` og `c` være knyttet til hver to skuespillere.

d) Gentage forrige delopgaver, men nu med syv koreografer (1+2+4) og et passende antal skuespillere. Modificér metoden `skrivAntalTricks` så den benytter metoden `getAntal` til at foretage udskrivningen.

Opgave 2.2.7

a) Lav en klasse, `Terning`, der repræsenterer en spilleterning. Der skal være to metoder i klassen, `kast` og `antalØjne`, der giver mulighed for

dels at slå med termingen, dels at inspicere hvor mange øjne terningen p.t. viser. Det skal være muligt at inspicere terningen mange gange mellem hvert kast.

Hint: Lav klassen inkrementelt, d.v.s. i små skridt. Start f.eks. med at implementere metoden `kast` ved altid at lade den resultere i at der slås en sekser. Når du har fået dette til at fungere tages næste skridt i implementeringen.

For at opnå et realistisk element af tilfældighed i udfaldet af slag med terningen, kan metoden `Math.random` benyttes. Den returnerer et tilfældigt tal (`double`) i intervallet $[0; 1[$ (se beskrivelsen af klassen `Math` i dokumentationen for Javas API). Multipliceres dette tal med 6 fremkommer et tal i intervallet $[0; 6[$. Adderes 1 fremkommer en double-værdi i intervallet $[1; 7[$, og hvis denne afskæres til et heltal (med et type cast (`int`)) fremkommer et heltal i det lukkede interval $[1; 6]$.

b) Lav en klasse, `Raflebaeger`, der repræsenterer et raflebæger med to terninger. Der skal være to metoder i klassens interface, `slå` og `antalØjne`, der giver mulighed for dels at slå med bægeret, dels at inspicere hvor mange øjne der blev slået i alt.

Hint: Implementer klassen med to attributter (feltvariable) af typen `Terning`.

c) Lav en klasse, `Statistiker`, der repræsenterer en person der udfører statistiske eksperimenter med en terning. Statistikeren slår et antal gange med en terning og producerer i den forbindelse et histogram der viser antallet af 1'ere, 2'ere, osv. der blev slået med terningen. Implementér den beskrevne opførsel i en metode, `udførEksperiment` med følgende signatur:

```
public void udførEksperiment(int antalSlag)
```

og test metoden med et antal eksperimenter med forskelligt antal slag. Hvad indikerer eksperimenterne om "tilfældighedskvaliteten" af metoden `Math.random`?

Hint: Lad statistikeren benytte et passende antal farvekridt (`Crayon`) til at producere histogrammet.

2.3 JAVA

Opgave 2.3.1
Betragt følgende JAVA-program:

```
public class classfield
{
    public static void main( String args[] )
    {
        B.method1();                            // (1)
        B.method3();                            // (2)
        B.method5();                            // (3)
        B.method6();                            // (4)

        B instance = new B();

        instance.method1();                     // (5)
        instance.method3();                     // (6)
        instance.method5();                     // (7)
        instance.method6();                     // (8)
    }
}

public class A
{
    public static int x = 42;
    public int y = 43;

    public void method1()
    {
        x = 52;
    }
    public void method2()
    {
        y = 53;
    }
    public static void method3()
    {
        x = 62;
    }
    public static void method4()
```

```
    {
      y = 63;
    }
}

public class B extends A
{
    public void method5()
    {
        method1();
        method3();
    }
    public static void method6()
    {
        method2();
        method4();
    }
}
```

Beskriv og begrund for hver af metoderne `method1` til `method6` samt sætningerne (1) til (8) om de er gyldige i JAVA og, hvis dette er tilfældet, hvad der sker på objekt- og klasseniveau.

Opgave 2.3.2

a) Hvad er output fra følgende program (og hvorfor)?

```
public class Scope
{
    int i, j;

    public Scope()
    {
        i = 0;
        j = 1;
    }

    public void pip()
    {
        i++;
        {
            i++;
            write(i);
```

```
            int j = i, i = 0;
            i++;
            write(j); write(i); write(-1);
            pap();
            write(i); write(j); write(-1);
            kuk();
            write(i); write(j); write(-1);
        }
    }

    public void pap()
    {
        int j = i;
        write(j);
        j++;
        i++;
    }

    public void kuk()
    {
        int i = j;
        write(i);
        j++;
        i++;
    }

    private static void write(int i)
    {
        System.out.println(i);
    }

    public static void testMethod()
    {
        Scope s = new Scope();
        s.pip();
    }
}
```

b) Hvad sker der hvis erklæringen `j = i, i = 0` i `pip` erstattes af `int i = 0, j = i`? Forklar.

3 Kontrolstrukturer

3.1 Metodekald

Opgave 3.1.1
Betragt følgende program:

```
public class Exercise
{
    public static void main(String args[])
    {
        A instance;
        instance = new A(); instance.poiap();
        instance = new B(); instance.poiap();
        instance = new C(); instance.poiap();
        instance = new D(); instance.poiap();
    }
}

public class A
{
    public String pop() { return "Apop "; }
    public String pip() { return "Apip "; }
    public String pap() { return "Apap "; }
    public void poiap()
    {
        System.out.println( pop() + pip() + pap() );
    }
}

public class B extends A
{
    public String pop() { return "Bpop "; }
    public String pap() { return "Bpap " + super.pap(); }
}
```

```
public class C extends B
{
    public String pop() { return "Cpop "; }
    public String pip() { return "Cpip " + super.pip(); }
}

public class D extends C
{
    public String pop() { return "Dpop " + super.pop(); }
    public String pap() { return "Dpap " + super.pap(); }
}
```

Beskriv for hver af de fire kald af metoden `poiap` i hovedprogrammet hvad programmet udskriver og forklar kaldssekvensen (d.v.s. rækkefølgen af metodekald) for hvert kald.

Opgave 3.1.2
Betragt følgende klasser:

```
public class Eksempel
{
    private int a;

    public Eksempel(int a) { this.a = a; }

    public int metodeA (int a) { return a*3; }
    private ??? metodeB (int m) { return (m%4)==0; }
    public int inc1() { a++; return a; }
    public int inc2() { int a = 0; a++; return a; }
}

public class EksempelTester
{
    public static void main(String[] args)
    {
        int a = 5;
        int b = 6;
        Eksempel e1 = new Eksempel(b);
        Eksempel e2 = new Eksempel(8);

        System.out.println(e1.metodeA(a));
        System.out.println(e1.metodeB(7));
```

```
        System.out.println(e1.inc1());
        System.out.println(e1.inc2());
        System.out.println(e1.inc2());
        System.out.println(e1.a);
        System.out.println(e2.inc1());
        System.out.println(e2.inc2());
    }
}
```

a) Hvad skal `???` erstattes med i `Eksempel`-klassen?

b) Hvilke linier i `EksempelTester` accepteres *ikke* af JAVA-oversætteren? Hvorfor?

c) Antag at de ulovlige linier er fjernet fra `EksempelTester`. Hvad bliver skrevet ud, når `EksempelTester` køres?

3.2 Selektion

Opgave 3.2.1
De logiske operatorer $\wedge$, $\vee$ og $\neg$ kan beskrives ved sandhedstabellerne (0 betegner `false` og 1 betegner `true`):

x	$\neg$
0	1
1	0

x	y	$\wedge$
0	0	0
0	1	0
1	0	0
1	1	1

x	y	$\vee$
0	0	0
0	1	1
1	0	1
1	1	1

a) Lav en tilsvarende sandhedstabel for $\Rightarrow$ (logisk implikation).

b) Konstruér et JAVA-udtryk for $x \Rightarrow y$.

Følgende sandhedstabel definerer en boolsk funktion *select* af tre variable:

x	y	z	*select*
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

c) Overvej hvorfor funktionen benævnes *select*.

d) $select(x, y, z)$ kan i JAVA beregnes direkte ved udtrykket

```
(x ? y : z)
```

Konstruér et JAVA-udtryk for $select(x, y, z)$ uden at anvende konstruktionen `(x ? y : z)`.

3.3 Iteration

Opgave 3.3.1
Følgende klasse beskriver en byttepengeautomat, der ud fra en (ubegrænset) beholdning af 1-, 5- og 10-kroner udbetaler et beløb.

```
public class Change
{
    private int k_1 = 0;
    private int k_5 = 0;
    private int k_10 = 0;

    public Change(int amount)
    {
        if(amount < 0) {
            throw new RuntimeException("negative amount");
        }
        int rest = amount;
        while(rest >= 10) {
            rest -= 10;
            k_10++;
        }
```

```
        while(rest >= 5) {
            rest -= 5;
            k_5++;
        }
        while(rest >= 1) {
            rest--;
            k_1++;
        }
    }
    public int get_1()  { return k_1;}
    public int get_5()  { return k_5;}
    public int get_10()  { return k_10;}
}
```

a) Argumentér for, at automaten altid udbetaler så få mønter som muligt.

b) Modificér klassen, så en byttepengeautomat også kan udbetale 7-kroner.

c) Udbetaler den ændrede automat så få mønter som muligt? Hvis ikke, kan du skrive en klasse, der gør?

