Introduktion til Objekt-Orienteret Programmering

Øvelsesbog – Modul 3.2: Collections

Aslak Johansen <asjo@mmmi.sdu.dk>
Anders Clausen <ancla@mmmi.sdu.dk>

November 2, 2021

Contents

Part I
Øvelser

1 Typer

1.1 Hello, World

Skriv et program, der udskriver teksten "Hello, World".

1.2 Temperatur

Hvilke datatyper er velegnede til at representere en temperatur?

1.3 Måned

Hvilke typer er velegnede til at repræsentere en måned?

1.4 Heltallige Grænser

1. Hvilke heltallige datatyper kan man benytte sig af i Java?
2. Vælg én af dem.
3. Skriv et program der eksperimentelt afslører hvad der sker når man overskrider den størst mulige værdi.
4. Beskriv hvad det er I observerer?

1.5 Casting

Når vi godt vil konvertere mellem `int` værdier og `long` værdier skal der udføres et cast. Men, hvornår er det at dette cast skal gøres eksplicit, og hvornår må det være implicit?

- Hvorved adskilder `int` og `long` sig?

- Prøv at skrive et program der (i) erklærer `i` som en `int` variabel, (ii) tildeler den en værdi, (iii) erklærer `l` som en `long` variabel, (iv) tildeler værdien af `i` til `l`, og (v) slutteligt tildeler værdien af `l` til `i`.
- Eksperimentér med hvor det er nødvendigt at have eksplicitte casts.
- Gør nu det samme med en `float` variabel `f` og en `double` variabel `d`.
- Eksperimentér igen med hvor det er nødvendigt at have eksplicitte casts.
- Afhænger resultatet af disse eksperimenter af den værdi som I initielt tildeler den første variabel?

2 Expressions

2.1 Definition

Hvad er et expression?

2.2 Tildeling

Er en tildeling (af en værdi til en variabel) et expression?

2.3 Expression vs Statement

Hvad er forskellen på et expression og et statement?

3 Variable

3.1 Areal af Cirkler

Skriv et program der udregner og udskriver arealet ($\pi \cdot r^2$) af tre cirkler med radius på hhv. 1, 3 og 5.

3.2 Sum af Areal af Cirkler

Skriv et program der udregner og udskriver omkredsen ($2 \cdot \pi \cdot r$) af tre cirkler med radius på hhv. 1, 3 og 5, og afslutter med at udskriven summen af disse.

3.3 Celcius til Fahrenheit

Skriv et program, hvori

1. En temperatur angives i Celcius via en variabel.

2. Denne temperatur konverteres til Fahrenheit og gemmes i en anden variabel.
    - Formel: $T_F = 32 + \frac{9}{5}T_C$
3. Konverteringen udskrives på en passende måde.

3.4 Epoch

Skriv et program hvori

1. Et antal sekunder siden et bestemt tidspunkt (fx 1. Januar 1970) gemmes i en variabel.
2. Konvertér dette tal til et helt antal år (lad os antage at der er 365 dage på et år) og et antal hele dage indenfor det sidste år. Lægges disse to tal sammen skal resultatet altså være indenfor 24 timer af udgangspunktet.
3. Udskriv disse to tal.

Verificér at programmet virker.

3.5 Inkrementering af Måned

Skriv et program, hvori

1. En heltallig variabel bruges til at repræsentere en måned.
2. Denne variabel tildeles en værdi (du vælger selv).
3. Udskriv variablens værdi.
4. Forøg værdien af denne variabel med en halv.
5. Udskriv variablens værdi.
6. Forøg værdien af denne variabel med en halv.
7. Udskriv variablens værdi.

Få dette program til at oversætte, kør det og beskriv hvad du observerer.

Forklar hvorfor programmet opfører sig sådan.

3.6 Værdi vs Variabel

Hvad er forholdet mellem en værdi og en variabel?

3.7 Daglige Differencer

Skriv et program, der givet 7 dagstemperaturer udregner og udskriver temperaturdifferencen mellem alle to på hinanden følgende dage (dvs. Tirsdag-Mandag, Onsdag-Tirsdag . . . Søndag-Lørdag).

