IMADA

Programmering A

Noter og opgaver

Niels Kjeldsen &
Jacob Aae Mikkelsen

Med grundig korrektur og rettelser
af Edmund Christiansen


UNIVERSITY OF
SOUTHERN DENMARK

Indhold

1 Indledning

Denne notesamling er blevet til efter mange grædte tårer fra studerende, som ikke synes programmering er let. Vi har her forsøgt at samle eksempler på den grundlæggende syntax brugt i JAVA, sammen med opgaver der specielt retter sig mod studerende der aldrig har programmeret før eller prøvet styresystemet Linux.

Noterne er ikke ment som en lærebog, men som en starthjælp.

Der er endvidere stillet opgaver i at omsætte pseudokode til JAVA, hvor pseudokoden er skrevet som i Cormen et. al.[2], brugt i DM507.

Opgaverne med en stjerne er lidt sværere end de andre, men absolut ikke uløselige. Gør dig selv en tjeneste, og gør i det mindste et forsøg på at løse dem.

Stor tak til Balkonens øvrige beboere for råd, vejledning og gode diskussioner.

2 Kom i gang med programmering

Første del af dette afsnit er meget specifikt for systemet på IMADA. Den anden del er for lige at få dig igang, og introduceret til nogle begreber indenfor programmering. Hvis du allerede er godt bekendt med IMADAs terminalrum og har programmeret før, kan du sikkert roligt springe dette afsnit over.

2.1 Lær IMADA systemet at kende

IMADAs computere er udstyret med styresystemet Ubuntu, der er en af mange Linux distributioner. Den grafiske brugerflade er er som standart sat til Gnome.

De nedenstående punkter er ment som en kort introduktion til IMADA systemet.

1. Log på systemet.

2. For at skifte dit password skal du gå op i menuen `Applications -> Accessories -> Terminal`
   Det starter en shell/terminalprompt, som er stedet alt kan laves i Linux. Fortvivl ikke, meget kan laves på andre måder!
   I din shell skriver du `passwdch` . Så skal du skrive dit udleverede password 2 gange og derefter dit nye[1] password to gange. Du har nu skiftet password på din IMADA konto

3. Hvis dit brugernavn er `dummy07` har du nu en imada-mail addresse som `dummy07@imada.sdu.dk`. Hvis du ikke har tænkt dig at bruge den som primær mail addresse, kan du få den til at videresende til din anden mail ved at gøre følgende:
   Start en tekst editor: `Applications -> Accessories -> Text Editor`
   skriv den mail addresse du bruger på den øverste linie i filen.
   Gem filen som `.forward` i dit hjemmebibliotek[2].

4. Du har også fået en hjemmeside på imada! Hvis dit brugernavn er `dummy07`, så har din hjemmeside adressen `www.imada.sdu.dk/~dummy07`. De filer du gemmer i biblioteket WWWpublic er tilgængelige på nettet[3].

5. Opret en mappe til faget DM502, ved først at vælge `Places -> Home Folder`, højreklikke i vinduet File Browser (Dokumenter) og vælg `Create Folder`. Giv den navnet DM502. Dobbeltklik på mappen, og lav en ny mappe ved navn uge1. På denne måde har du mulighed for at finde dine filer igen!

6. Start en shell (se punkt 2), og skriv `ls` Det står for *list* og du skulle gerne få listet filer og biblioteker i dit hjemmekatalog.

[1] Dit password skal være mindst 8 tegn langt og bestå af både store og små bogstaver, tegn og tal.

[2] Dit hjemmebibliotek er den mappe der hedder det samme som dit brugernavn, og er der shell'en starter fra.

[3] Hvis vel og mærke rettighederne er så andre kan læse dem, se om rettigheder i appendix A.1

skriv `cd DM502` (cd er en forkortelse for *change directory*). Skriv `ls`, og du skulle gerne se mappen uge1.
`cd uge1` skulle gerne bringe dig ind i mappen uge1, og `ls` skulle gerne afsløre at den er tom (endnu).
`cd ..` bringer dig tilbage i filtræet.

7. Du logger ud igen ved at klikke på den den røde knap med den hvide cirkel, øverst i højre hjørne, hvorefter du vælger `Log out`

Opgave 1: Indledende øvelser
Læs og udfør de ovenstående punkter

2.2 Hello world!

Hvad er nu det for en overskrift?

Hello World! er det første eksempel der i et vilkårligt programmeringssprog stiftes bekendskab med, her ingen undtagelse!

Programmet ser således ud:

```
/**
 * The very basic example of JAVA: Hello World
 */
public class HelloWorld
{
    public static void main(String[] args)
    {
        System.out.println("Hello World!");
    }
}
```

Det ser ret uoverskueligt ud, men fortvivl ikke. Det meste af arbejdet udføres af linien:

```
System.out.println("Hello World!");
```

Det er den linie der skriver `Hello World!` ud på skærmen. Resten ser vi på hen af vejen.

Kompilering og kørsel af JAVA programmer

Filer med java kode skal gemmes med endelsen `.java`. For at kompilere koden bruges kommandoen `javac filnavn.java`, og for at køre det kompilerede program bruges kommandoen `java filnavn`

Opgave 2: Hello World!
Start en text editor (`Applications -> Accessories -> Text Editor`). Indtast 'Hello World' programmet fuldstændigt som det står ovenfor. Gem filen i biblioteket *uge1*, (som i punkt 6 ovenfor) som 'HelloWorld.java'.
Start en shell, og gå ind i mappen uge1. I din shell skriver du nu: `javac HelloWorld.java`. Programmet er nu blevet oversat til kode, som computeren

kan læse. Kør programmet ved at skrive `java HelloWorld`, og glæd dig over at have kompilet og kørt dit første program (Det skulle gerne skrive HelloWorld! ud på skærmen til dig).

Opgave 3: Hello Mette
Modificer programmet Hello World, således at det skriver "Hello Mette!" på skærmen. Erstat "Mette" med dit eget navn.

2.3 Udskrift af kode

Udskrift af kode kan gøres fra din tekst editor, men et pænere format opnåes ved at bruge kommandoen `a2ps` i en shell. Det sker ved at finde det rigtige bibliotek (der hvor filen ligger) i en shell. derefter skriver du:

```
a2ps -Pd3 <filnavn.java>
```

a2ps er en forkortelse for 'Anything to PostScript'. og `-Pd3` angiver at den skal sende resultatet til printeren d3, som er den printer der står lige udenfor terminalrummet.

Opgave 4: a2ps
Udskriv Hello World programmet ved brug af a2ps.

3 JAVA syntax

I dette afsnit gennemgåes de mest brugte syntaktiske konstruktioner i JAVA. For hvert afsnit er stillet simple opgaver.

3.1 Kommentarer

Der er flere måder at skrive kommentarer i JAVA på. Hvis det er en enkelt linie er det ud sådan her:

```
// Jeg er en kommentarlinie på grund af de to skrå streger!
```

Hvis det er flere liniers kommentarer, ser det således ud:

```
/*
    Alt hvad der står mellem den skrå streg
    efterfulgt af en stjerne,
    og indtil en stjerne efterfulgt af en skrå streg
    er kommentarer.
    Kode kan også udkommenteres, fx:
    System.out.println("Jeg udføres ikke");
*/
```

Der er god kodestil (se appendix A.3) at skrive kommentarer til alle metoder (se afsnit 3.7) og klasser (afsnit 3.17).

Kommentarer kan hjælpe både dig selv og andre med at forstå hvad koden gør (eller bør gøre).

3.2 Typer

I JAVA er der følgende primitive typer[4]:

int	Heltal
char	Et enkelt bogstav eller tegn
double	Decimaltal (punktum bruges som komma)
boolean	Sandt eller falsk

String

Ud over de primitive typer bruges ofte typen `String` der er implementationen af tekststrenge i JAVA. Følgende er en tekststreng: `"Jeg er en tekststreng"` String er altid med gåseøjne, hvorimod char er med ping'er: `'a'`
Der er meget mere om strenge senere, specielt i afsnit 4.1

[4]Der er flere, læs mere i appendix A.2, men det er blot variationer af dem der præsenteres her

3.3 Variable

Variabler kan ses som beholdere der kan indeholde information. Variabler skal erklæres, det vil sige de skal vide hvilken type information de kan indeholde. Almindelige variabler starter med et lille bogstav.

```
int yearOfBirth;
```

Her kan variablen `yearOfBirth` nu indeholde et heltal, og den værdi kan vi tildele med et enkelt lighedstegn:

```
yearOfBirth = 1954;
```

`yearOfBirth` variablen indeholder nu værdien 1954, efter de to liniers kode.

De to linier kan samles i én således:

```
int yearOfBirth = 1954;
```

Hvis variablen er en konstant, der ikke ændres i løbet af programmets afvikling, kan det erklæres ved at erklære en variabel final. Konstanter skrives med store bogstaver:

```
final double PI = 3.14;
final int TIMER = 24;
```

Opgave 5: Gennemskuelse af et simpelt program
Hvad udskriver det nedenstående program, hvis du kører det?

```
// Klasse til en opgave i Programmering
public class Opgave
{
    //Main metode
    public static void main(String[] args)
    {
        int yearOfBirth;
        yearOfBirth = 54;
        System.out.println(yearOfBirth);
        yearOfBirth = 87;
        System.out.println(yearOfBirth);
    }
}
```

Opgave 6: * Lovlige tildelinger af variabler
Hvilke af de nedenstående tildelinger til variabler er tilladte, og hvilke giver fejl.