Opgave 3.3.2
Følgende metode, hvor indmaden er fjernet, konverterer arabertal til romertal.

```
/**
 * toRoman konverterer arabertal til romertal
 * @param n Et positivt heltal fra intervallet 1,...,399
 * @return romertallet svarende til n
 */
public String toRoman(int n) { ... }
```

F.eks vil udtrykket `toRoman(78)` evaluere til `"LXXVIII"`, og `toRoman(94)` vil evaluere til `"XCIV"`

a) Skriv indmaden til metoden.

b) Skriv et program, der indlæser et heltal i intervallet 1,...,399 og udskriver det tilsvarende romertal.

3.4 Rekursion

Opgave 3.4.1
Betragt følgende rekursive funktion:

```
/**
 * fib beregner det n'te Fibonnacci-tal
 * @param n Et positivt heltal
 * @return det n'te Fibonnacci-tal
 */
public int fib (int n)
{
    if(n == 1) {
        return 1;
    }
    else if(n == 2) {
        return 1;
    }
    else {
        return fib(n - 1) + fib(n - 2);
    }
}
```

a) Hvor mange kald af funktionen vil blive genereret af kaldet `fib(8)`? Hvad med `fib(50)`?

b) Kan du finde en mere effektiv måde at implementere funktionen på?

(Hint: Brug iteration, ikke rekursion.)

Opgave 3.4.2

a) Skriv en rekursiv funktion, der beregner n^p, $p \geq 0$, defineret ved:

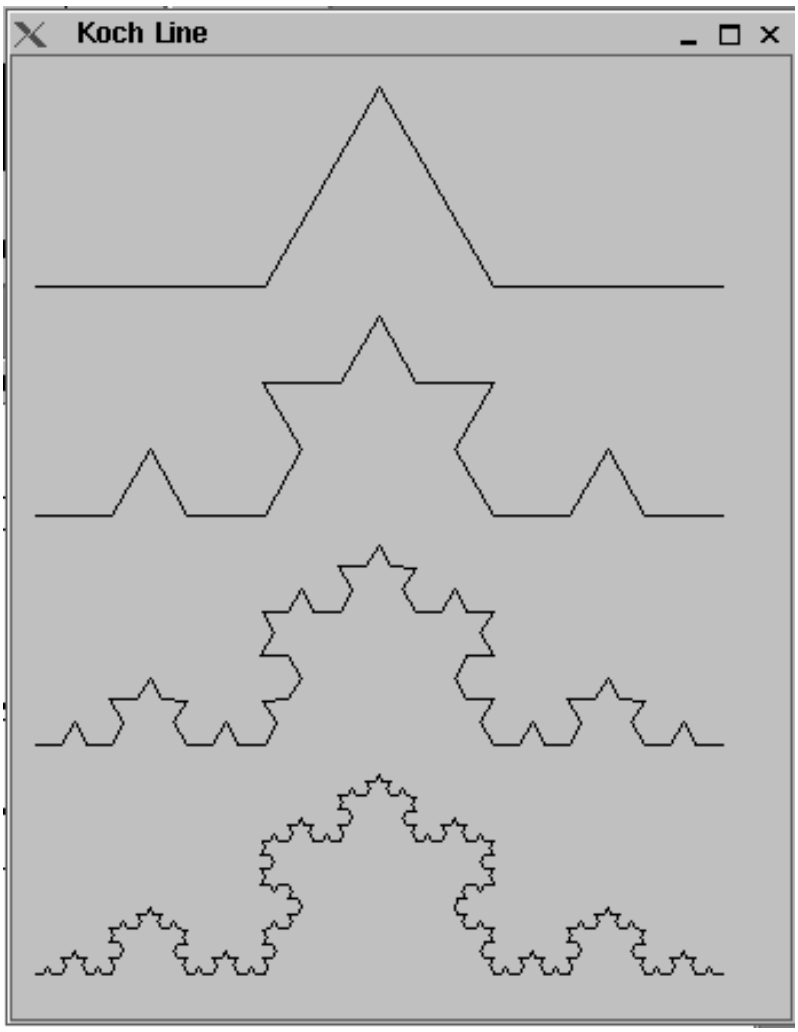
$$n^p = \begin{cases} 1, & p = 0, \\ n^{p-1} * n, & p > 0. \end{cases}$$

b) Skriv derefter en rekursiv funktion med samme specifikation, men ud fra følgende definition:

$$n^p = \begin{cases} 1, & p = 0, \\ n^{p-1} * n, & p > 0 \text{ og } p \text{ er ulige}, \\ (n * n)^{p/2}, & p > 0 \text{ og } p \text{ er lige}. \end{cases}$$

c) Nogen kommentarer?

Opgave 3.4.3
En *fraktal* er en kurve, som intetsteds er glat nok til at kunne approksimeres med linjestykker. Et simpelt eksempel på en fraktal er den såkaldte *Koch-kurve*, der opstår som grænseværdien (for n gående mod uendelig) af *Koch-linjen* af orden n. En Koch-linje af orden n opnås ud fra en Koch-linje af orden $n-1$ ved at give alle linjestykker i den et trekantet hak. De følgende er Koch-linjer af orden 1, 2, 3 og 4:



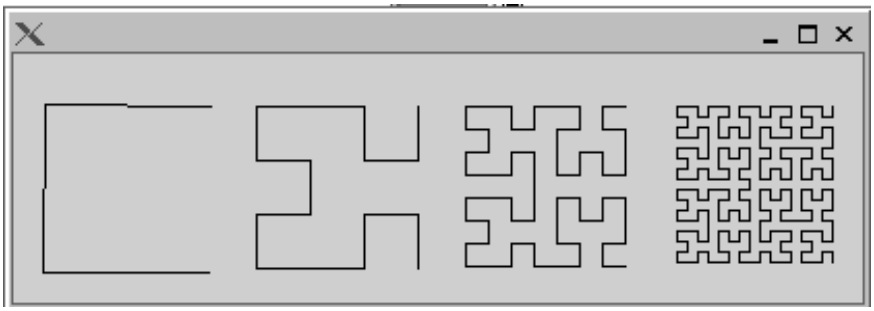
De 4 Koch-linjer er alle tegnet ved hjælp af det følgende JAVA-program, der er bygget op omkring den rekursive metode `kochLine`, og anvender klassen `Crayon` fra opgave 1.2.1.

```
import java.awt.*;

public class Koch
{
    private static Crayon pencil = new Crayon(Color.black,1);

    /**
     * kochLine tegner en Koch-linje af orden n >= 0.
     * @param n Ordenen af den Koch-linje der tegnes
     * @param dir Retning fra start til slutpunkt i grader
     * @param len Afstand mellem kurvens start og slutpunkt
     */
    private static void kochLine(int n, int dir, double len)
    {
        if(n == 0) {
            pencil.turnto(dir);
            pencil.move(len);
        }
        else if(n > 0) {
            kochLine(n - 1,dir,len/3);
            kochLine(n - 1,dir - 60,len/3);
            kochLine(n - 1,dir + 60,len/3);
            kochLine(n - 1,dir,len/3);
        }
    }

    public static void main(String[] args)
    {
        pencil.jumpto(50,150);
        kochLine(1,0,300);
        pencil.jumpto(50,250);
        kochLine(2,0,300);
        pencil.jumpto(50,350);
        kochLine(3,0,300);
        pencil.jumpto(50,450);
        kochLine(4,0,300);
    }
}
```

Bemærk, at vi foretager 4 rekursive kald, og at vi her har et eksempel på en rekursiv metode, hvor en ikke-rekursiv formulering forekommer at være

endog meget besværlig.