Dagstemperaturerne kunne være:

- Mandag: 21.5
- Tirsdag: 23.7
- Onsdag: 19.6
- Torsdag: 22.5
- Fredag: 25.3
- Lørdag: 21.7
- Søndag: 18.9

3.8 Gennemsnitlig Alder

Betragt følgende program:

```
public class AvgAge
{
    public static void main (String[] args) {
        int ada_lovelace      = 36; // https://en.wikipedia.org/wiki/Ada_Lovelace
        int dennis_ritchie    = 70; // https://en.wikipedia.org/wiki/Dennis_Ritchie
        int grace_hopper      = 85; // https://en.wikipedia.org/wiki/Grace_Hopper
        int hedy_lamarr       = 85; // https://en.wikipedia.org/wiki/Hedy_Lamarr
        int edsger_dijkstra   = 72; // https://en.wikipedia.org/wiki/Edsger_W._Dijkstra
        int douglas_engelbart = 88; // https://en.wikipedia.org/wiki/Douglas_Engelbart

        float male_avg   = (float)(dennis_ritchie+edsger_dijkstra+douglas_engelbart)/3;
        float female_avg = (float)(ada_lovelace+grace_hopper+hedy_lamarr)/3;
        float avg = (male_avg+female_avg)/2;
        float diff = male_avg-female_avg;

        System.out.print("Average livespan of a male computer scientist: ");
        System.out.println(male_avg);
        System.out.print("Average livespan of a female computer scientist: ");
        System.out.println(female_avg);
        System.out.print("Average livespan of a computer scientist: ");
        System.out.println(avg);
        System.out.print("Males lives this much longer than female: ");
        System.out.println(diff);
    }
}
```

Udfør programmet. Hvad sker der?

Skriv nu en tekst på dansk hvor I ved hjælp af fagtermer forklarer hvad der sker. Sørg for at denne tekst er grundig nok til at en programmør kan genkonstruere ovenstående kode.

3.9 Printf

Undersøg hvordan følgende stykke kode fungerer ved at modificere indholdet af strengen i den sidste linje:

```
int i = 42;
long l = 56;
float f = 3.14159;
double d = 3.14159*10;
System.out.printf("i=%d l=%,4d f=%f d=%6.2f", i, l, f, d);
```

4 Booleans

4.1 Definition

Hvad er en boolean?

4.2 Oprettelse

Hvilke operatorer kan man bruge til at få et *expression* til at evaluere til en boolsk værdi?

4.3 Købsbeslutning

Betragt følgede kodeudtræk:

```
double price = 599.95;
double budget = 1000.0;
boolean requiredReading = true;
boolean shouldBuy = price < budget && requiredReading;
```

Forklar sidste linje og fokuser på:

- I hvilken rækkefølge bliver hvad udregnet?
- Hvilke værdier (navngivne eller ej) udføres de enkelte operatorer på?
- Hvilke typer har disse værdier?
- Hvad repræsenterer variablen `shouldBuy`?

4.4 Terninger

Skriv et program, hvori

1. Værdien af et terningeslag er gemt i en variabel ved navn `dice`.
   - Hvilken type giver det mening at erklære variablen som?
   - Vælg selv en specifik værdi.
2. Opret en boolsk variabel og tildel den en værdi der repræsenterer hvorvidt værdien fra variablen `dice` er lige og større end 3.
3. Udskriv den værdien af denne boolske variabel.

5 Branches

5.1 Epoch

Skriv et program, hvori

1. I tager udgangspunkt i opgaven fra afsnit 3.4.
2. I erklærer en variabel hvis værdi repræsenterer et antal sekunder siden nytår.
3. På baggrund af værdien af denne variabel udregner I hvilken måned og hvilken dag der er tale om (I kan gå ud fra at alle måneder er 30 dage lange).
4. Udskriv "Det er jul!" hvis det er tilfældet (det er jul den 24. December).

5.2 Epoch Diff

Udvid opgaven fra afsnit 5.1 til – hvis man skal vente – at udskrive hvor lang tid man skal vente på at det er jul.

5.3 Juleudsalg

Skriv et program, hvori

1. En variabel oprettes (deklareres) og initialiseres til værdien 21816000. Dette tal repræsenterer et antal sekunder siden nytår (alle måneder antages at være 30 dage lange).
2. En anden variabel indeholder en pris på 599,95 dkr.
3. Der skal gives et 30% tilskud hvis det er Jul. Find selv på en fornuftig definition af hvornår det er Jul.