1. `int alder = 2;`
2. `int nyAlder = 2.5;`
3. `char bogstav = 'a';`

4. `char niveau = '2';`
5. `char bogstav = 4;`
6. `char karakter = '42';`
7. `double rentesats = 2.24;`
8. `double rente = 2;`

3.4 Aritmetiske operationer

I JAVA er der følgende simple aritmetiske operationer:

+	Addition
-	Subtraktion
*	Multiplikation
/	Division
%	Modulus (rest ved heltalsdivision)

I det følgende:

```
int antal = 14 / 4;
int rest = 14 % 4;
```

bliver `antal` sat til 3 og `rest` sat til 2.

Casting

Har du f.eks to `int`'s, men gerne vil have resultatet ud som et kommatal, bliver du nødt til at fortælle JAVA at det er det du vil. Dette hedder *casting*. Det foregår ved at man i parantes skriver den type man gerne vil have resultatet som foran beregningen. Eksempelvis:

```
double antal = (double) 14 / 4;
```

Nu bliver `antal` sat til 3.5. Havde der ikke været casting ville `antal` være blevet sat til 3.0, fordi resultatet af højre side er division af heltal, og derfor selv et heltal. Heltallet vil så blive omdannet til en double, og derfor giver det 3.0.
I dette tilfælde kunne det være lavet ved at skrive 14.0, da dette laver det til en beregning $\frac{double}{int}$, der giver en double som resultat.

Præcedens

Præcedens af operationerne er som i matematik almindeligt, så `2 + 3 * 4` fortolkes som `2 + (3 * 4)`

`++`

Plus tegnet `+` har flere betydninger, alt efter i hvilken sammenhæng det står. I `42+10` angiver det addition af to heltal, hvorimod i udtrykket:
`"baseball" + "bat"`
angiver det at de to strenge bliver konkateneret (sammensat) og giver strengen `"baseballbat"`.

Mange gange i programmering bruges at addere med én. En kort måde at skrive det på: `++`. De følgende to linier gør derfor det samme:

```
i = i + 1;
i++;
```

3.5 Logiske operationer

Hvis du skal lave et klimastyringsanlæg, er det fornuftigt at definere to variabler:

```
final int MINTEMP = 19;
final int MAXTEMP = 25;
```

Hvis man skal teste om sin stuetemperatur overskridder disse grænser, kræver det to check:

```
stueTemp >= MINTEMP
stueTemp <= MAXTEMP
```

Hvis de skal bindes sammen i en linie, kan det gøres ved en AND operation, der i JAVA er `&&`. Det kan samles i:

```
(stueTemp >= MINTEMP) && (stueTemp <= MAXTEMP)
```

Tilsvarende er der OR, der i JAVA er `||`
Parenteserne kunne udelades, men det er lettere at læse og forstå hvis de er der.

`&&`	AND
`||`	OR

3.6 Input fra tastaturet

For at lave lidt mere spændende eksempler, ser vi kort på hvordan man kan få input fra tastaturet her. Der er nok dele som virker underlige, fordi de først bliver forklaret senere, bær venligst over med det.

Vi bruger det der hedder en Scanner, som er en datatype andre har været venlige at skrive til os. For at få lov til at bruge en Scanner, skal vi først fortælle JAVA kompileren at den må bruge den, og hvor den er.
Det gøres ved linien:
`import java.util.Scanner;`
der skal stå øverst i koden. Nu kan vi oprette en variabel af typen Scanner med linien:
`Scanner sc = new Scanner(System.in);`
Når vi i parantesen skriver `System.in`, betyder det tastaturet. Vi kan nu kalde

en metode (mere i afsnit 3.7) der hører til variablen `sc` nemlig `nextLine()`. Det gøres i linien `String tekst = sc.nextLine();`. Her sættes `tekst` til at være den linie du skriver til programmet på kommandolinien, når den beder om det.

Nedenstående er programmet, der afsluttes med at udskrive det du har skrevet.

```
import java.util.Scanner;
/*
 * Pretty dumb class, just echoes what you write
 */
public class Echo {
    /*
     * Main method of the class
     */
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);
        System.out.println("Skriv noget");
        String tekst = sc.nextLine();
        System.out.println(tekst);
    }
}
```

Opgave 7: Scanner ekko

Indtast Echo programmet, og prøv at køre det et par gange.
Modificer programmet til at spørge efter to strenge, og udskriv dem begge til sidst.

Opgave 8: Regnemaskine

Ved at bruge `int x = sc.nextInt()`, (hvor `sc` er oprettet som i eksemplet ovenfor) kan du få et heltal fra tastaturet.
Lav et program der beder om to tal, og udskriv både deres sum og produkt.

Opgave 9: 'Pænere' regnemaskine

Prøv at forbedre din lommeregner fra opgaven før til at skrive: `Summen af 4 og 7 er 11` , og tilsvarende med multiplikationen. Hvis du har gemt de to tal i `x` og `y`, prøv følgende måde:
`System.out.println("Summen af"+ x + "og "+ y + "er "+ x + y);`
Hvad går galt? Hvorfor? Kan du reparere det? *Hint: brug parenteser*

3.7 Metoder

Forestil dig at du skal bruge oplysninger fra brugeren, og hver gang efter han har indtastet noget skal han have at vide at det er sikkert, og hans indtastede oplysninger ikke misbruges. Første (ikke pæne) eksempel kunne se således ud:

```
import java.util.Scanner;
/*
 *  Demonstrationsklasse til at vise metoders berettigelse.
 */
```

```
public class NotLikeThis
{
    // Main method of the class
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);
        String navn;
        String alder;
        String vej;

        System.out.println("Du skal vide at dine oplysninger");
        System.out.println("ikke vil blive videregivet eller");
        System.out.println("solgt til andre.");
        System.out.println("Indtast navn");

        navn = sc.nextLine();

        System.out.println("Du skal vide at dine oplysninger");
        System.out.println("ikke vil blive videregivet eller");
        System.out.println("solgt til andre.");
        System.out.println("Indtast alder");

        alder = sc.nextLine();

        System.out.println("Du skal vide at dine oplysninger");
        System.out.println("ikke vil blive videregivet eller");
        System.out.println("solgt til andre.");
        System.out.println("Indtast vej");

        vej = sc.nextLine();

        // Osv ....
    }
}
```

Hvis der er steder i ens kode der gentages, er det nok en god idé at lave det som en metode i stedet for. En metode til at klare ovenstående problem kan se således ud:

```
private static void printSikkerhed()
{
        System.out.println("Du skal vide at dine oplysninger");
        System.out.println("ikke vil blive videregivet eller");
        System.out.println("solgt til andre.");
}
```

Der er en del nye ord, og lad os så på nogen af dem:

private At erklære en metode private, betyder at den kun kan bruges fra sin egen klasse, mere om klasser i afsnit 3.17. Hvis en metode skal kunne bruges fra en anden klasse, skal man bruge `public` i stedet for `private`.

static Bliver forklaret i afsnit 3.17, indtil videre er det blot nødvendigt at det står der.

void Enhver metode skal have en returtype. Det kunne være int, String eller andre returtyper. Når returtypen er void, betyder det at den ikke sender noget tilbage, og derfor ikke behøver et `return` statement.

printSikkerhed() Metodens navn, som bruges når man skal udføre metoden. De tomme parenteser betyder at metoden ikke har nogen parametre (variable sendt med).

Det vil få vores kode til at se således ud:

```
import java.util.Scanner;
/*
 *  Demonstrationsklasse til at vise metoders berettigelse.
 */
public class LikeThis
{
    // Main method of the class
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);
        String navn;
        String alder;
        String vej;

        printSikkerhed();
        System.out.println("Indtast navn");

        navn = sc.nextLine();

        printSikkerhed();
        System.out.println("Indtast alder");

        alder = sc.nextLine();

        printSikkerhed();
        System.out.println("Indtast vej");

        vej = sc.nextLine();

        // Osv ....
    } //end main
```

```
    // Metode der udskriver sikkerhedsoplysninger
    private static void printSikkerhed()
    {
        System.out.println("Du skal vide at dine oplysninger");
        System.out.println("ikke vil blive videregivet eller");
        System.out.println("solgt til andre.");
    } //end printSikkerhed
}
```

Koden bliver hermed kortere og mere overskuelig. Programmet starter fra `main` metoden, og når det når til linien `printSikkerhed();` springer den til metoden `printSikkerhed`, udfører det kode der står der (udskriver de tre linier), og vender så tilbage til `main` metoden og fortsætter til næste linie.

Lad os se igen på lidt af Hello World programmet, nemlig:

```
public static void main(String[] args)
```

Det er jo bare en metode der er defineret her! Præcis denne metode er her hvor JAVA er defineret til at starte med at læse koden, og udføre programmet. Den er public, så den kan bruges udenfor klassen, og parametren `String args[]` er et array med input, se mere i afsnit 3.15

I det næste eksempel ser vi på en metode til at beregne x^3.

```
// Metode til at beregne x i tredje
private static int iTredje(int x)
{
    int resultat;
    resultat = x*x*x;
    return resultat;
}
```

Det skal her bemærkes at metoden ikke har void som returtype, men int. Derfor skal metoden returnere en int, ved en return statement. Metoden kan så kaldes således: `iTredje(5)` eller hvis z er af typen int:`iTredje(z)`. Ønskes z sat i tredje, kan det gøres således: `z = iTredje(z)`. `z` bliver her sat til det som `iTredje(z)` returnerer, hvilket præcis er værdien `z*z*z`

Det ovenstående eksempel kan skrives kortere, ved at slå nogen af linierne sammen:

```
// Metode til at beregne x i tredje
private static int iTredje(int x)
{
    return x*x*x;
}
```

Opgave 10: Kodeforståelse
Læs de to `iTredje` metoder igennem, og forklar forskellen. Hvilken synes du er lettest at forstå?