En *Hilbert kurve* af orden n er den n'te tegning i følgende række, der nummereres H_1, H_2, H_3, H_4, og så videre.



a) Angiv "rekursionsformlen" for H_n.

b) Skriv et program, der kan tegne Hilbert kurver. Følgende skabelon med de indbyrdes rekursive metoder west, north, east og south kan evt. benyttes:

```
import java.awt.*;

public class Hilbert
{
    private static Crayon pencil = new Crayon(Color.black,1);

    private static void west(int n, double len)
    {
        if(n > 0) {
            south(n - 1,len);
            pencil.turnto(180); pencil.move(len);
            west(n - 1,len);
            pencil.turnto(90); pencil.move(len);
            west(n - 1,len);
            pencil.turnto(0); pencil.move(len);
            north(n - 1,len);
        }
    }

    private static void north(int n, double len) { ... }

    private static void east(int n, double len) { ... }
```

```
    private static void south(int n, double len) { ... }

    public static void main(String[] args)
    {
        pencil.jumpto(100,50);
        west(1,80/(2 - 1));
        pencil.jumpto(200,50);
        west(2,80/(4 - 1));
        pencil.jumpto(300,50);
        west(3,80/(8 - 1));
        pencil.jumpto(400,50);
        west(4,80/(16-1));
    }
}
```

Opgave 3.4.4
Euklids algoritme til beregning af største fælles divisor af to positive heltal er baseret på reglerne:

$$\gcd(x,y) = \begin{cases} x, & \text{for } x = y, \\ \gcd(x-y,y), & \text{for } x > y, \\ \gcd(x,y-x), & \text{for } y > x. \end{cases}$$

a) Skriv en *rekursiv* metode (med to parametre x og y) som beregner og returnerer $\gcd(x,y)$.

b) Skriv en ny metode, der ligeledes beregner $\gcd(x,y)$, men anvender en `while`-løkke i stedet for rekursion.

c) Argumentér for at udførelsen af metoderne fra a) og b) altid standser.

Opgave 3.4.5
Binomialkoefficienten $\binom{n}{k}$ angiver på hvor mange måder man kan udtage k elementer fra en mængde med n elementer, og den opfylder følgende ligning (for $n \geq k \geq 0$):

$$\binom{n}{k} = \begin{cases} \binom{n}{k-1} \cdot (n-k+1)/k, & \text{for } k > 0 \\ 1, & \text{for } k = 0 \end{cases}$$

a) Skriv en *rekursiv* metode `binomial` (med to parametre n og k) som beregner og returnerer $\binom{n}{k}$.

b) Lav et "execution trace" for evalueringen af udtrykket `binomial(4,2)`.

4 Datatyper og datastrukturer

4.1 Array

Opgave 4.1.1
Skriv indmaden til denne metode:

```
/**
 * mirror constructs the mirror image of a matrix
 * across its diagonal.
 * @param r A matrix
 * @return a matrix that looks like r mirrored across its
 * diagonal
 */
public int[][] mirror(int[][] r) { ... }
```

D.v.s. hvis den aktuelle parameter er en $m \times n$ matrix, M, så bliver resultatet en $n \times m$ matrix, N, hvor $N[i][j] = M[j][i]$, d.v.s.

$$N = \begin{bmatrix} M[0][0] & M[1][0] & \dots & M[m-1][0] \\ M[0][1] & M[1][1] & \dots & M[m-1][0] \\ \vdots & \vdots & & \vdots \\ M[0][n-1] & M[1][n-1] & \dots & M[m-1][n-1] \end{bmatrix}$$

Opgave 4.1.2
I denne opgave betragtes en datatype for håndtering af lange heltal, hvor datatypens værdi er

- et ikke-negativt heltal bestående af op til 100 cifre,

og datatypens operationer kan

- konstruere et tal bestående af et enkelt ciffer,
- addere to lange heltal,
- udskrive et langt heltal.

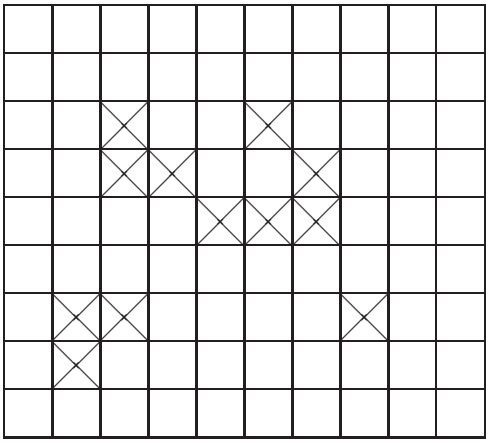
a) Vis, hvordan et langt heltal kan repræsenteres som en værdi af typen `int[]`.

b) Realisér datatypen som en klasse i Java. Du kan evt. benytte vedlagte skabelon.

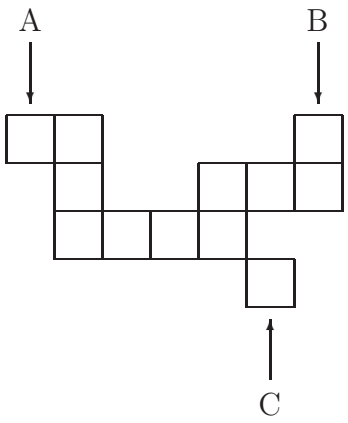
c) Brug datatypen til at beregne Fibonnacci-tal nummer 100 og 2-potensen 2^{100}.

```
public class Lang
{
    private static final int DIGITS = 100;
    private static final int BASE = 10;

    private int[] value;

    // REPRÆSENTATIONSINVARIANT:
    // value er et array af længde DIGITS
    // hvert tal i value har en værdi i intervallet
    // 0,1,2,...,BASE-1

    /**
     * Lang konstruerer et langt heltal
     * @param i, hvor 0 <= i < BASE
     */
    public Lang(int i) { ... }

    /**
     * plus finder summen af to lange heltal
     * forudsat summen har <= DIGITS cifre
     * @param t Et langt heltal
     * @return summen af t og dette lange tal */
    public Lang plus(Lang t) { ... }

    /**
     * toString konverterer tallet til en cifferfølge
     * forudsat BASE==10.
     * @return værdien af det lange heltal på tekstform */
    public String toString() { ... }
}
```

Opgave 4.1.3
På et ternet stykke papir er ternene enten hvide eller sorte



To tern er *naboer*, hvis de har en side fælles, og to tern er *sammenhængende*, hvis de er forbundet af en følge af naboer der har samme farve. I nedenstående tegning er fx ternene A og B sammenhængende, hvorimod C ikke hænger sammen med noget.



Et stykke ternet papir kan repræsenteres af klassen, der er vist på næste side.

a) Udfyld kroppen af metoden `count`, så den opfylder specifikationen. Det kan antages, at papirets yderste kant er helt hvid. Hint: definer en ekstra private metode.

```
/**
 * CheckedPaper definerer et stykke ternet papir med en række
 * operationer
 */
public class CheckedPaper
{
    public static final int HEIGHT = 57;
    public static final int WIDTH = 40;
    public static final int WHITE = 1;
    public static final int BLACK = 2;

    private int[][] paper = new int[HEIGHT][WIDTH];

    /**
     * clear farver alle tern hvide
     */
    public void clear()
    {
        for(int i = 0; i < HEIGHT; i++) {
            for(int j = 0; j < WIDTH; j++) {
                paper[i][j] = WHITE;
            }
        }
    }

    /**
     * colour farver tern (i,j)
     * @param i Rækkenummer (0 <= i < HEIGHT)
     * @param j Søjlenummer (0 <= j < WIDTH)  */
    public void colour(int i, int j) { paper[i][j] = BLACK; }

    /**
     * count returnerer antal farvede tern, i sammenhæng med
     * tern(i,j)
     * @param i Rækkenummer (0 <= i < HEIGHT)
     * @param j Søjlenummer (0 <= j < WIDTH)
     * @return 0 hvis tern(i,j) er hvid; ellers antal sorte
     * tern, der er sammenhængende med tern(i,j)
     */
    public int count(int i, int j) { // Udfyld ! }
}
```

Opgave 4.1.4
I eksempelkataloget for denne opgave findes to klasser `Card` og `CardDeck` der kan bruges til at repræsentere et sæt spillekort.

Følgende klasse beskriver en spiller, der kan modtage en korthånd et kort ad gangen:

```
public class Player
{
    private Card[] deck;
    private int size;

    public Player(int hand_size)
    {
        deck = new Card[hand_size];
        size = 0;
    }

    public void receiveCard(Card c)
    {
        if(size<deck.length)
        {
            deck[size] = c;
            size++;
        }
    }

    public void printHand()
    {
        for(int i = 0; i < size; i++) {
            System.out.println(deck[i].suitOf() + " " +
                               deck[i].countOf());
        }
    }
}
```

Metoden `main` i følgende klasse bevirker at en tilfældig hånd på tretten kort udskrives:

```
public class PlayerTest
{
    public static void main(String[] args)
    {
        CardDeck d = new CardDeck();
        Player p = new Player(13);
        for(int i = 0; i < 13; i++) {
            p.receiveCard(d.newCard());
        }
        p.printHand();
    }
}
```

a) Omskriv metoden `printHand`, så kortene udskrives pænt sorterede, f.eks. således:

```
S   K Q 10 8
H   A K 7
R   Q J 9 2
K   8 3
```

b) Omskriv metoden `main`, så alle 52 kort uddeles i 4 bridgehænder (à 13 kort), som derpå udskrives (Du bør anvende et `array` af `Player`).