4. Udregn den gældende pris (eventuelt tilskud medregnet) og gem denne i en variabel.
5. Udskriv denne variabel.
6. Sørg for at teste den logik I har skrevet ved at prøve at tildele den første variable forskellige andre værdier. Hvilke værdier vil være fornuftige at teste?

5.4 Ferie

Undervisningskalenderen fortæller os at der (blandt andet) er følgende ferier:

- ***Efterårsferie*** Oktober
- ***Juleferie*** December
- ***Påskeferie*** April
- ***Sommerferie*** Juli + August

Skriv et program, hvori

1. Et månedsnummer gives via en variabel.
2. Afhængigt af indholdet af denne variabel udskrives en feries navn (hvis der er ferie i måneden) eller "Hårdt arbejde" (hvis der ikke er)

6 Loops

6.1 Celcius til Fahrenheit

Skriv et program, hvori

- Der udskrives en tabel af matchende Celcius og Fahrenheit værdier.
  - Formel: $T_F = 32 + \frac{9}{5}T_C$
- Der skal være ét sæt matchende værdier per linje.
- Listen skal starte med -5°C og slutte ved 40°C.
- Listen skal have én linje for hver 0,5°C.

6.2 Celcius til Fahrenheit i Omvendt Rækkefølge

Omskriv programmet fra opgave 6.1 til at vende rækkefølgen om sådan at første linje udskriver 40°C og sidste -5°C.

6.3 Celcius til Fahrenheit Alternativer

Lav to andre udgaver af programmet fra opgave 6.1, hvori loopet omskrives til hver af de to resterende typer af loops.

6.4 Areal af Cirkler

Skriv et program der udregner og udskriver arealet ($\pi \cdot r^2$) af tre cirkler med radius på hhv. 1, 3 og 5.

6.5 Længden af en Måned

Skriv et program, hvori

1. En måneds nummer gemmes i en variabel. Denne variabel skal fungere som input til jeres program.
   - Hvilket navn ville være passende for denne variabel?
   - Hvilken type ville være passende for denne variabel?
2. Skriv noget kode der på baggrund af denne variabel bestemmer hvor mange dage der er i denne måned (vi antager at det ikke er skudår).
3. Skriv dette tal ud på skærmen.
4. Overbevis jer selv om at jeres kode er korrekt.

6.6 Primtal

Skriv et program, der udregner alle primtal under 1.000.000, og udskriver det største.

Hints (se bort fra dette hvis I er friske på en udfordring):

- Benyt en del-og-hersk strategi, hvor I ser opgaven som tre delopgaver:
  - Gennemløb alle positive heltal under 1.000.000. Har vi ikke en konstruktion der kan det?
  - Bestem om et givent positivt heltal er et primtal.
  - Udprint et heltal (hvis det altså er et primtal).
- Et positivt heltal er et primtal hvis og kun hvis der ikke er andre heltal end 1 der går op i det.
- For at bestemme om et givent positivt heltal er et primtal kan I endnu engang benytte del-og-hersk:
  1. Opret en variable af typen `boolean` ved navn `is_prime` som er True.
  2. Gennemløb alle heltal fra og med 2 til (men ikke med) 1.000.000.

3. For hvert af disse tal skal I checke om dette tal går op i det potentielle primtal. Gør de det, så sættes `is_prime` til False. Men hvordan undersøger man om ét tal går op i et andet?
   - Man prøver det da!
   - Hvis en heltalsdivision går op er det ikke nogen rest og en modulo operation (via operatoren `%`) giver nul.
   - Alternativt kan man udnytte at en heltalsdivision kommer til at foretage en afrunding hvis divisionen ikke går op sådan at $(a/b) \cdot b \neq a$.
4. Herefter repræsenterer `is_prime` en sandhedsværdi for hvovidt et tal er et primtal.

7 Arrays

7.1 Definition

Hvad er et array?

7.2 Anvendelse

Hvornår giver det mening at bruge arrays?

7.3 Type af Indhold vs Type af Array

Hvad er sammenhængen mellem typen af et array og de data som det kan indeholde?

7.4 Størst i Array

Skriv et program, der finder det største tal i et array af typen `int[]`, og udskriver indeks for dette tal. Find selv på noget passende indhold til dette array.

7.5 Deklaration af Størrelse

Hvilket syntaktiske element bruges – ved oprettelse af et array – til at erklære hvor mange elementer der skal være plads til?