Opgave 11: Hypotenusen i en trekant
For at finde kvadratroden af en double x, kan du skrive `x = Math.sqrt(x);`

Skriv en statisk metode der får kateterne fra en retvinklet trekant som parametre, og returnerer hypotenusens længde. Altså udfyld kode til metoden:
`public static double hypotenusen(double a , double b)`
Lav også en main metode der bruger `hypotenusen` og udskriver resultatet ved hjælp af `System.out.println`

3.8 Typer af variable og scope

Vi har nu set hvordan et program kan deles op i metoder der kan lave mindre dele for os. Vi vil her se lidt på hvad det betyder for de forskellige variable, hvor de er defineret.

Et program kan nu, meget abstrakt, se således ud:

```
public class KlassensNavn
{
    private static int variabel1;

    public static void main(String[] args) {
        int variabel2;
        programkode;
    }

    private static void metode1(int parameter3)
    {
        int variabel4;
    }
}
```

Der er en variabel udenfor alle metoderne: `variabel1`.Variable af denne type kaldes *instans variable*, og kan ses og bruges af alle metoderne. Altså er det lovligt i `metode1` at have koden: `variabel1 = 5;`.

`Variabel2` og `variabel4` til gengæld er oprettet indeni en metode, og kan derfor kun ses og bruges indeni sin egen metode. `variabel2` kan ikke bruges fra `metode1` og `variabel4` kan ikke bruges i `main` metoden. Variable defineret i metoder kaldes *lokale variable*.

Endeligt er der `parameter3`, der kaldes en *parameter*. Den har de samme egenskaber som lokale variable, at den kun kan ses og bruges indenfor den metode den er defineret i.

Reglerne for hvor variable kan ses kaldes scope.

Type af variable	**Kan ses (scope):**
instans variable	i alle metoder i filen (klassen)
lokale variabler	kun i sin egen metode
parametre	kun i sin egen metode

3.9 Sammenligning

Hvis det er simple typer (se afsnit 3.2) der sammenlignes, bruges følgende operationer:

`==`	Lig med
`!=`	Ikke lig med
`<`	Mindre end
`<=`	Mindre end eller lig med
`>`	Større end
`>=`	Større end eller lig med

Er det derimod objekter, f.eks strenge eller andet, skal metoden `.equals()` bruges[5]. Et eksempel er:

```
String eksempel = "Eksempel";
boolean ens = eksempel.equals("eksempel");
```

Her får variablen `ens` værdien `false`, da JAVA er 'case sensitive' altså tager højde for små og store bogstaver.

3.10 If/else

Det er ikke alt kode der skal udføres (hver gang). Noget kan afhænge af omstændighederne. Til dette bruges `if/else` konstruktionen.

```
if ( Sandt/Falsk betingelse) {
    Kode der udføres hvis betingelsen er sand
}
```

Og med else ser det således ud

```
if ( Sandt/Falsk betingelse) {
    Kode der udføres hvis betingelsen er sand
} else {
    Kode der udføres hvis betingelsen er falsk
}
```

Et eksempel er test af om et tal er lige eller ulige:

[5]Mange gange går det godt at bruge == på strenge og lignende, men JAVA garanterer det ikke, så det er ikke pæn kodestil at forsøge.

```
import java.util.Scanner;
/*
 * Program der tester om et tal er lige eller ulige
 */
public class EvenOdd
{
    // Main metode for programmet.
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);
        System.out.println("Indtast et tal");
        int tal = sc.nextInt();

        if( tal % 2 == 0) {
            System.out.println("Tallet er lige");
        } else {
            System.out.println("Tallet er ulige");
        }
    }
}
```

Opgave 12: Brug af if/else
Skriv et program der beder om to tal, og fortæller om det første eller det andet er det største.

Opgave 13: Mere brug af if/else
Skriv et program der beder om et tal, og afgør om det kan deles med 5. Hvis det ikke kan, så afgør om det kan deles af 3.
Hint: Man kan sætte flere if/else inden i hinanden.

3.11 While løkke

I en while løkke testes betingelsen først, og denne afgør om koden i løkken skal udføres. Koden i løkken bliver udført indtil betingelsen ikke længere evalueres til `true`.

```
while ( Sandt/Falsk betingelse) {
    Kode der udføres så længe betingelsen er sand
}
```

Det følgende eksempel lægger alle indtastede tal sammen, indtil du indtaster et 0 (nul). Så udskriver den resultatet.

```
import java.util.Scanner;

// Eksempel klasse til demonstration af while løkke
public class AddAll
{
    // Main metoden
    public static void main(String[] args)
```

```
    {
        Scanner sc = new Scanner(System.in);
        int total;
        int tal;

        total = 0;
        tal = sc.nextInt();

        while(tal > 0) {
            total = total + tal;
            tal = sc.nextInt();
        }
        System.out.println(total);
    }
}
```

Bemærk:
Variable defineret inden i en løkke har scope inden i løkken. De kan altså ikke ses når løkken er stoppet.

Opgave 14: Nedtælling
Lav et program der laver en nedtælling fra 20 til 1, og udskriver tallene. (Skal selvfølgeligt laves med en while løkke).

Opgave 15: Forskellen på while løkke og if/else statement
Beskriv med dine egne ord hvad forskellen er på en while løkke og en if/else statement og hvad de kan bruges til.

3.12 For løkke

```
for( statement1 ; betingelse ; statement2) {
    Kode der udføres så længe betingelsen er sand
}
```

Et eksempel der udskriver de første 10 kvadrattal ser således ud:

```
for( int i = 0; i < 10 ; i = i+1) {
    System.out.println(i*i);
}
```

Opgave 16: Brug af for løkke
Lav et simpelt program der udskriver de første 20 tal i 17 tabellen. (Fordi 17 tabellen er sej)

Opgave 17: Nedtælling igen
Lav igen et program der laver en nedtælling fra 20 til 1, og udskriver tallene, denne gang med en for løkke.
Er det pænere end med en while løkke? nemmere?

3.13 Do while løkke

Forskellen mellem while løkker og do-while løkker er at man i do-while løkken afgør om løkken skal fortsætte ved slutningen af løkken. Derfor udføres løkken altid mindst én gang.

```
do {
    System.out.println("Hej hej");
} while (true);
```

Her omskrevet eksemplet, der lægger tal sammen til du indtaster nul, og udskriver summen.

```
import java.util.Scanner;

// Eksempel klasse til demonstration af do while løkke
public class AddAll
{
    // Main metoden
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);
        int total;
        int tal;

        total = 0;
        do {
            tal = sc.nextInt();
            total = total + tal;
        } while(tal != 0);
        System.out.println(total);
    }
}
```

Opgave 18: Gæt et tal
Lav et program, der går ud på at gætte et tal. Tallet må gerne være en konstant (Du gider alligevel ikke lege med det mere end én gang). Brug en do-while løkke der bliver ved med at spørge om et gæt, indtil du gætter rigtigt.

Opgave 19: Gæt forskellige tal
Lav dit program om fra forrige opgave, til nu at spørge om hvilket tal der skal gættes som det første. Så udskriver 100 tomme linier, så modstanderen ikke kan se tallet længere, og så beder om at gætte tallet.

3.14 Switch statement

En switch statement ser således ud:

```
switch (expression) {
    case værdi1:
        kode hvis expression har værdi1;
        break;
    case værdi2:
        kode hvis expression har værdi2;
        break;
    case værdi3:
        kode hvis expression har værdi3;
        break;
    default:
        kode der udføres hvis ingen andre cases matcher;
        break;
}
```

Bemærk at der kan være vilkårligt mange case værdier. Det er vigtigt at huske break efter koden for hver case, da koden for den næste case ellers udføres efterfølgende. defalut værdien kan udelades, men hvis man ikke mener at det er nødvendigt kan man jo bruge den til en fejlmeddelse, der viser at man er ramt forbi de andre cases.

Her er et eksempel der tager udgangspunkt i karakterskalaen:

```
import java.util.Scanner;
/*
 *   Oplysninger om karaktererne i den nye karakterskala
 */
public class KarakterSkala
{
   // Main metode i programmet
   public static void main(String[] args)
   {
       Scanner sc = new Scanner(System.in);

       System.out.println("Indtast din karakter");
       int karakter = sc.nextInt();
       switch (karakter) {
       case -3:
          System.out.println("-3: Ringe præstation");
          break;
       case 0:
          System.out.println("00: Utilstrækkelig præstation");
          break;
       case 2:
          System.out.println("02: Tilstrækkelig præstation");
          break;
       case 4:
          System.out.println("4: Jævn præstation");
          break;
```

```
        case 7:
           System.out.println("7: God præstation");
           break;
        case 10:
           System.out.println("10: Fortrinlig præstation");
           break;
        case 12:
           System.out.println("12: Fremragende præstation");
           break;
        default:
           System.out.println("Det er ikke en karakter");
           break;
        }
   }
}
```

Opgave 20: Ugedagene
Lav et program (der bruger en switch) der spørger brugeren om et tal, og fortæller hvilken ugedag det er i ugen. (1 = mandag)

3.15 Arrays

Hvis nu man skal gemme regnmængderne for en uge i et program, kan det selvfølgeligt gøres på den følgende (ikke pæne) måde:

```
int mandagsRegn;
int tirsdagsRegn;
int onsdagsRegn;
osv...
```

Det kan lade sig gøre, men hvad nu hvis man senere vil gøre det over et helt år? Så bliver det meget lang kode!

I stedet for kan man vælge at benytte sig af et array (dansk: tabel). Syntaksen ser således ud:

```
int[] regn;
regn = new int[7];
```

Variablen `regn` indeholder nu en tabel med 7 pladser, der kan holde `int`'s. Skal man bruge det der står på en plads i array'et kan det gøres ved:
`int torsdagsregn = regn[3]`

Bemærk:
Arrays starter fra 0. Et array `p` med 3 pladser indeholder altså pladserne `p[0]`, `p[1]` og `p[2]`

Et typisk gennemløb af et array laves med en `for` løkke, her laves et array med syv tabellen:

```
int[] syvTabel;
syvTabel = new int[10];

for(int i = 0; i < syvTabel.length ; i = i + 1) {
    syvTabel[i] = i*7;
}
```

Bemærk at syvTabel.length giver arrayets størrelse, der er pæn stil at benytte i løkken, hvis man senere udvider arrayet, udvides løkken automatisk.