Opgave 4.1.5
I denne opgave betragtes en datatype for heltalsmængder, hvor datatypens værdi er

- en mængde af heltal: s

og datatypens operationer kan

- konstruere den tomme mængde: $s = \emptyset$
- indsætte et tal i mængden: $s = s \cup \{i\}$
- afgøre om et givet tal er i mængden $i \in s$?
- udskrive listen af mængdens elementer

Denne datatype angiver et *interface* der kan implementeres ved en JAVA-klasse `Set`, som vist på næste side.

```
public class IntSet
{
    // s indeholder mængdens tal
    private int[] s;

    /**
     * IntSet konstruerer en tom mængde
     */
    public IntSet() { s = new int[0]; }

    /**
     * insert indsætter et tal i mængde n
     * @param i Det tal der skal indsættes i s
     */
    public void insert(int i)
    {
        if(! member(i)) {
            int[] t = new int[s.length + 1];
            for(int j = 0; j < s.length; j++) { t[j] = s[j]; }
            t[s.length] = i;
            s = t;
        }
    }

    /**
     * member tester medlemskab af mængden s
     * @param i Et tal
     * @return sandhedsværdien af udsagnet
     * tallet i ligger i mængden s
     */
    public boolean member(int i)
    {
        for(int j = 0; j < s.length; j++) {
            if(s[j] == i) {
                return true;
            }
        }
        return false;
    }
```

```
    /**
     * toString producerer en tekstrepræsentation
     * @return mængden på tekstform
     */
    public String toString()
    {
        if(s.length == 0) { return "empty set"; }
        String result = "";
        for(int j = 1; j < s.length; j++) {
            result += ", " + s[j];
        }
        return s[0] + result;
    }
}
```

`IntSet` kan bruges til at indlæse en række heltal og udskrive de tal, der forekommer mere end en gang, som i det følgende eksempel:

```
import javax.swing.*;

public class Dublet
{
    public static void main(String[] args)
    {
        IntSet numbers = new IntSet();
        IntSet dublets = new IntSet();
        int n = Integer.parseInt(JOptionPane.showInputDialog(
          "type integer: "));
        while(n != 0) {
            if(numbers.member(n)) { dublets.insert(n); }
            else { numbers.insert(n); }
            n = Integer.parseInt(JOptionPane.showInputDialog(
              "type integer: "));
        }
        System.out.println(dublets);
    }
}
```

a) Udvid datatypen med en operation `delete`, der kan slette et heltal: $s = s - \{i\}$.

b) Brug den udvidede datatype til følgende: Indlæs en række tal afsluttet med 0; udskriv de af tallene, der forekommer et ulige antal gange.

5 Modellering

5.1 Objektorienteret modellering

Opgave 5.1.1
I det følgende skitseres opbygningen af et *adventure*-lignende computerspil. Læs beskrivelsen igennem og opstil en objektbaseret model for spillet, d.v.s. identificér objekter og operationer på objekter.

Adventure-spil Spillet simulerer en simpel bygning. Bygningen har mindst fire rum. Der er mindst tre personer i bygningen; en person kontrolleres af spillets bruger og de øvrige kontrolleres af datamaskinen. Der er også nogle genstande i rummene.

En person kan bevæge sig fra det rum, hun er i, til et tilstødende rum, hvis de to rum er forbundet med en dør. Det skal være muligt at komme fra ethvert rum til ethvert andet rum, men alle rum bør ikke være direkte forbundne.

En person kan også opsamle og bære ting fra et rum til et andet rum, hvor der gives slip på dem igen. En person kan højst bære to ting ad gangen.

Programmet tilbyder brugeren en kommando-menu med bl.a.

Look around giver brugeren en beskrivelse af det rum, som hun er i lige nu, omfattende døre, ting og andre personer.

Inventory giver en liste over de ting brugeren bærer på lige nu.

Pick up giver brugeren mulighed for at samle en ting op.

Set down lader brugeren afhænde en ting.

Move lader brugeren gå igennem en af de mulige døre.

Quit afslutter programmet.

Programmet kan alternere mellem alle mennesker i bygningen. Når det er brugerens tur, vælger hun en kommando. Når det er en af de øvrige personers tur, lader programmet denne person vælge tilfældigt mellem at tage/afhænde en ting eller gå igennem en mulig dør.

Opgave 5.1.2
En *musiker* spiller muligvis i et *band* (men højst i ét). Der kan optages nye medlemmer i et band og der kan være vilkårligt mange musikere i et band.

I denne opgave skal du lave et system, hvori man skal kunne:

- oprette en musiker
- oprette et band
- optage en musiker i et band
- ekskludere en musiker fra et band
- udskrive en liste over de musikere, der spiller i et bestemt band (herunder hvilket instrument, de hver især spiller på)
- få oplyst, om en musiker spiller i et band
- hvis en musiker spiller i et band, skal man kunne få oplyst, hvilket band det drejer sig om

Du skal løse følgende opgaver:

a) Skitsér et UML klassediagram for systemet.

b) Realisér systemet i JAVA; dette inkluderer (naturligvis) at skrive et testprogram, der aftester og demonstrerer at systemet virker efter hensigten.

c) En festival er som bekendt en samling bands der spiller (typisk over nogle dage). Udvid modellen til at omfatte festival-begrebet. Det skal være muligt at:

   - oprette en festival
   - knytte et band til en festival
   - udskrive listen af bands der er hyret til en festival
   - udskrive listen af festivaler som et band skal spille på

Opgave 5.1.3
Der skal laves et system á la PowerPoint der kan lave dias som indeholder både tekst og grafik. Systemet skal være effektivt, brugervenligt og med mulighed for senere udvidelse af funktionaliteten.

Systemet skal være et WYSIWYG-system hvor brugeren løbende skal kunne se hvordan den færdige præsentation kommer til at se ud. Tilsvarende skal det være muligt at ændre layout og indhold af præsentationens dias på en simpel måde og gennem direkte manipulation af elementerne i hvert dias.

For eksempel skal man kunne lave en figur på et dias ved at lægge et grafisk billede og en billedtekst ind og få tegnet en kasse rundt om. Det skal være

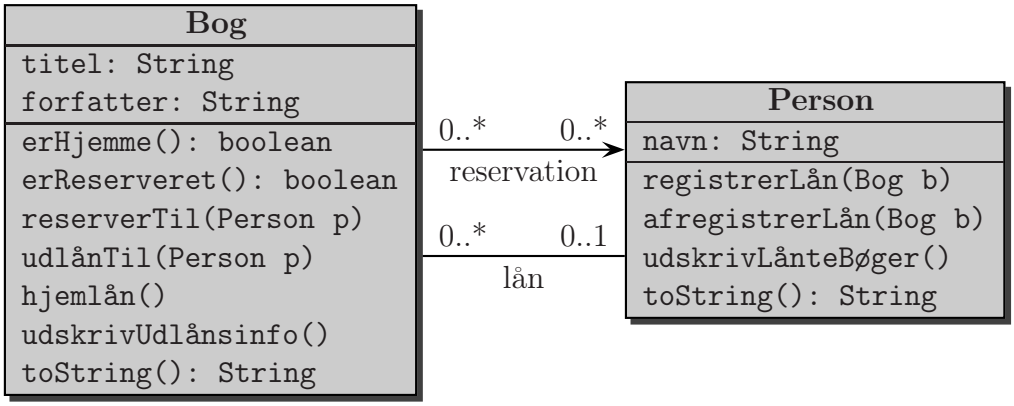
muligt at skifte størrelse på figuren, og kassens størrelse skal da følge figurens størrelse.

Det skal være muligt at flytte figuren både sammen med og uafhængigt af teksten (dog ikke samtidig ;-), og det skal være muligt at gruppere elementer i et dias således at disse kan manipuleres under et (for eksempel ved flytning, kopiering, animering, etc.).

a) Lav en klassemodel for centrale dele af den skitserede applikation.

b) Implementér og aftest modelkomponenten.

Opgave 5.1.4
Opgaven går ud på at implementere følgende UML-klassemodel i JAVA; der kan med fordel tages udgangspunkt i biludlejningseksemplet `car-rental`.



a) Implementér først klassen `Bog` med en passende konstruktør.

   Implementér på tilsvarende vis klassen `Person`.

   I første omgang skal du ikke bekymre dig om realisering af associeringerne, og dermed heller ikke om metoderne `udlånTil` og `hjemlån` i klassen `Bog` samt metoderne `registrerLån` og `afregistrerLån` i klassen `Person`.

b) Hvis en bog er udlånt, kan man reservere bogen (og låne den når den afleveres og det er blevet ens tur). Associeringen *reservation* holder styr på de reservationer der aktuelt er i systemet.