7.6 Multiplikationstabel

Skriv et program, hvori

1. En heltallig variabel ved navn `size` initialiseres til en værdi under 30. I vælger selv den konkrete værdi.

2. Der oprettes et array med en længde som modsvarer værdien af `size`.
3. Arrayet fyldes op med 3-tabellen. Altså, elementet med indeks n skal have værdien $3 \cdot n$.
4. Et (eller flere) velvalgt(e) element(er) udskrives for at verificere korrektheden.

7.7 Sudoku Plade

Hvordan ville du repræsentere en sudoku[1] plade i Java?

Hvordan hænger denne datastruktur sammen i hukommelsen?

7.8 Areal af Cirkler

Skriv et program der udregner og udskriver arealet ($\pi \cdot r^2$) af tre forskellige cirkler med radius 1, 3, og 5.

7.9 Daglige Differencer

Skriv et program, der givet 7 dagstemperaturer udregner og udskriver temperaturdifferencen mellem alle to på hinanden følgende dage (dvs. Tirsdag-Mandag, Onsdag-Tirsdag ... Søndag-Lørdag).

Dagstemperaturerne kunne være:

- Mandag: 21.5
- Tirsdag: 23.7
- Onsdag: 19.6
- Torsdag: 22.5
- Fredag: 25.3
- Lørdag: 21.7
- Søndag: 18.9

7.10 Længden af en Måned

Skriv et program, der givet en måneds nummer udskriver antallet af dage i denne måned (se bort fra skudår).

[1] `https://en.wikipedia.org/wiki/Sudoku`

7.11 Primtal

Skriv et program der udregner alle primtal under 1.000.000 og udskriver det største.

Gør dette ved at implementere Eratosthenes Si[2].

Kvadratroden af `i` udregnes som `java.lang.Math.sqrt(i)`.

7.12 Kalender

Skriv et program, hvori

1. En variabel initialiseres til at være et array der indeholder antallet af dage i hver af de 12 måneder i et normalt år. Det første element vil da indeholde antallet af data i Januar.
2. En anden variabel initialiseres til at være et array der indeholder antallet af dage i hver af de 12 måneder i et skudår.
3. Gennemløb årene 2000 til 2020.
4. Brug en ny variabel af samme type som de to forregående til at holde styr på hvilket array der er korrekt for det aktuelle år.
    - **Hint:** Vi kan i denne opgave tillade os at forsimple skudårsreglerne til at det er skudår hvis 4 går op i årstallet.
5. For hvert år udskrives indholdet af det array som den sidste variabel peger på.

7.13 Kalender Prettyprinting

Lav et program, hvori

1. Der oprettes en datastruktur til at holde en kalender der spænder ét år. Man skal kunne indeksere ind med en dato (måned + dag) og få en dag (fx Mandag).
    - Hvad er typen af denne datastruktur?
    - Hvordan initialiseres en variabel af denne type?
    - Hvordan kan man sørge for at indholdet er korrekt?
2. Udskriv denne datastruktur på en "pæn" måde.

[2] `https://en.wikipedia.org/wiki/Sieve_of_Eratosthenes`

7.14 Sudoku Checker

Skriv et program, hvori

- En variabel initialiseres til at indeholde en udfyldt sudokuplade.
- Skriv kode der vurderer om pladen representerer en korrekt løsning.
  - **Hint 1:** Dette er tilfældet når alle følgende ting er sande:
    * Samtlige felter er udfyldte.
    * Der er ingen række med to ens felter. Alternativt: Alle tallene 1-9 eksisterer i samtlige rækker.
    * Der er ingen søjle med to ens felter. Alternativt: Alle tallene 1-9 eksisterer i samtlige søjler.
    * Der er ingen 3x3 gruppe med to ens felter. Alternativt: Alle tallene 1-9 eksisterer i samtlige 3x3 grupper.

    Hint 2: Hvis man skal undersøge om tallene 1-9 eksisterer kunne man jo oprette et `boolean[9]` array der repræsenterer om tallene er fundet, initialisere dette til kun at indeholde `false` værdier, gennemløbe alle tal som er i et felt og sætte indekset med dette nummer (minus 1) til `true`. Er der nogen `false` værdier tilbage er der et manglende tal og sudokuen er ikke korrekt løst.
- Udskriv resultatet af denne vurdering.