Faktisk har du set syntaksen for array mange gange! I linien:
`public static void main(String[] args)`
Her indeholder arrayet af strenge `args` de argumenter man giver programmet på kommandolinien. I følgende eksempel udskrives parametrene man giver programmet på kommandolinien, altså når man i en shell skriver:
`java Test Hej med dig`
udskriver programmet Hej med dig, med ét ord på hver linie.

```
/*
 * Test af kommandolinie input
 */
public class Test
{
    // Main metode for programmet
    public static void main(String[] args)
    {
        for(int i = 0; i < args.length ; i = i + 1) {
            System.out.println(args[i]);
        }
    }
}
```

> **Bemærk:**
> Et array har et fast antal pladser når det er oprettet. Hvis du ikke ved hvor mange pladser du skal bruge fra start kan ArrayList være en løsning. Vi ser på ArrayList i afsnit 4.2

Opgave 21: Metoden `swap`
Skriv en metode:
`private void swap(int[] array , int i, int j)`
Der tager et array og to tal som parametre, og bytter om på de to elementer gemt på pladserne for de to tal.
Skriv også et program der tester dette, ved at oprette et array på 10 elementer. Fylder det med tallene fra 1 til 10. Brug metoden på det, og udskriver elementerne i arrayet igen.

Opgave 22: Antal parametre
Lav et program der udskriver antallet af parametre du angiver på kommando-

linien, gerne på formen:
`Du har angivet 2 parametre`

Opgave 23: * Input fra kommandolinien
Modificer eksemplet Test til at skrive alle argumenter ud på samme linie (brug evt. plus til at konkatenere strenge med, og udskriv udenfor løkken. Husk mellemrum)

Opgave 24: ** Array og bruger interaktion
Lav et program der udfører følgende handlinger:

- Spørger brugeren om antal inputs.
- Opretter et int array af den angivne størrelse.
- Spørger brugeren om tal, til det ønskede antal er opnået, disse gemmes i arrayet.
- Spørger brugeren om en værdi: x.
- Gennemløber arrayet og udskriver de tal der er større end x.

3.16 Exceptions og try/catch

Hvis brugeren skal indtaste et tal, kan det jo gøres således:

```
Scanner sc = new Scanner(System.in);
System.out.println("Indtast venligst er heltal.");
int heltal = sc.nextInt();
```

Men hvad nu hvis brugeren indtaster ‘‘syv‘‘? Programmet kommer med en fejlmeddelelse:
`Exception in thread "main" java.util.InputMismatchException`
og stopper uventet. Brugeren kan altså ødelægge programmet blot ved en forkert indtastning! Det kan vi ikke tillade, altså skal vi have et redskab til at forbedre robustheden.

Det kan gøres med en try/catch konstruktion:

```
try{
    Kode der kan give fejl
}
catch( Exception e) {
    Kode hvis der har været fejl
}
```

Hvor `Exception` ifølge god kodestil skal være den specifikke fejl der kan opstå. Der kan være flere `catch` dele af konstruktionen hvis der er flere forskellige fejl der kan opstå.

I vores eksempel bliver det altså:

```
Scanner sc = new Scanner(System.in);
System.out.println("Indtast venligst er heltal.");
int heltal = 0;

try{
    heltal = sc.nextInt();
}
catch (InputMismatchException e) {
    System.out.println("Det indtastede er ikke et tal");
    System.out.println("Der bruges nul i stedet for!");
}
```

Hvor vi i toppen af filen har:

```
import java.util.InputMismatchException;
```

Opgave 25: Brug af try/catch
Find et par af dine gamle opgaver frem, der får input fra brugeren, og gør dem mere robuste ved hjælp af try/catch.

3.17 Klasser og objekter

Udfra det der er blevet præsenteret indtil nu er det muligt at skrive at skrive næsten alle programmer. Hvorfor skal vi så lære mere kan man spørge? Det skyldes, at det følgende gør mange ting meget meget lettere. I de programmer I har skrevet indtil nu har I (forhåbentlig) brugt kommentarer og metoder til at opdele programmet så dette blev mere overskueligt. Dog kan programmer kun til en vis grænse laves overskuelige med disse redskaber.

Indtil nu har vi kunnet repræsentere heltal, decimaltal, boolske værdier, tegn og tekststrenge. Nu skal vi se på en struktur som kan hjælpe os med at repræsentere mere komplekse ting. Tag for eksempel en bil, biler er stort set alle forskellige, dog har de en række egenskaber som beskriver dem: produktionsår, producent, farve, vægt, antal kørte kilometer osv. Hvis vi skal skrive et program hvor vi ønsker at repræsentere en bil er det disse egenskaber, som vi får programmet til at huske. Alt dette kan godt virke noget abstrakt, men husk på at dette blot er en lettere måde at gøre ting I teoretisk set allerede kan. I programmerings sammenhæng knytter vi to begreber til ovenstående: en klasse og et objekt. En klasse er den række egenskaber vi ønsker at huske (i vores eksempel det vi forstår ved en bil), og et objekt er så en specifik bil (f.eks. en sølvgrå Audi A4 der har kørt 104.238 kilometer).

Vi vælger altså nogle egenskaber som vi gerne vil huske. I det første program ønsker vi at vide tre ting om en bil, nemlig farven, vægten og producenten. I kan allerede godt få et program til at huske en farve, en vægt og en producent. Farven og producenten kan gemmes som tekststrenge og vægten som et heltal. Eksempelvis ved:

```
    //Repræsentation af en bil
    String producent;
```

```
String farve;
int vaegt; //Angives i kg

producent = "Skoda";
farve = "Rød";
vaegt = 1100;
```

Vi ønsker at kunne gøre følgende for at repræsentere en bil i vores program (i modsætning til ovenstående):

```
Bil skoda1 = new Bil("Skoda", "Rød", 1100);
```

Dette ser måske meget nyt ud, men forskellen fra det i har gjort indtil nu er ikke så stor. Se f.eks. på denne sammenligning.

```
Bil skoda1 = new Bil("Skoda", "Rød", 1100);
String skoda2 = "Rød Skoda der vejer 1100 kg.";
```

I stedet for at oprette tekststreng med den information vi ønsker, opretter vi et objekt af typen Bil som kan holde styr på den information vi ønsker. Vi skal som altid give en variabel et navn, i de to tilfælde vores variable navnene skoda1 og skoda2. Delen til højre for lighedstegnet er lidt forskellig, men pointen er at "Rød", "Skoda"og 1100 optræder begge steder.

Ovenstående virker naturligvis kun hvis vi har forklaret computeren hvad vi forstår ved en Bil. Følgende er en klasse, som gør netop dette.

```
/*
 * En simpel klasse, der repræsenterer en bil.
 */
public class Bil {
   //Informationen vi ønsker at gemme om en bil.
   private String producent;
   private String farve;
   private int vaegt;

   //Constructor
   public Bil(String bilProducent,String bilFarve,int bilVaegt)
   {
       producent = bilProducent;
       farve = bilFarve;
       vaegt = bilVaegt;
   }
}
```

Ovenstående kode skal gemmes i en fil med navnet Bil.java, og det er nu muligt at oprette objekter af typen Bil, som det er gjort ovenfor. De tre linier

```
private String producent;
private String farve;
private int vaegt;
```

siger at objekter af typen Bil gemmer information om producent, farve og vaegt. Når vi så skal oprette en ny Bil bruges følgende, som kaldes constructoren:

```
public Bil(String bilProducent,String bilFarve,int bilVaegt)
{
    producent = bilProducent;
    farve = bilFarve;
    vaegt = bilVaegt;
}
```

En `constructor` minder om en metode, men der er dog nogle væsentlige forskelle, og man må ikke forveksle de to. Først ser vi på de ting der er de samme. `public` betyder at vi kan bruge constructoren fra andre klasser (dvs. fra andre filer, så længe filerne ligger i samme bibliotek). De tre parametre der gives med, fortæller klassen hvilke værdier der skal bruges til at oprette det nye objekt, i dette tilfælde gemmes værdierne i objektets egne variable. En constructor har altid samme navn som klassen (i vores tilfælde hedder de begge Bil), og den har ikke en returtype da det den returnere er givet på forhånd, nemlig et objekt af typen Bil.

Bemærk:
at constructorer, da de har sammen navn som klassen, altid starter med stort begyndelsesbogstav - til forskel fra metoder som har lille begyndelsesbogstav.