   Implementér associeringen *reservation* ved at implementere metoden `reserverTil` i klassen `Bog`. Bemærk pilen på associeringen *reservation*

i retning mod klassen `Person`; pilen betyder at associeringen kun skal implementeres den ene vej: fra `Bog` til `Person`.

c) Implementér associeringen *lån* ved først at implementere associeringen i retningen fra `Bog` til `Person` (erklær en passende attribut i klassen `Bog`, initialisér denne i konstruktøren og opdater den i metoderne `udlån` og `hjemlån`). Tag specielt stilling til hvorledes opførslen af metoden `udlånTil` skal være hvis bogen i forvejen er udlånt.

   Implementér herefter den anden del af associeringen, nemlig i retningen fra `Person` til `Bog`. Implementér denne del af associeringen i metoderne `registrerLån` og `afregistrerLån` i klassen `Person` og kald disse på behørig vis fra metoderne `udlånTil` og `hjemlån` i klassen `Bog` (sammenlign med biludlejningseksemplet).

   Bemærk at der ved hjemlån skal checkes for om en bog er reserveret. Hvis en bog er reserveret, skal der foretages et udlån til den person der har reserveret bogen (hvis flere personer har reserveret bogen, skal udlånet foretages til den person der har reserveret bogen i længst tid). Den pågældende persons reservation slettes.

d) Implementér metoden `toString` på begge klasser. I klassen `Person` skal metoden bare returnere personens navn; i klassen `Bog` skal metoden returnere en streng med titel og forfatter.

e) Implementér metoden `udskrivLånteBøger` i klassen `Person`. Metoden skal udskrive en liste over de bøger, der er udlånt til personen.

f) Implementér metoden `udskrivUdlånsinfo` i klassen `Bog`. Metoden skal udskrive bogens forfatter, titel, eventuel låner samt en liste over de personer, der har reserveret bogen (se klassen `Customer` i biludlejningseksemplet).

g) Lav et lille testprogram, der demonstrerer, at biblioteksystemet fungerer efter hensigten. Vælg et passende interessant sted i testprogrammet og tegn objektmodellen, som den ser ud netop på dette sted i programmet.

Opgave 5.1.5

Denne opgave er en fortsættelse af (en del af) opgave 5.1.4 og tager udgangspunkt i projektet bibliotek-arv.

a) Lad klassen `Item` implementere Comparable-interfacet ved at implementere `compareTo`-metoden i `Item`. Metoden skal sammenligne to `Item`-objekter baseret på titel. (Hint: `String`-klassen implementerer

allerede `compareTo`; send Sorteper videre hertil. Sammenlign med projektet person-3).

b) Test at `compareTo` virker efter hensigten, også ved sammenligning af `Bog`- og `Video`-objekter.

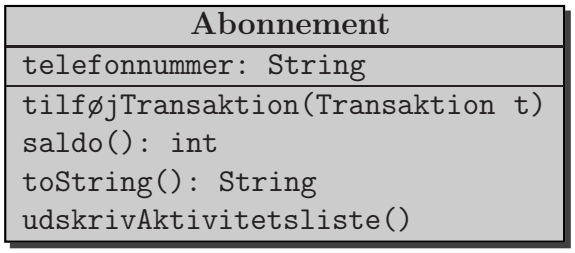
c) Implementér metoden `printEmne` i klassen `Person` (se javadoc-specifikationen for metoden i klassen).

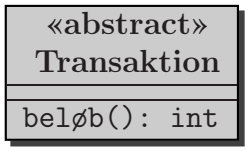
d) Udvid systemet med en klasse der repræsenterer en musik-CD. Test at klassen fungerer efter hensigten, f.eks. at det er muligt at sammenligne et `CD`-objekt med et andet `Item`-objekt.

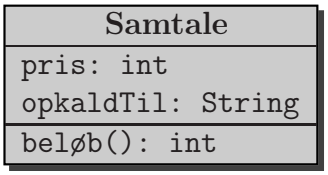
e) Udvid `Driver`-klassen til at omfatte `CD`-klassen ved at oprette et antal `CD`-objekter og udlåne (nogle af) disse til personerne i systemet.

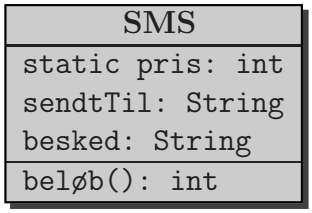
Opgave 5.1.6

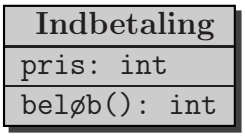
Denne opgave tager udgangspunkt i projektet talkmore, og går ud på at implementere systemet beskrevet i følgende UML-diagram.











a) Implementér klassen `Abonnement` med en passende konstruktør. Når man opretter et abonnement hos Talkmore får man indsat 200 kr. på sin konto; dette skal afspejles i konstruktøren for klassen. I første omgang skal du ikke bekymre dig om metoderne `udskrivAktivitetsliste` og `saldo`.

b) Implementér klasserne `Samtale` og `SMS`. I klassen `SMS` er attributten `pris` erklæret static, hvorfor?

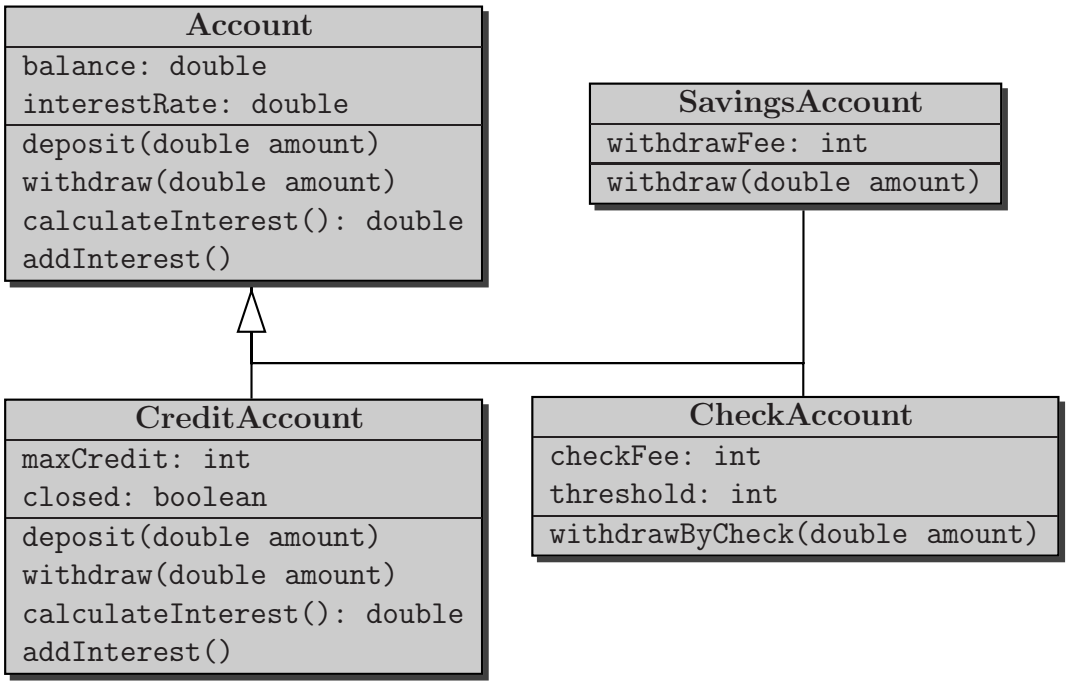
c) Implementér `toString`-metoder på samtlige klasser (undtagen den abstrakte klasse `Transaktion`).

d) Implementér metoden `saldo` i klassen `Abonnement`.

e) Implementér metoden `udskrivAktivitetsliste` i klassen `Abonnement`. Metoden skal udskrive en liste over samtlige transaktioner for abonnementet med én transaktion pr. linie.

f) Udvid `Driver`-klassens testmetode med passende JAVA-kode til test af programmet.

Opgave 5.1.7
Denne opgave tager udgangspunkt i projektet kontohierarki, og går ud på at implementere systemet beskrevet i følgende UML-diagram.



a) Implementér klassen `SavingsAccount`. En opsparingskonto er en konto hvor der kun kan hæves mod betaling af et gebyr på kr. 100. Gebyret fratrækkes kontoens saldo ifm. hævning. Lav javadoc for de metoder der tilføjes/redefineres.

b) Implementér klassen `CheckAccount` inklusiv javadoc. En checkkonto er en konto hvor man man også kan hæve penge via checks; dette modelleres med metoden `withdrawByCheck`. Når der hæves via check, pålægges et gebyr på kr. 10,- som fratrækkes saldoen på kontoen. Det er dog gratis at benytte check hvis saldoen er over en bestemt tærskel. Normalt er denne tærskel kr. 10.000, men der kan træffes individuel aftale herom. Lav javadoc for de metoder der tilføjes/redefineres.

c) Lav en `Driver`-klasse der illustrerer virkemåden af de forskellige kontoklasser.