8 Metoder

8.1 Anvendelse

Hvad bruges en metode til?

8.2 Mangel på Returværdi

Hvordan fortæller man at en metode *ikke* returnerer en værdi?

8.3 Sudoku Prettyprinter

Følgende datastruktur repræsenterer en udfyldt sudoku plade:

```
int[][] puzzle = {
    {7, 3, 6, 4, 5, 2, 9, 8, 1},
    {1, 9, 8, 6, 3, 7, 4, 5, 2},
    {4, 2, 5, 9, 8, 1, 3, 7, 6},
    {3, 6, 4, 5, 2, 8, 1, 9, 7},
    {9, 5, 2, 7, 1, 4, 6, 3, 8},
    {8, 1, 7, 3, 9, 6, 2, 4, 5},
```

```
    {2, 8, 9, 1, 7, 3, 5, 6, 4},
    {6, 7, 3, 2, 4, 5, 8, 1, 9},
    {5, 4, 1, 8, 6, 9, 7, 2, 3},
};
```

Skriv et program, hvori

1. Ovenstående datastruktur er defineret.
2. En metode er defineret der som parameter tager en sådan struktur og skriver den ud på skærmen.
   - Hvilken prototype (i.e., returværdi og signatur) skal denne metode have?
   - Skal man udprinte en række af gangen eller en søjle af gangen?
   - **Hint:** Metoden `System.out.print` gør det samme som `System.out.println` men undlader at bryde linjen.
3. En `main` metode kalder denne metode.

8.4 Sum

Skriv en metode, der lægger to heltal sammen. Skriv derudover et program der viser hvordan denne metode skal bruges.

8.5 Egen Kvadratrod

Skriv et program, hvori

1. En metode udregner kvadratroden af en `double` med fx 7 decimale cifre.
   - **Hint:** Prøv jer frem for hvert ciffer, og brug et loop til at iterere over cifrene 0.000000001 til 1000000000.
2. Der er en `main` metode som demonstrerer denne metode.

8.6 Matematisk Ækvivalent

Hvilken matematiske konstruktion kan repræsenteres rigtigt godt med en metode, og hvorved adskilder de to sig fra hinanden?

8.7 Diskriminanter og Rødder

Skriv et program, hvori

1. En metode kan udregne en diskriminant[3] ud fra parametrene a, b og c.
2. En metode tager parametrene a, b og c fra et andengradspolynomie[4], og

[3] `https://en.wikipedia.org/wiki/Discriminant`
[4] `https://en.wikipedia.org/wiki/Quadratic_function`

returnerer et array af rødder.

- **Hint:** I kan bruge `java.lang.Math.sqrt(9.0)` i til at udregne $\sqrt{9}$.

3. En `main` metode som kalder den sidste metode og udskriver resultatet.

8.8 Fakultet

Skriv et program, hvori

1. En metode udregner fakultet[5] (e.g., fac(4)=4*3*2*1) uden brug af et loop.
   - **Hint:** Dette kan gøres ved hjælp af *rekursion*.
2. En `main` metode som kalder denne metode og udskriver resultatet.

8.9 Cirkler i Tal

Skriv et program der udregner og udskriver både arealet ($\pi \cdot r^2$) og omkredsen ($2 \cdot \pi \cdot r$) af tre cirkler med radius på hhv. 1, 3 og 5.

9 Grundlæggende Programmering

9.1 Sudoku Løser

Lav et program der løser en Sudoku.

Bemærk: Dette er en svær opgave, som kræver at der tænkes kreativt.

9.2 Game of Life

Lav en implementering af John Conway's Game of Life[6].

Bemærk: Dette er en svær opgave, som kræver at der tænkes kreativt.

9.3 8 Dronninge Problemet

Lav en implementering af 8-dronninge-problemet[7].

Opgaven er at lave et program der finder én (eller alternativt *alle*) opstillinger hvor 8 dronninge brikker er placeres på et skakbræt med 8 tern på hver led, uden at nogen dronning kan "slå" nogen anden dronning. En dronning kan slå alle brikker på samme række, søjle eller diagonal.

Bemærk: Dette er en svær opgave.

[5] `https://en.wikipedia.org/wiki/Factorial`
[6] `https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life`
[7] `https://en.wikipedia.org/wiki/Eight_queens_puzzle`

Hint: Prøv at gennemløbe alle konfigurationer af dronningepositioner og check hvilke der er gyldige løsninger.