Den information der gemmes i objektet er gjort private, dvs. at vi ikke kan nå informationen fra andre klasser. Så når vi har oprettet objekter er informationen vi gav det gemt for os. For at få fat i informationen skal vi skrive en række metoder som vi kan bruge fra andre klasser. Efter disse er skrevet ser klassen således ud.

```
/*
 * En simpel klasse, der repræsenterer en bil.
 * Med get-metoder.
 */
public class Bil {
   //Imformationen vi ønsker at gemme om en bil.
   private String producent;
   private String farve;
   private int vaegt;

   //Constructor
   public Bil(String bilProducent,String bilFarve,int bilVaegt)
   {
      producent = bilProducent;
      farve = bilFarve;
      vaegt = bilVaegt;
```

```
   }

   //Returnerer producenten
   public String getProducent()
   {
      return producent;
   }

   //Returnerer farven
   public String getFarve()
   {
      return farve;
   }

   //Returnerer vaegten
   public int getVaegt()
   {
      return vaegt;
   }
}
```

De tre nye metoder kaldes get-metoder fordi man får fat i information gennem dem. Typen på den information man ønsker at få angives som returtype (i det første tilfælde String) og ved linien

```
return producent;
```

returneres den ønskede information (her producenten af bilen). Når man skal bruge en metode (f.eks. `getVaegt()`) på et objekt (her af typen `Bil`) gøres det ved

```
skoda.getVaegt();
```

Syntaksen for at kalde en metode på en variabel er altså:

```
objekt.methodeNavn(parametre);
```

Et eksempel på brug af klassen Bil gives nu.

```
public class BilEksempel {
    public static void main(String[] args) {
        //To bil objekter oprettes
        Bil skoda = new Bil("Skoda", "Rød", 1100);
        Bil audi = new Bil("Audi", "Sølvgrå", 1385);

        //Sammenligner vægten på de to biler.
        if( skoda.getVaegt() > audi.getVaegt ) {
            //Hvis skodaen er tungere end audien
```

```
            System.out.println("Den tungeste bil er en "
            + skoda.getFarver() + skoda.getProducent());
        }
        else {
            //Hvis audien er tungere end skodaen
            System.out.println("Den tungeste bil er en "
            + audi.getFarver() + audi.getProducent());
        }
    }
}
```

Næste eksempel gemmer endnu en gang information omkring et objekt fra vores hverdag, denne gang et kvadrat. Informationen der gemmes er sidelængden.

```
public class Square {
   private double sideLength;

   public Square(double newSideLength) {
      sideLength = newSideLength;
   }

   public double getSideLength() {
      return sideLength;
   }

   public double calculateArea() {
      double area = sideLength * sideLength;
      return area;
   }
}
```

Der er en forskel fra den første klasse vi så på, nemlig den sidste metode `calculateArea()`. Her returneres ikke blot en værdi vi har gemt tidligere, i stedet beregnes en værdi hvorefter denne returneres. Her er et eksempel på anvendelse af klassen `Square`.

```
/*
 * En klasse der udskriver sidelængde og
 * areal af de 10 kvadrater med sidelængder
 * mellem 1 og 10.
 */
public class SquareExample {

   public static void main(String[] args) {
      //For hvert tal fra 1 til 10 udskriv
      //sidelængde og areal af et kvadrat.
```

```
      for(int i=1; i<=10; i=i+1) {
         //Opretter et nyt kvadrat med sidelængde i
         Square kvadrat = new Square(i);
         System.out.println(
           "Arealet af et kvadrat med sidelængde "
           + kvadrat.getSideLength() + " er "
           + kvadrat.calculateArea();
         );
      }
   }
}
```

Keywordet `static`

Foran `main`-metoden er et begreb vi endnu ikke har forklaret nemlig `static`. I de ovenstående to klasser giver alle metoderne kun mening når man har oprettet et objekt. F.eks. giver det ikke mening at spørge om vægten på en bil, medmindre det er en helt specifik bil der er tale om. Man kan dog sagtens forestille sig metoder som kunne være nyttige at have i en bestemt klasse uden at det var nødvendigt at oprette et objekt først. Nedenfor er et sådant eksempel `printSecurity()`. Hvis man havde mange forskellige beskeder som alle havde noget med sikkerhed at gøre kunne disse samles i en klasse for at give et bedre overblik. Her ønsker man ikke at det er nødvendigt at oprette et objekt før man kan bruge metoder, det er i sådanne tilfælde at man bruger keywordet `static`.

Når `static` bruges skal metodens funktionalitet være uafhængig af de oprettede objekter - som f.eks. når man skriver en tekst ud til skærmen. Statiske metoder kaldes på næsten samme måde som ikke statiske. I stedet for at kalde dem på et objekt kaldes de på klassen de ligger i.

```
/*
 * En klasse med sikkerheds beskeder
 */
public class Security {
   public static void printSecurity() {
      System.out.println("Du skal vide at dine oplysninger");
      System.out.println("ikke vil blive videregivet eller");
      System.out.println("solgt til andre.");
   }

   public static void printPassword() {
      System.out.println("Dit kodeord skal indeholde mindst 8");
      System.out.println("tegn, og bør indeholder både små og");
      System.out.println("store bogstaver samt tal og tegn.");
   }
}
```

Syntaksen for at kalde en statisk metode er:

```
Klassenavn.methodeNavn(parametre);
```

Disse metoder kan anvendes uden at der oprettes et objekt, da de er statiske. Her er et eksempel på anvendelse.

```
/*
 * En klasse med der udskriver sikkerheds-
 * beskeder fra Security-klassen.
 */
public class SecurityExample {
   public static void main(String[] args) {
      System.out.println("Velkommen til IMADAs system!");
      System.out.println("Vær opmærksom på følgende.");
      //Udskriver de to beskeder fra Security-klassen
      //Bemærk af man skriver: klassenavn.metodenavn()
      //for at bruge statiske metoder.
      Security.printSecurity();
      Security.printPassword();
   }
}
```

Vi har nu set eksempler med klasser der indeholder ikke-statiske metoder, og klasser der indeholder statiske metoder. En klasse kan dog sagtens indeholde begge typer metoder.

Opgave 26: Cirkel

Skriv en klasse `Cirkel`, hvor konstruktoren tager en double for radius. Implementer følgende metoder:

- `getRadius()` -returnerer radius
- `getDiameter()`-returnerer diameteren
- `getArea()` - returnerer arealet (Areal af cirkel = πr^2)
- `getCircumference()` - returnerer omkredsen (omkreds af cirkel = $2\pi r$)

Opret π som en konstant (`final`) med værdien 3,14.
Prøv at lav en beregning på et par store cirkler, og lav π om til at få værdien `Math.PI`, dvs noget der ligner:
`private final double PI = Math.PI;`
Ændrer resultatet sig, hvis du regner på de samme cirkler?

Opgave 27: Andengrads ligningen

Lav en klasse `QuadraticEquation`, til at løse ligninger på formen: $ax^2+bx+c = 0$, der som konstruktor tager tre `double`'s `a`, `b`, `c`. Og har metoderne:

- `double getDiscriminant()`
- `boolean hasRealSolution()`
- `double getFirstSolution()`

- `double getSecondSolution()`

Implementer også en main metode der opretter mindst tre objekter og bruger dem. Hint: `hasRealSolutions` kan med fordel benytte `getDiscriminant`

3.18 `toString` - metoden

Vi ser endnu engang på `Bil`-klassen fra afsnittet tidligere. Ofte er der brug for at man hurtigt kan se den vigtigste information om en given bil, f.eks. hvis man skal have bilen solgt. Før når vi skulle se hvilken værdi vi havde gemt i en `int` kunne vi skrive:

```
int i = 5;
System.out.println(i);
```

Hvor vi så ville se: "5". Hvis vi forsøger det samme med et objekt af typen `Bil`, eksempelvis:

```
Bil skoda1 = new Bil("Skoda", "Rød", 1100);
System.out.println(skoda1);
```

Her fåes et ikke særligt brugbart resultat, nemlig noget der minder om `Bil@3e25a5`. Dette betyder at vi kan finde vores `Bil`-objekt på adressen 3e25a5 i computeren hukommelse - formodentlig ikke det vi ønskede at vide. Heldigvis kan der rettes op på dette.

Det der sker at objektets `toString`-metode kaldes. Denne metode findes altid også selvom det er et objekt af en type vi har fundet på. Løsningen på ovenstående problem er at skrive vores egen `toString`-metode, som så "overskriver"den Java ellers vi have brugt. En `toString`-metode for klassen `Bil`, ser f.eks. således ud (metoden skal skrives i `Bil` klassen):

```
public String toString() {
   return producent;
}
```

Det er vigtigt at metodens først linie er `public String toString()`, altså at den returnerer en `String`, er public og ikke tager nogen parameterer. Hvis linien ikke ser således ud overskriver (erstatter) man ikke den standard `toString` der ellers bruges. Hvis ovenstående var implementeret ville man i eksemplet i stedet får outputtet: "Skoda", som trods alt giver lidt mere mening. En mere omfattende `toString`-metode kunne se således ud:

```
public String toString() {
   String tekst = "Producent: "+producent+"\n";
   tekst = tekst+"Farve: "+farve+"\n";
   tekst = tekst+"Vægt: "+vaegt+"\n";
   return tekst;
}
```

Hvor output af eksemplet så ville blive:

```
Producent: Skoda
Farve: Rød
Vægt: 1100
```

Opgave 28: `toString` metode til tidligere klasser
Opret en `toString` metode i `Cirkel` og `QuadraticEquation` klasserne fra det forrige afsnit. Opret et par objekter og udskriv dem.

3.19 Rekursive funktioner

Rekursion er helt kort beskrevet en metode der kalder sig selv.

Fra matematikken kender vi fakultet: $n! = n \cdot (n-1) \cdot (n-2) \ldots 2 \cdot 1$. Det er defineret rekursivt, da $n!$ er det samme som $n \cdot (n-1)!$ (nb. $0! = 1$ pr definition). Det kan vi i JAVA skrive som:

```
private static int fakultet(int n)
{
   if(n == 0 || n == 1) {
       return 1
   }
   else {
       return n * fakultet(n-1);
   }
}
```

Flere sorteringsalgoritmer bruger rekursion blandt andet merge sort og quicksort.

Opgave 29: Fibonacci tallene
Fibonnacci tallene: $0, 1, 1, 2, 3, 5, 8, 13, 21 \ldots$ kan defineres rekursivt. Det n'te fibonaccital f_n kan defineres som $f_n = f_{n-1} + f_{n-2}$, sammen med $f_0 = 0$ og $f_1 = 1$. Implementer en rekursiv funktion der kan beregne fibonacci tal, og lav en udskrift af de første 30 fibonnacci tal.