Opgave 5.1.8
Denne opgave tager udgangspunkt i projektet filsystem.

a) Implementér en metode, `size`, der returnerer størrelsen af et filsystemelement (fil eller katalog). For filer skal metoden blot returnere størrelsen af filen; for kataloger skal metoden returnere summen af størrelsen af elementerne i kataloget.

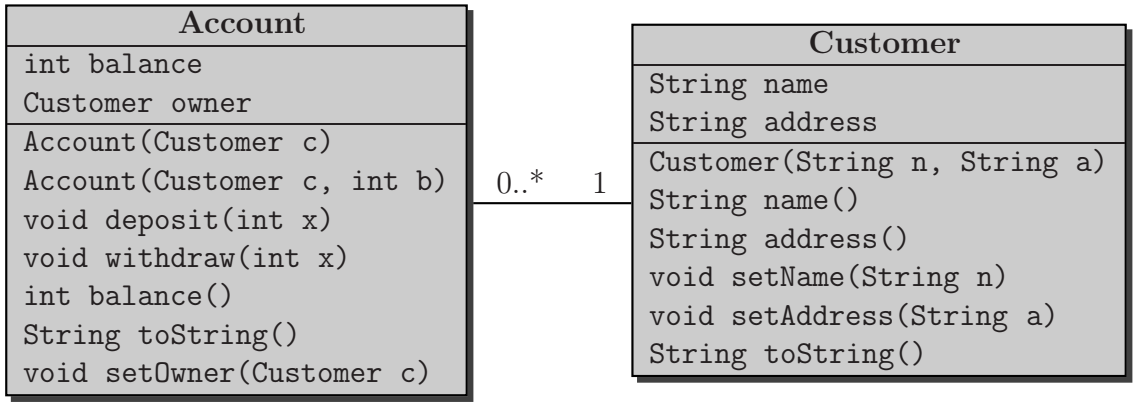
Hint: Metoden kan implementeres i superklassen `FileSystemElement` og redefineres i subklassen `Directory`; eller den kan erklæres abstract i superklassen og (re-)defineres i begge subklasser.

5.2 Implementering af klassemodeller

Opgave 5.2.1
I denne opgave skal du modellere (en lille med vigtig del af) et system til en bank. I en bank er der blandt andet *konti* (`Account`) og *kunder* (`Customer`). I denne bank er alle konti ejet af præcis én kunde, og det er muligt for en kunde (kortvarigt) ikke at eje nogle konti.

- Det skal selvfølgelig være muligt at indbetale penge på og hæve penge fra en konto (mutatorer). Det skal også være muligt at få oplyst saldoen på en konto (accessor).
- For kunder skal det være muligt at ændre deres navn og adresse uafhængigt.
- Når en ny konto åbnes skal det specificeres hvem ejeren er (husk, at en konto ikke kan eksistere uden en ejer). Det er derimod valgfrit at opgive en startsaldo; hvis ingen startsaldo bliver givet, bliver den sat til nul. (Der skal altså være to konstruktører: én, der tager et argument af typen `Customer` og én, der tager to argumenter, en af type `Customer` og en af typen `int`).
- Når en kunde oprettes bliver navnet og adressen opgivet, ikke andet.

Ovenstående specifikation kan udtrykkes i et UML klassediagram som følger:



Implementér ovenstående system trinvist, d.v.s. implementer de simple dele først, test dem, og fortsæt med at implementere de mere avancerede dele af systemet. Vi foreslår den følgende rækkefølge:

a) Implementér kontruktøren og de to accessor-metoder `getAddress` og `getName` i `Customer`-klassen.

b) Implementér kontruktøren og accessor-metoden `getBalance` i `Account`-klassen.

c) Implementér `deposit` og `withdraw` i `Account`-klassen.

d) Implementér `setName` og `setAddress` i `Customer`-klassen.

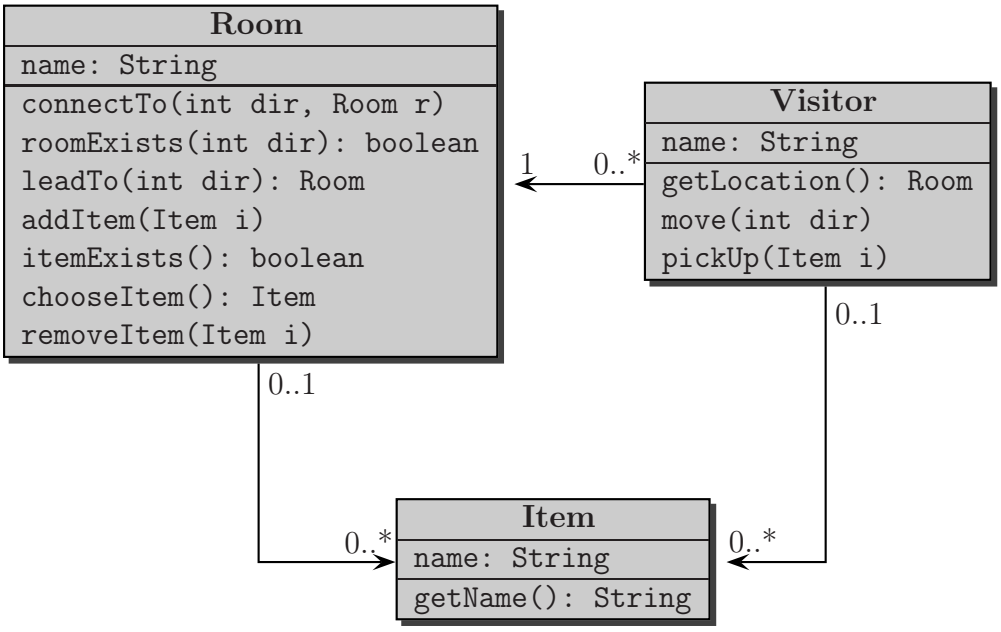
e) Implementér begge `toString`-metoder.

f) Implementér `setOwner`-metoden i `Account`-klassen.

g) Modellen skal repræsentere at en kunde kan eje flere konti. Tilføj to metoder til `Customer`-klassen til at tilføje og fjerne konti. (Hint: Brug en `Collection` internt i `Customer`-klassen til at opbevare referencerne til de konti, der er ejet af kunden.)

h) Modificér `setOwner`-metoden for at sikre, at objektmodellen er konsistent: hvis en konto er ejet af en kunde, vil den kunde, og kun den kunde, have kontoen registreret som hans/hendes. (Hint: Lad `setOwner`-metoden kalde `remove`-metoden hos den gamle ejer og `add`-metoden hos den nye ejer. Overvej om der er andre steder i Account-klassen der skal ændres for at sikre konsistens af objektmodellen, og foretag i givet fald disse rettelser.)

Opgave 5.2.2
I denne opgave skal du skrive et program, der modellerer rum, der er forbundne til hinanden i retningerne EAST, NORTH, WEST og SOUTH. Et antal besøgende kan bevæge sig omkring i rummene. Rummene og de besøgende kan have ting og en ting er enten i et rum eller hos en person.



Vi foreslår at du implementerer systemet i følgende rækkefølge:

a) Hent det delvist implementerede system.

b) Implementér klassen `Room` og test den interaktivt i BlueJ.

c) Implementér den del af konfigurationen i test-driveren, der har at gøre med opsætning af rum (der er et lille eksempel i den udleverede kode).

d) Implementér klassen `Visitor` og test den interaktivt i BlueJ.

e) Implementér den del af konfigurationen i test-driveren, der håndterer oprettelsen af besøgende (husk, at en ny besøgende skal placeres i et rum, d.v.s. konstruktøren i `Visitor` tager et rum som parameter).

f) Implementér den del af programmet, der har at gøre med at samle ting op, i test-driveren.

g) Tilføj en metode i `Visitor`-klassen, der gør det muligt for en besøgende at smide en ting i rummet, hvor den besøgende befinder sig i øjeblikket.

h) Implementér `toString`-metoder i alle klasser.

Hvis du undervejs føler behov for at ændre signaturen af metoder (eller tilføje nye metoder) må du gerne gøre det. Husk blot, at gøre det eksplicit i specifikationen af metoden.

Til sidst, modificér programmet, så alle metoder skriver en besked, der tillader dig at overvåge opførslen af programmet. Udskrifterne skal være noget i stil med (pånær kommentarerne til højre):

```
John moves to Studio 1              // printed by move
John picks up Apple in Studio 1     // printed by pickUp
Paul moves to Studio 2              // printed by move
Paul picks up Bass in Studio 2      // printed by pickUp
...
```

6 Programdesign

6.1 Interfaces

Opgave 6.1.1
En multimængde er som bekendt en konstruktion, der kan indeholde mere end et eksemplar af et bestemt element (i modsætning til almindelige mængder). Ved indsættelse af 7 i multimængden $[2, 1, 1, 7, 4]$ fås den nye multimængde $[2, 1, 1, 7, 4, 7]$ med to forekomster af 7. Ligesom man for sædvanlige mængder kan spørge om et element ligger i mængden, så kan man for multimængder spørge om antal forekomster af et bestemt element. I det foregående eksempel er der således to forekomster af elementet 1.

I denne opgave betragtes en datatype for multimængder af tegn, hvor datatypens værdi er:

- en multimængde af tegn,

og datatypens operationer kan:

- konstruere den tomme multimængde,
- indsætte et tegn i multimængden,
- tælle antal forekomster af et bestemt tegn.

a) Realisér datatypen ved at skrive en klasse, der implementerer følgende interface:

```
/**
 * Multi definerer en datatype, hvis værdi er en
 * multimængde af tegn med en række tilhørende operationer
 */
public interface Multi
{
    /**
     * makeEmpty bevirker at multimængdens værdi
     * bliver den tomme mængde
     */
    public void makeEmpty();

    /**
     * insert tilføjer et tegn til multimængden
     * @param c Et tegn der skal indsættes i multimængden
     */
    public void insert(char c);

    /**
     * count returnerer antal forekomster
     * af et tegn i multimængden
     * @param c Et tegn
     * @return antal forekomster af "c" i multimængden
     */
    public int count(char c);
}
```

b) Brug klassen fra a) til at lave en frekvensanalyse af en tekst, det vil sige en tabel over de indgående bogstaver og deres frekvenser.

Opgave 6.1.2
Man kan sortere et array af heltal ved at benytte $\leq$-operatoren på tallene. Følgende interface (fra pakken `java.lang`) generaliserer $\leq$-operatoren:

```
public interface Comparable
{
    /**
     * compareTo
     * @return a negative integer, zero, or a positive integer
     * as this object is less than, equal to, or greater than
```

```
     * the specified object.
     */
    public int compareTo(Object o);
}
```

Her er en sorteringsalgoritme `selectionSort`, der kan sortere et array af type `C[]` for enhver klasse `C`, der implementerer `Comparable` (og det gør f.eks. `String` og `Integer`):

```
public class SortModel
{
    public void selectionSort(Comparable[] l)
    {
        int q = 0;
        while(q < l.length) {
            int m = findLeast(l,q,l.length);
            Comparable temp = l[q]; l[q] = l[m]; l[m] = temp;
            q++;
        }
    }

    private int findLeast(Comparable[] r, int i1, int i2)
    {
        int min = i1;
        int q = i1 + 1;
        while(q < i2) {
            if(r[q].compareTo(r[min]) < 0) { min = q; }
            q++;
        }
        return min;
    }
}
```

a) Nedenstående klasse `Rational` tilbyder en meget simpel repræsentation af rationale tal uden aritmetiske operationer. Udvid klassen så den implementerer interface `Comparable`:

```
public class Rational
{
    private int num;
    private int den;

    public Rational(int n, int d)
    {
        if(d == 0) {
            throw new RuntimeException(
              "RationalError: zero denominator");
        }
        else { num = n; den = d; }
    }

    public String toString() { return (num+"/"+den); }
}
```

b) Brug den udvidede klasse til at skrive et program, der indlæser en række rationale tal og derefter udskriver dem i sorteret orden. Kør programmet på de første kædebrøksapproksimationer for π:

$$\left(\frac{3}{1}, \frac{22}{7}, \frac{333}{106}, \frac{355}{113}, \frac{103993}{33102}, \frac{104348}{33215}, \frac{208341}{66317}\right)$$

c) Udvid klassen, der repræsenterer lange heltal (fra opgave 4.1.2), så den ligeledes implementerer interfacet `Comparable`.

d) Overvej hvorledes du kan konstruere en klasse, der repræsenterer vilkårlige rationale tal (d.v.s. rationale tal, hvor tæller og nævner kan være lange heltal). Klassen skal som minimum implementere interface `Comparable`. I hvilket omfang kan du genbruge/udvide klasser som du allerede har lavet?

Opgave 6.1.3
Betragt følgende JAVA-program, der anvender en ukendt klasse `C`.

```
public class A
{
    private static C c;
```

```
    public static Object center(int n)
    {
        if(n == 0) {
            Object a = c.getObject();
            return a;
        }
        else {
            Object[] d = (Object[])center(n - 1);
            return d[d.length/2];
        }
    }

    public static void main(String[] args)
    {
        int n = (new Integer(args[0])).intValue();
        c = new C();
        c.setDepth(n);
        c.printObject(center(n));
    }
}
```

a) Hvilke metoder skal klasse `C` indeholde for at klasse `A` kan oversættes korrekt?

b) Hvad laver metoden `center`?

c) Er det principielt muligt at skrive en klasse `C`, så kørsel af programmet i klasse `A` (med et vilkårligt ikke-negativt heltalsargument på kommandolinjen) undgår at standse i utide på grund af en Exception?

6.2 Klassehierarkier

Opgave 6.2.1
Denne opgave går ud på at opnå genbrug og simplificering ved at opbygge et klassehieraki. En bank har 3 kontoformer, der hidtil er beskrevet ved uafhængige klasser:

- `Basiskonto` med metoder:
  - `insert(n)`: indsæt n kroner på kontoen
  - `withdraw(n)`: hæv n kroner fra kontoen (hvis saldoen er stor nok)

  - `getAmount`: returner saldoen

- `Opsparingskonto` med metoder:
  - `insert(n)`: indsæt n kroner på kontoen
  - `withdraw(n)`: hæv n kroner fra kontoen (hvis saldoen er stor nok)
  - `computeInterest`: beregn årlig rente (2% af saldoen)
  - `getAmount`: returner saldoen
- `Kassekredit` med metoder:
  - `insert(n)`: indsæt n kroner på kontoen
  - `withdraw(n)`: hæv n kroner fra kontoen
  - `computeInterest`: beregn årlig rente (4% af positiv saldo; 10% af negativ saldo)
  - `getAmount`: returner saldoen

Ved oprettelse af en konto er saldoen altid 0 kroner. Til bogholderi og beregning af samlet årlig renteindtægt/udgift anvender banken følgende programstump:

```
public class Bank
{
    private Object[] accounts;

    public Bank()
    {
        accounts = new Object[3];
        accounts[0] = new Basiskonto();
        ((Basiskonto)accounts[0]).insert(100);
        accounts[1] = new Opsparingskonto();
        ((Opsparingskonto)accounts[1]).insert(1000);
        accounts[2] = new Kassekredit();
        ((Kassekredit)accounts[2]).withdraw(300);
    }

    public int computeInterest()
    {
        int sum = 0;
        for(int i = 0; i < accounts.length; i++) {
            if(accounts[i] instanceof Opsparingskonto) {
                sum += ((Opsparingskonto) accounts[i]).
                          ccomputeInterest();
            }
            if(accounts[i] instanceof Kassekredit) {
                sum += ((Kassekredit) accounts[i]).
                          computeInterest();
            }
        }
        return sum;
    }
}
```

a) Foreslå et klassehierarki af bankkonti, som dels reflekterer en naturlig kontostruktur, dels muliggør genbrug klasserne imellem, og dels muliggør simplificering af programkoden i `Bank`. Lav selv nye abstrakte klasser i det omfang det er relevant.

b) Skriv JAVA-koden for alle klasserne i dit hierarki og i den modificerede `Bank`.

c) Lav et lille JAVA-program der afprøver klasserne fra b) og udfør det.

7 Applikationer

7.1 Tekstbehandling

Opgave 7.1.1
Denne opgave går ud på at konstruere et orddelingsprogram efter en simpel regel: Et ord kan deles midt imellem hvert par af nabovokaler. Hvis der står et ulige antal konsonanter mellem to vokaler, så får den sidste flest konsonanter. Følgende eksempler viser anvendelsen af denne regel (der ikke altid giver det korrekte resultat):

tekst: tekst (kan ikke deles)
type: ty–pe
korrekt: kor–rekt
datalogi: da–ta–lo–gi
algoritme: al–go–rit–me
program: prog–ram (undtagelse)

Du kan med fordel benytte en speciel variant af `StringTokenizer`, der ikke springer over skilletegnene, men leverer dem en ad gangen som separate tokens. F.eks. vil programstumpen:

```
StringTokenizer t =
  new StringTokenizer("museumsangst","aeiouyæøåAEIOUYÆØÅ",true);
while(t.hasMoreTokens()) {
  System.out.print(t.nextToken() + "  ");
}
```

udskrive følgende:

```
m  u  s  e  u  ms  a  ngst
```

a) Skriv indmaden til følgende metode, der kan dele et enkelt ord (f.eks. skal `hyphenateWord("skrubbeangst")` returnere `"skrub-be-angst"`):

```
public String hyphenateWord(String w) { ... }
```

b) Lav et JAVA-program der anvender metoden fra a) i modelklassen, f.eks. et program der indlæser enkelte ord og udskriver dem med delestreger indsat.

c) Har du ideer til forbedring af orddelingsalgoritmen anvendt i a)?