Opgave 30: e - Eulers tal
Som i måske ved: $e = \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \frac{1}{5!} + \ldots$
Implementer et program der finder e ud fra sumrækkens første 10 led. Hvad med de første 15 led. (Hint i en løkke er det nemt at rette). Hvor mange led skal der til før det bliver en god approksimation? (lad os her sige at en god approksimation er 2,7182818285).

4 Nogle af JAVAs indbyggede klasser

Vi har tidligere set et par eksempler på klasser som stillede noget funktionalitet til rådighed, så vores arbejde blev noget lettere (Security-klassen og opgave med simple matematiske operationer). En stor del af at lære at programmere i Java er at vide hvilken funktionalitet der allerede er lavet for os. I dette afsnit gennemgåes et lille udvalg af ofte brugte metoder fra Javas eget bibliotek, der i sin helhed kan findes på:

`http://java.sun.com/javase/6/docs/api/`

Dele af biblioteket er gengivet i [1].

4.1 String

Der er en lang række metoder som gør det muligt at modificere og sammenligne tekststrenge. Alle disse funktioner kan bruges uden at det er nødvendigt at importere en special klasse.

`char charAt(int)`

Man kan få fat i et tegn i en streng ved at bruge metoden `charAt`. Metoden skal have en parameter, som angiver hvilket tegn man ønsker - husk på at man starter ved 0. Dvs. `charAt(2)` giver det tredje tegn i strengen.

`int length()`

Længden af en tekststreng kan findes ved brug af metoden length().

```
/*
 * Læser en streng fra brugeren og skriver
 * den ud med et tegn på hver linie.
 */
public class StringEksempel1 {
   public static void main(String[] args) {
      Scanner s = new Scanner(System.in);

      //Læser tekst fra brugen
      System.out.println("Indtast noget tekst:");
      String input = s.nextLine();

      //Løber gennem teksten og udskriver den
      //med et tegn på hver linie
      for( int i=0; i<input.length(); i=i+1) {
         System.out.println(s.charAt(i));
      }
   }
}
```

`boolean equals(String)`

Man kan teste om to strenge ens ved at bruge equals metoden. Den ene streng er den man kalder metoden på, og den man sammenligner med gives som parameter. Hvis de to strenge er ens returneres true eller returneres false.

`boolean equalsIgnoreCase(String)`

Hvis man ønsker at sammenligne to tekststrenge, men gerne vil ignorere forskelle mellem små og store bogtaver kan dette gøres med equalsIgnoreCase(). Metoden bruges på samme måde som equals.

```
/*
 * Læser to strenge fra brugeren og
 * undersøger om de ens.
 */
public class StringEksempel2 {
   public static void main(String[] args) {

      //Læser tekst fra brugeren
      Scanner s = new Scanner(System.in);
      System.out.println("Indtast noget tekst:");
      String input1 = s.nextLine();
      System.out.println("Indtast noget mere tekst:");
      String input2 = s.nextLine();

      //Hvis de to strenge er ens ud skrives teksten
      if( input1.equals(input2) ) {
         System.out.println("Du skrev det sammme");
         System.out.println("begge gange.");
      }
   }
}
```

`String toLowerCase()` **og** `String toUpperCase()`

En tekststreng kan modificeres med følgende to metoder, som gør det man forventer (nemlig ændrer alle bogstaver til store bogstaver eller til små bogstaver). De to metoder tager ingen parametere.

`String substring(int, int)`

Hvis man skal bruge en del af en tekststreng kan dette gøres med metoder subString(), som returnerer en delstreng og som tager to parametere nemlig start indekset og slutindekset. Den delstreng der returneres er fra og med startindekset og til men ikke med slutindekset.

`int indexOf(String)`

Denne metode giver det første indeks hvor parameter strengen findes i den streng metoden kaldes på. Dvs. indekset af "de"i "kludder"er 3 (husk at man starter ved nul). Hvis parameterstrengen ikke findes returneres -1.

```
/*
 * Læser to strenge fra brugeren og afgør
 * om den anden er en delstreng af den første.
 */
public class StringEksempel3 {
   public static void main(String[] args) {

      //Læser tekst fra brugeren
      Scanner s = new Scanner(System.in);
      System.out.println("Indtast noget tekst:");
      String input1 = s.nextLine();
      System.out.println("Indtast noget mere tekst:");
      String input2 = s.nextLine();

      //Hvis input2 er en delstreng af input1
      //udskrives en tekst der fortælle dette
      if( input1.subString(input2) != -1 ) {
         System.out.println(input2+" er en delstreng");
         System.out.println("af "+input1);
      }
      else {
         System.out.println(input2+" er ikke en");
         System.out.println("delstreng af "+input1);
      }
   }
}
```

Opgave 31: STORE og små bogstaver
Skriv et program som tager en tekststreng som input og skriver den ud udelukkende med små bogstaver, og derefter udelukkende med store bogstaver. Hint: Brug metoderne toLowerCase og toUpperCase.

Opgave 32: Bagvendt
Skriv et program som tager en tekststreng som input og skriver den ud baglæns. Hint: Brug metoderne length() og charAt() samt en for-løkke.

Opgave 33: Propaganda program
Skriv et program som bliver ved med at bede brugeren om input indtil denne skriver "DM502 er sjovt", hvis brugeren skriver noget andet udkskriver programmet blot teksten "DM502 er sjovt". Hint: Brug en while-løkke og metoden equals() (man kan også bruge equalsIgnoreCase()).

4.2 ArrayList

Indtil nu har vi brugt arrays når vi skulle opbevare en liste af tal eller objekter. Problemet med at bruge arrays opstår når vi gerne vil tilføje et ekstra tal eller objekt til listen - da arrays jo ikke kan ændre størrelse. Det er her at `ArrayList` kommer ind i billedet. `ArrayList` er en liste, som ændrer størrelse efterhånden som vi gemmer flere objekter i den.

For at bruge en `ArrayList` skal man først oprette et objekt. Når man opretter en `ArrayList` skal man fortælle hvad det er man ønsker at gemme i den, her gives et eksempel med String.

```
ArrayList<String> list = new ArrayList<String>();
```

Vi opretter altså en `ArrayList` på samme måde som med alle andre objekter, dog med den forskel at vi angiver at vi kun vil gemme ting af typen `String`. Dette gøres med `<String>` efter de steder hvor der står `ArrayList`. Hvis ønskede at gemme objekter af en anden type, f.eks. `Square` ville der står `Square` i stedet `String`. I nedenstående vil `String` så også blive ændret til `Square`.

`int size()`

Hvis man vil vide hvor mange objekter man har gemt indtil videre kan denne metode bruges. Den returnerer størrelsen på `ArrayList`'en, dvs. hvis man har fem elementer i sin liste får man at vide, at listen har størrelsen fem.

`void add(String)`

Man kan tilføje objekter til listen ved at bruge denne metode. Objektet tilføjes til sidst i listen.

`String get(int)`

Man kan få at vide hvilket objekt man har gemt på en given plads ved at bruge denne metode. Som med arrays starter nummeringen fra 0, dvs. `get(4)` giver en det femte objekt i listen. Metoden ændrer ikke på listen, for at fjerne et objekt bruges `remove`.

`String remove(int)`

Denne metoder virker ligesom `get`, dog med den forskel at objektet på den plads man efterspørg fjernes fra listen. Listen bliver altså et element mindre efter man har kaldt `remove`. Resten af elementerne i listen rykke sammen, så hvis man fjerne element nummer fire, rykker de efterfølgende elementer en plads ned (fra plads fem til fire, fra seks til fem osv.).

4.3 HashMap

Hvis du f.eks skal lave en telefonbog, får du brug for en måde du kan gemme telefonnumre som en 'mapping' af navnet. (lad os er antage at hver person kan

nøjes med ét telefonnummer). I JAVA kan man bruge klassen HashMap til denne funktionalitet.

For at oprette et HashMap, skal den vide hvilken mapping den skal indeholde. I vores tilfælde vil vi gerne gemme en String, og få en int tilbage. Da en int er en simpel type, og HashMap opererer på Objekter, bruges en 'wrapper class' så vores mapping ser således ud: `String` $\xrightarrow{m}$ `Integer`
I JAVA ser det således ud:

```
HashMap<String,Integer> navn = new HashMap<String,Integer>();
```

når du har importeret:

```
import java.util.HashMap;
```

`put` og `get`

Hvis vi nu vil gemme i vores HashMap, bruges metoden `put('Key','Værdi')`, og hvis vi skal hente noget ud bruges metoden `get('Key')`.

Vores telefonbogseksempel kan derfor se således ud:

```
import java.util.HashMap;
/*
 * Eksempel på en meget simpel telefonbog
 */
public class Telefonbog {

    HashMap<String,Integer> tlfbog;

    //Construktor for telefonbogs klassen
    public Telefonbog()
    {
        tlfbog = new HashMap<String,Integer>();
    }

    //Tilføj til telefonbogen
    public void tilfoej(String navn, Integer nummer)
    {
        tlfbog.put(navn,nummer);
    }

    //Slå op i telefonbogen
    public Integer opslag(String navn)
    {
        return tlfbog.get(navn);
    }
}
```

`containsKey`

Hvis du vil vide om et HashMap indeholder en mapping fra en nøgle, kan du bruge metoden `containsKey('Key')`, der returnerer en boolean. I telefonbogen kunne en metode altså se således ud:

```
private boolean harTelefonNummer(String navn)
{
    return tlfbog.containsKey(navn);
}
```

Hvis du prøver at slå op i et HashMap, og den ikke har en mapping med den angivne nøgle, returneres værdien `null`, Dette er den JAVA værdi, der repræsenterer det tomme objekt, altså et objekt der ikke eksisterer.