Opgave 7.1.2
LIX-værdien (LæsbarhedsIndeX) for en tekst, der skal opfattes som et mål

for tekstens sværhedsgrad, er defineret som: (det gennemsnitlige antal ord pr. meningsenhed) + (den % af ordene, der indeholder mindst syv tegn).

a) Der skal skrives et program, der læser en fil og beregner LIX-værdien af teksten i filen.

   En del af opgaven er præcist at definere, hvad "ord" og "meningsenhed" betyder.

b) Beregner du de samme LIX-værdier som andre på dit øvelseshold?

Opgave 7.1.3
(repetition af rekursion)

Betragt følgende grammatik, der definerer en lille delmængde af sproget dansk:

Sætning	::=	**Substantiv Bisætning**$^{\bigcirc}$
		Prædikat Fortsættelse$^{\bigcirc}$
Fortsættelse	::=	**Konjunktion Sætning**
Bisætning	::=	, som **Substantiv Bisætning**$^{\bigcirc}$
		TransitivtVerbum, |
		, der **IntransitivtVerbum**, |
		, der **RefleksivtVerbum** sig,
Prædikat	::=	**VerbalLed Adverbial**$^{\bigcirc}$
VerbalLed	::=	**TransitivtVerbum Substantiv** |
		IntransitivtVerbum |
		RefleksivtVerbum sig
Konjunktion	::=	og | mens | hvorimod
Substantiv	::=	studenten | forelæseren | instruktoren |
		tutoren | Java | programmet
TransitivtVerbum	::=	dumpede | roste | underviste | oversatte
IntransitivtVerbum	::=	forsvandt | bestod | drak | gik ned
RefleksivtVerbum	::=	kom | dummede | forberedte | morede
Adverbial	::=	ret så voldsomt | temmeligt hurtigt |
		ganske dårligt | helt perfekt

Notationen **Bisætning**$^{\bigcirc}$ betyder at kategorien **Bisætning** kan udelades.

a) Skriv et program, der genererer en tilfældig sætning defineret af ovenstående grammatik. Hint: Skriv en metode for hver kategori.

b) Argumentér for, at hvis grammatikken ændres til

Sætning	::=	**Substantiv Bisætning**$^{\bigcirc}$ **Prædikat** |
		Sætning, **Konjunktion Sætning**

så vil den stadig definere de samme sætninger.

c) Vil dit program opføre sig anderledes, hvis det ændres i overensstemmelse med b)? I bekræftende fald hvordan og hvorfor?

Opgave 7.1.4
Pe-pe-sproget er en variant af dansk, hvis brug hjælper med til at gøre selv den simpleste talestrøm ganske uforståelig...

Man laver en almindelig sætning om til pe-pe-sprog ved at dele alle ord op i stavelser. I pe-pe-sproget gentager man alle stavelserne to gange, med den modifikation, at anden gang stavelsen siges, udskiftes de indledende konsonanter med et 'p'. Følgende er nogle simple eksempler:

"Java"	→	"Japavapa"
"Unix"	→	"UpUnixpix"
"Datalogi"	→	"Dapatapalopogipi"
"Ugeseddel"	→	"UpUgepesedpeddelpel"

a) Konstruér indmaden til følgende metode, der oversætter til pe-pe sprog (f.eks. skal metodekaldet `pepetizeWord("skrubbeangst")` returnere `"skrubpubbepeangstpangst"`).

```
public String pepetizeWord(String w) { ... }
```

Hint: du kan med fordel bruge `hyphenateWord` metoden fra opgave 7.1.1 som en hjælpemetode.

b) Lav et JAVA-program der anvender metoden fra a) i modelklassen, således at programmet kan indlæse enkeltord og udskrive dem oversat til pepesprog.

c) Skriv eventuelt et program, der oversætter den anden vej.

7.2 Talbehandling (lommeregnere)

Opgave 7.2.1

a) Oversæt og kør programmet fra eksempel-kataloget. Vær sikker på, at du forstår strukturen og opførslen af programmet.

b) Implementér en stak baseret på en `Collection` (mere specifikt: baseret på en `List`) ved at udfylde hullerne i klassen `CollectionStack`.

c) Test din stak fra b) ved at modificere `Driver`-klassen til at benytte din stak i stedet for en kædet stak.

d) Implementér en stak baseret på et array ved at udfylde de manglende huller i klassen `ArrayStack`.

e) Igen, test din løsning ved at lave passende ændringer i `Driver`-klassen.

f) Modificér `Calculator`-programmet, så det i stedet benytter den indbyggede `Stack` i JAVA. Hvilke ændringer er nødvendigt for at gøre dette?

(Hint: Den indbyggede `Stack` i JAVA er en generisk stak, der tager parameter og returnerer værdier af typen `Object`. Derfor er det nødvendigt at bruge wrapper-klassen `Integer` i stedet for den simple type `int`. Kig i JAVA dokumentationen for yderligere information.)

Opgave 7.2.2

a) Udvid lommeregneren fra opgave 7.2.1 så den understøtter flere operationer, f.eks.:

- % (modulus)
- ! (fakultet)
- | (potens, også kendt som ^)
- # (kvadratrod)

Vær opmærksom på, at nogle af operatorerne er unære (d.v.s. de tager kun en operand).

Det følgende er en forklaring på semantikken af potens:

```
2 4 |      =   16         (2 i fjerde)
2 3 4 + |  =  128        (2 i syvende)
2 3 4 | +  =   83   (2 + (3 i fjerde))
```

b) Find flere operatorer og implementer dem.

c) Hvad kræves det for lommeregneren for at kunne understøtte floating-point tal og ikke bare heltal?

7.3 Billedbehandling

Opgave 7.3.1

Denne opgave tager udgangspunkt i projektet `Billedbehandling`, der ligger i eksempel-kataloget. Implementér nedenstående billedoperationer (ud-

vid klassen `PictureDriver`). Opgaverne står i stigende sværhedsgrad, og de sidste er forholdsvist komplicerede.

a) `darkenPicture`: gør billedet lidt mørkere.

b) `mirror`: spejlvend billedet omkring den vertikale midterakse.

c) `flip`: spejlvend billedet omkring den horisontale midterakse.

d) `thresholding1`: sæt alle pixels der højst har tærskelværdien til 0 og alle pixels større end tærskelværdien til 255. Parameter: tærskelværdien.

e) `thresholding2`: sæt hver pixel til sort (0), grå (127) eller hvid (255).

f) `border`: sæt en sort ramme omkring billedet. Parameter: rammens bredde. (Det gør ikke noget at den yderste kant af billedet ødelægges af rammen.)

g) `rotateL`: rotér billedet 90 grader (90 grader mod uret, venstre om). Hint: Opret med metoden `createEmptyPicture` et nyt billede med en bredde der svarer til det oprindelige billedes højde, og en højde der svarer til det oprindelige billedes bredde (og en tilfældig farve til alle pixels i billedet). Kopier derefter pixels fra det oprindelige billede til det nye. Kunsten er naturligvis at finde ud af hvor hvert enkelt pixel skal flyttes hen: hvis (x, y) er koordinater i det nye "væltede" billede, så svarer de til punktet $(\texttt{width} - y - 1, x)$ i det oprindelige billede (`width` betegner her bredden af det oprindelige billede).

h) `rotateR`: rotér billedet -90 grader (90 grader med uret, højre om).

i) `smoothing`: erstat hvert pixel med gennemsnitsværdien i det 3×3 kvadrat det er centrum i.

j) `oilPainting`: erstat hvert pixel med den hyppigst forekommende værdi i det $n \times n$ kvadrat det er centrum i. Parameter: størrelsen på kvadratet.

Opgave 7.3.2
Denne opgave tager udgangspunkt i projektet `Billedbehandling` fra opgave 7.3.1. Implementér følgende funktioner på klassen `PictureDriver`:

a) En funktion der med udgangspunkt i en pixel kan beregne størrelsen af et ensfarvet område i et billede (antallet af pixels der er sammenhængende med den udpegede og har samme farve som denne). Hint: Lav en kopi af billedet hvor de pixels der er talt med farves i en anden farve for at undgå uendelig rekursion.

b) En udfyldningsfunktion således at et felt, og alle felter af samme farve sammenhængende hermed, får en ny farve.

Opgave 7.3.3

Denne opgave tager udgangspunkt i projektet `Billedbehandling` fra opgave 7.3.1. Implementér en undo-funktionalitet på klassen `PictureDriver` således at en sekvens af billedoperationer kan fortrydes.