Opgave 34: Telefonbog og adressebog
Lav en implementation af en telefon og adresse bog i JAVA.
Start med at lave en Person klasse, der kan indeholde oplysninger om navn, adresse og telefonnummer.
Lav så en Telefon og Adresse bogs klasse, der gemmer oplysningerne i et `HashMap<String,Person>`. Din klasse skal kunne gemme personer, hente oplysninger ud gen, og du må ikke returnere `null`. I stedet kan du vælge at returnere `0` som telefon nummer, eller du kan oprette en person der hedder `Ikke Eksisterende` og returnere ham.
Lav til sidst en klasse med en main metode, der opretter en telefonbog, gemmer tre personer, og søger på både et eksiosterende og et ikke eksisterende navn.

4.4 Scanner

En efterhånden velkendt klasse, hvis du har læst dette notesæt. Der er alligevel her en opsamling af det der er præsenteret forskellige steder før og nyt.

Input fra tastaturet

Importer Scanner klassen:
`import java.util.Scanner;`
Nu kan vi oprette en variabel af typen Scanner med linien:
`Scanner sc = new Scanner(System.in);`
`System.in` angiver tastaturet.
Nu kan du kalde en af de nedenstående metoder på din Scanner variabel `sc`.

Input fra en fil

Scanner klassen er let at bruge, hvis du skal have tekst indlæst fra en fil (klassen File kan du selv slå op i JAVA API'en). Følgende måde oprettes en scanner, der indlæser fra filen tekst.txt:

```
Scanner sc = new Scanner(new File("tekst.txt"));
```

Hvor det kan bemærkes at filen oprettes som et objekt af typen File. Det kræver at følgende imports er lavet:

```
import java.io.File;
import java.util.Scanner;
```

Metoder til input

Når du har oprettet et Scanner objekt, kan du kalde følgende metoder:

`hasNext` Returnerer `true`, hvis scanneren har flere 'tokens'

`next` Returnerer en `String` med det næste ord.

`nextLine` Returnerer en `String` med hele den næste linie.

`nextInt` Returnerer en `int`.

`nextDouble` Returnerer en `double`

`nextBoolean` Returnerer en `boolean`

Her er et eksempel på hente hele filens indhold, som én lang streng, uden linieskift, og udskriver indholdet:

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;
/*
 * Program der indlæser en tekstfil
 * og udskriver indholdet
 */
public class ScannerDemo
{
    // Main metode for programmet.
    public static void main(String[] args)
    {
        Scanner sc;
        String inputText = "";
        try {
            sc = new Scanner(new File("tekst.txt"));
            while(sc.hasNext()) {
                inputText = inputText + sc.nextLine();
            }
        } catch (FileNotFoundException e) {
            System.out.println("Kunne ikke finde filen");
        }
        System.out.println(inputText);
    }
}
```

Opgave 35: Input fra tekstfil
Modificer den ovenstående ScannerDemo klasse, til at indsætte linieskift i tekst strengen. (linieskift er '\n').

Opgave 36: Ordtæller
Lav et program, der spørger om et filnavn, og fortæller hvor mange ord der er i filen.

4.5 Math

Funktionaliteten fra Math-klassen kan bruges uden at det er nødvendigt at importere denne, da metoderne er erklærede `static`. Klassen indeholder en lang række funktioner til at lave beregninger som ville være svære med de almindelige regneoperationer. Her er en liste over funktionerne samt et eksempel hvordan de bruges.

- double sin(double)
- double cos(double)
- double tan(double)
- double asin(double)
- double acos(double)
- double atan(double)
- double pow(double, double) - tager den første parameter og opløfter til den anden
- double abs(double) - tager den absolutte værdi af parameteren
- double sqrt(double) - tager kvadratroden af parameteren
- double floor(double) - runder parameteren ned til nærmeste heltal
- double ceiling(double) - runder parameteren op til nærmeste heltal
- double log(double) - tager den naturlige logaritme til parameteren
- double log10(double) - tager 10-tals logaritmen til parameteren
- double max(double, double) - returnerer det største af de to tal
- double min(double, double) - returnerer det mindste af de to tal
- double random() - returnerer et tilfældigt tal mellem 0 og 1

Bemærk at abs, max og min også findes i udgaver det tager parametere af typen int. Der findes desuden to konstanter i Math-klassen: pi og e (grundtallet for den naturlige logaritme). Disse kan bruges ved at skrive Math.PI og Math.E. Her gives et eksempel på anvendelse af enkelte af metoderne fra Math-klassen.

```
/*
 * Skriver værdi af pi og e ud, samt værdien af
 * enkelte simple beregning med pi og e.
 */
public class MathEksempel1 {
   public static void main(String[] args) {
      //Udskriver pi og e
      System.out.println("Pi er cirka "+Math.PI);
      System.out.println("e er cirka "+Math.E);

      //Opløfter pi i anden og udskriver resultatet
      double pi2 = Math.pow(Math.PI, 2.0);
      System.out.println("Pi i anden er cirka "+ pi2);

      //Beregner og udskriver cos til pi
      System.out.println("cos(pi) = "+Math.cos(Math.PI));
   }
}
```

4.6 Comparable Interface

Hvis du har fået en titel som Cand.Scient med speciale i datalogi, så forpligter det. Blandt andet forventes det at du kan programmere, analysere køretid og meget andet. Det samme gør sig gældende for et *interface*. Hvis en klasse implementerer (i JAVA: `implements`) et interface, så forpligter klassen sig til at have de metoder som interfacet specificerer.

Et interface minder om en klasse i den forstand at den har et navn og metoder. Der er ingen kode i metoderne, den fortæller kun hvilke metoder der kan kaldes på klasser der implementerer den.

Til at definere en naturlig ordning af objekter kan man derfor implementere interfacet *Comparable*. Her forpligter klassen sig til at have en metode compareTo, der skal opføre sig således, når A.compareTo(B)

returnere:	Hvis:
A < B	-1
A = B	0
A > B	1

Her er et meget tænkt eksempel for at demonstrere hvordan det kan gøres. Det er værd at bemærke, at hvis man har en ArrayListe med elementer, der implementerer Comparable, kan den sorteres blot ved at kalde den statiske metode: `Collections.sort(listen)`

```
import java.util.ArrayList;
import java.util.Collections;

/*
```

```
 * Klasse der demonstrerer implementation af interface
 */
public class InterfaceDemo implements Comparable<InterfaceDemo>
{
    int objektVaerdi;

    // constructor for InterfaceDemo
    public InterfaceDemo(int v)
    {
        objektVaerdi = v;
    }

    //get metode for objektVardi
    public int getObjektVaerdi()
    {
        return objektVaerdi;
    }

    // Objekter af denne klasse kan sammenlignes
    public int compareTo(InterfaceDemo demo)
    {
        // Objektet her er størst
        if(objektVaerdi > demo.getObjektVaerdi()) {
            return 1;
        }
        // Det andet objekt er størst
        if(objektVaerdi < demo.getObjektVaerdi()) {
            return -1;
        }
        // Hvis ingen er størst, er de ens
        return 0;
    }

    // Laver en streng af objektet
    public String toString()
    {
        return ""+objektVaerdi;
    }

    // Main metode for programmet
    public static void main(String[] args)
    {
       ArrayList<InterfaceDemo> list =
                                    new ArrayList<InterfaceDemo>();
       for(int i = 15 ; i > 0; i--) {
           list.add(new InterfaceDemo(i));
       }
```

```
        System.out.println(list);
        Collections.sort(list);
        System.out.println(list);
    }
}
```

Når eksemplet køres, giver det følgende output:

```
[15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
```

Opgave 37: Comparable interface
Skriv en klasse Punkt, der repræsenterer et punkt i planen, og implementerer Coparable, ud fra et sammenligningeskriterium om afstand fra centrum (0,0). Skriv en main metode med to for løkker indeni hinanden, der opretter en ArrayList med alle punkter med heltals koordinater i intervallet $0 < x < 6$ og $0 < y < 6$. Lav en toString metode i Punkt klassen, og udskriv ArrayListen både før og efter sortering.

5 Test og debugging

Før et program er færdigt skal det testes for både funktionalitet og korrekthed. Det vil sige kan det gøre det som opgaven beskriver, og er resultatet korrekt. Debugging er den proces der går ud på at rette fejl, når de er fundet.

5.1 Test

Der er utallige måder at teste sit program på. Her beskrives kort nogle af de ting der skal testes for.

Positive test

Reagerer programmet korrekt, når det får et korrekt input?

I positiv testning, er man venlig ved programmet, og giver det pæne input, men 'regner' efter i hånden om resultatet er korrekt. Her er det primært funktionaliteten der testes. Husk at opstille tests der tester alle opgavens dele.

Negative test

Reagerer programmet korrekt når det får forkert input? Hvis brugeren skal indtaste et tal, og indtaster noget andet, får han så en pæn fejlbesked og lov til at prøve igen, eller stopper programmet uventet? Hvad med negative tal, nul og positive tal? En integer har en maksimal og en minimal størrelse (se evt appendix A.2), hvad sker der hvis det er præcis så stort/småt eller større/mindre?

I negativ testning prøves programmet med alle slags input, forkerte og i grænseområder, f.eks. meget store tal, meget store strenge, den tomme streng, osv. Dit program skal gerne håndtere dette på en pæn måde, og give brugeren mulighed for at prøve igen.

Opgave 38: Middelværdiprogram

Skriv følgende program til at beregne middelværdi af, og lav forbedringer så det virker både korrekt og robust. Lav det også om så det kan håndtere negative tal, og decimal tal.

```
import java.util.Scanner;
/*
 * Beregning af middelvaerdi af positive hele tal
 * Til slut indtastes et ikke-positivt tal
 */
public class Middelvaerdi
{
    // Main metode i programmet
    public static void main(String[] args)
    {
        int antal;
        int nytTal;
```

```
        int sum;
        double resultat;
        Scanner tastatur = new Scanner(System.in);

        antal = 0;
        sum = 0;

        System.out.println("Indtast et helt tal:");
        nytTal = tastatur.nextInt();
        while(nytTal > 0) {
            sum = sum + nytTal;
            antal = antal + 1;
            System.out.println("Indtast et nyt tal:");
            nytTal = tastatur.nextInt();
        }
        resultat = (double) sum/antal;
        System.out.println("Middelvaerdi = " + resultat);
    }
}
```

5.2 Simpel debugging

Den mest anvendte metode til debugging er at udskrive information efterhånden som man når til specielle steder i programmet, og gøres let med:

```
System.out.println();
```

Er du i tvivl om du kommer ind i en løkke og får lavet arbejde i den, så indsæt en `System.out.println("Inden i løkken");` som den første linie indeni. Så kan du køre programmet, og tydeligt se om linien udskrives.

Det er også en god idé at udskrive information om sine variabler og lignende.

5.3 Assertions

En bedre og mere permanent måde at sikre korrektheden i sit program er at bruge assertions. En assertion er en oplysning man ved skal være sand på det sted i programmet som man indsætter den.

Assertions er på formen:

```
assert(det man ved skal gælde, fx: rente > 0);
```

Når programmet skal køres skal du så huske at skrive `java -ea ProgramNavn` . `-ea` er en forkortelse for 'enable assertions'. Når `-ea` udelades, springer JAVA over assertions, så du kan derfor lade dem blive stående, i modsætning til at bruge `System.out.println()`, der skal udkommenteres eller slettes.

Et godt eksempel er hvis man regner restgæld ud for et lån, ved en metode der hedder `beregnRestGaeld()`, og naturligt kan en restgæld ikke være negativ, så kunne det se således ud:

```
double restGaeld;
restGaeld = beregnRestGaeld();
assert(restGaeld >= 0);
```

Hvis vi får en fejl på denne assertion, er værdien af `restGaeld` negativ. Den er lige blevet tildelt af metoden `beregnRestGaeld()`, så vi skal nok have testet den metode bedre, da den giver et resultat der ikke er rigtigt.

Hvis du vil have information ud samtidig med at din assertion meddeler om fejl, er syntaksen:

```
assert(det man ved skal gælde, fx: rente > 0) : "Fejlbesked";
```

Vores renteeksempel kunne derfor se således ud:

```
double restGaeld;
restGaeld = beregnRestGaeld();
assert(restGaeld >= 0) : "Restgæld negativ: "+ restGaeld;
```

Hvor en fejlbesked og værdien af variablen der fejler udskrives til skærmen, hvis der er fejl.

Opgave 39: Assertions
Skriv et simpelt program, f.eks til at lægge tal sammen. Brug assertions, og få dem til at 'opdage' fejl i antagelserne om input.

6 Pseudokode til JAVA

Tit er løsningerne der skal implementeres beskrevet i pseudokode, der minder om programmeringssprog, blot har det ikke implementations specifikke detaljer med. Det er vigtigt at kunne omsætte pseudokode til JAVA kode, da i senere i studiet vil skulle bruge det.

Primtalstest

Primtal er løst defineret positive heltal større end 1, der kun kan deles af 1 og sig selv, dvs. $2, 3, 5, 7, 11, 13 \ldots$. Vi vil her se på kode der tester og finder primtal. Følgende er en naiv primtalstest skrevet i pseudokode:

```
isPrime(n)
1  for i ← 2 to n − 1
2        do
3           if i deler n
4              then return FALSE
5  return TRUE
```

Opgave 40: Simpel primtalstest
Implementer `isPrime` og lav en `main` metode der bruger isPrime til at teste om 7789 og 7791 er primtal.

Den ovenstående test deler med mange lige tal. Det eneste lige tal der er nødvendigt at dele med er 2.

Opgave 41: Forbedret primtalstest
Lav forbedringen på primtalstesten, til kun at dele med to, og så de ulige tal fra tre og op.
Faktisk er det nok at teste op til kvadratroden af tallet, da enhver faktor større end det ville skulle ganges med en mindre faktor, der allerede er testet.
Lav også denne forbedring.

Sorterings algoritmen SelectionSort

Selection sort virker ved at finde det mindste tal der ikke er sorteret, og placere det efter dem der er sorteret.

```
SelectionSort(A, size)
1  for i ← 0 to size − 1
2        do minIndex ← i
3           for j ← i + 1 to size
4                 do if A[j] < A[minIndex]
5                       then min ← j
6           Swap(A[i], A[minIndex])
```

Opgave 42: Selectionsort

implementer selection sort. Tænk på om det er nødvendigt at give parametren size med? (Hint. ved et array/ArrayList ikke sin egen størrelse i JAVA.) Lav også et program der fylder et array med tal (brug evt. et array af double, og metoden Math.random()), udskriver det, sorterer det og udskriver det igen.

A Appendix

A.1 Rettigheder for filer

Når du gemmer dine filer på IMADA systemet, er det groft set én stor harddisk som du deler med alle andre. Det betyder også at du selv skal angive for hver enkelt fil, hvem der må læse, skrive og køre dine filer.

I en shell kan du med `ls -la` se hvilke rettigheder der er på filer. et eksempel er:

```
-rwxrw-r--  dummy07 dummy07 53554 2007-06-23 22:37 HelloWorld
```

Den første bindestreg angiver at det er en fil og ikke et bibliotek. Havde det været et bibliotek, havde det første tegn været et d. De næste 9 tegn, skal læses tre af gangen. Tre til dig selv (user) tre til den gruppe som filen tilhører, og tre til alle andre. De første tre er `rwx`, der angiver at du selv må læse (r: read), skrive (w: write) og køre (x: execute). Her må gruppen læse og skrive, mens andre kun må læse. De to dummy07 betyder at filen ejes af dummy07, og at den tilhører gruppen dummy07. Så er der filens størrelse, klokkeslet for sidste ændring, og filnavnet.

Hvis du vil ændre rettigheder, og det vil du for at undgå eksamenssnyd!!! så skal du bruge kommandoen `chmod`. Eksempel:

```
chmod o-r HelloWorld
```

Angiver at andre (o) skal have fjernet (-) rettigheden til at læse.

Hvis du skal være den eneste der skal kunne læse og skrive til din fil (lyder fornuftigt ikke?), skal du altså skrive kommandoen:

```
chmod u+rw,g-rwx,o-rwx filnavn
```

Hvis det er alle filer i samme bibliotek og underbiblioteker du gerne vil have sat rettighederne på, kan du gøre det rekursivt ved at bruge parametren `-R`, og en stjerne angiver alle filer:

```
chmod u+rw,g-rwx,o-rwx -R *
```

Sætter altså læse og skrive rettighederne til kun dig for alt hvad der ligger i biblioteket og underbilioteker.

> **Bemærk:**
> Det er dit ansvar at andre ikke kopierer din obligatoriske aflevering, så sørg for at holde dine rettigheder på filer rigtige

Der er en kortere skrivemåde, hvis du skal have skrive og læserettigheder, og andre ikke skal have noget, det er:

```
chmod 600 filnavn
```

læs selv mere, i info/man siderne om chmod, som kan ses ved at skrive `man chmod` i en shell (eller `info chmod`).

Hvis du hellere vil gøre det grafisk, kan du finde din fil i `home folder`, højreklikke på den, vælge `Properties`, og `Permissions`, og sætte det korrekt i menuen der.

A.2 Typer i JAVA

Her er en tabel over de simple typer der eksisterer i JAVA, samt deres interval

Type	Beskrivelse	Interval
byte	meget lille heltal	-128 til 127
short	lille heltal	-32768 til 32767
int	Heltal	-2147483648 til 2147483647
long	stort heltal	-9223372036854775808 til 9223372036854775807
float	små decimaltal	$\pm 1.4 * 10^{-45}$ til $3.4 * 10^{38}$
double	Decimaltal (punktum som komma)	$\pm 4.9 * 10^{-324}$ til $1.8 * 10^{308}$
char	Et enkelt bogstav eller tegn	Unicode character
boolean	Sandt eller falsk	-

Hvis du skal bruge den største værdi en int kan indeholde, kan den fåes ved `Integer.MAX_VALUE`, mindste værdi ved `Integer.MIN_VALUE`.

A.3 Retningslinier for pæn kode

På Sun's hjemmeside er der udførlige guides til hvordan pæn JAVA-kode skal skrives. Det kan findes her: `http://java.sun.com/docs/codeconv/`
Pæn kode er også med til at du selv kan læse din egen kode, og andre hurtigere kan hjælpe hvis du har problemer, eller der er noget der ikke virker. Her er nogle af de vigtigste ting.

Navngivning

- Brug meningsfyldte navne. `rente` og `temperatur` fortæller meget mere end `x` og `y`
- Klasser starter med stort
- Metoder og variable starter med småt
- Konstanter er skrevet med kun store bogstaver

Kommentarer

- Alle klasser skal have en kommentar lige over klassens navn.
- Alle metoder skal have en kommentar over metoden
- Hvis din kode ikke er let at forstå skal den kommenteres.

Layout

- Alle statements i en blok indrykkes 4 mellemrum (eller 1 tab)
- Brug altid tuborg parenteser i if/else og loops. Også hvis det kun er én statement.
- Brug en tom linie mellem metoder (og konstruktorer)
- Brug mellemrum omkring operatorer (+,-,*, etc.)

Restriktioner i sproget

- Rækkefølgen i klassen skal være:
  1. Import statements
  2. Klasse kommentar
  3. Klasse header
  4. Instans variable
  5. Constructor
  6. Metoder
- Lav instans variable private, og brug `get` og `set` metoder til at ændre deres værdier.
- Brug altid `public` eller `private` ved metoder.

Litteratur

[1] *Supplerende noter til DM502 - DM503, Programmering A & B*, Edmund Christiansen.

[2] *Introduction to Algorithms, second ed.*, Cormen, Leiserson, Rivest & Stein.