Bonus: 8-dronninge-problemet kan generaliseres til et n-dronninge-problem.

10 Objekter

10.1 Kunder

Lav et program efter følgende opskrift:

1. Opret et nyt projekt med en klasse indeholdende en `main`-metode.
2. Opret, i dette projekt, en ny klasse ved navn `Customer`.
3. I klassen `Customer` tilføjes følgende attributter:
   - En attribut ved navn `name` af typen `String`.
   - En attribut ved navn `id` af typen `int`.
   - En attribut ved navn `balance` af typen `double`.
4. I klassen `Customer` tilføjes to Constructors:
   - Én Constructor der tager et navn og et id som argument og bruger disse værdier til at sætte værdierne af de tilsvarende attributter, og som derudover sætter balancen til 0.
   - Én Constructor der tager et navn, et id og en balance som argumenter og bruger disse værdier til at sætte værdierne af de tilsvarende attributter.
5. I klassen `Customer` tilføjes tre metoder:
   - En metode ved navn `deposit` med `void` return type, der tager et argument ved navn `amount` af typen `double`. Metoden skal lægge værdien af `amount` til værdien af attributten `balance` og gemme denne værdi i attributten `balance`.
   - En metode ved navn `withdraw` med `void` return type, der tager et argument ved navn `amount` af typen `double`. Metoden skal trække værdien af `amount` fra attributten `balance` hvis og kun hvis `balance` er større end `amount`, og gemme den nye værdi i attributten `balance`.
   - En metode ved navn `getBalance`, der har `double` som return type, og som returnerer værdien af attributten `balance`.
6. I klassen der indeholder `main`-metoden skal der i denne `main`-metode oprettes en variabel ved navn `aCustomer` af typen `Customer`.
7. `aCustomer` skal tildeles (reference til et nyt) `Customer` objekt, ved at lave en tildeling hvor der på højresiden oprettes et nyt `Customer`-objekt ved at kalde constructoren på `Customer`. Værdien af argumenterne bestemmer i selv.

- **Hint:** "`new`" bruges for at kalde en constructor i klasser.

8. Sæt penge ind på `Customer`-objektet ved at kalde metoden `deposit` med et beløb I selv bestemmer.
9. Træk penge ud af `Customer`-objektet ved at kalde metoden `withdraw` med et beløb I selv bestemmer, men som er mindre end beløbet I satte ind.
10. Udskriv værdien af `balance` på `Customer`-objektet, ved at kalde instansmetoden `getBalance` og kalde `System.out.println` med returværdien fra dette metodekald som argument.

10.2 Kunde Database

Udvid resultatet fra opgave 10.1 med følgende:

1. Opret en ny klasse ved navn `CustomerDatabase`.
2. I klassen `CustomerDatabase` tilføjes følgende attribut:
   - En attribut ved navn `customers` af typen `Customer[]`.
3. I klassen `CustomerDatabase` tilføjes én constructor, der ikke tager nogen argumenter men som initialiserer attributten `customers` med et tomt `Customer`-array der har 10 pladser.
4. I klassen `CustomerDatabase` tilføjes følgende metoder:
   - En metode der tager et `Customer`-objekt som argument, og gemmer det i arrayet `customers`. Hvad ville være passende navne for denne metode, og argumentet til denne metode?
   - En metode der tager en `int` som argument, og som gennemløber `customers`-arrayet for at kontrollere om `id` attributten for de enkelte elementer er lig med værdien at argument som metoden er blevet kaldt med. Hvis det er tilfældet, skal objektet fjernes fra arrayet. Ellers skal der ikke gøres noget. Hvad ville være passende navne for denne metode, og argumentet til denne metode?
     - **Hint:** `customers`-arrayet indeholder referencer til `Customer` objekter. Værdien `null` kan bruges til at indikere at en reference er ugyldig.
   - En metode der returnerer (alt indholdet af) `customers`-arrayet. Hvad ville være et godt navn for denne?
   - En metode der udskriver navne på alle `Customer`-objekter i `customers`. Hvad ville være et godt navn for denne?
5. Find selv på passende ting at implementere i programmets main-metode, som tester funktionaliteten af din `CustomerDatabase`-klasse.

10.3 Farver

De farver vi opfatter er resultatet at vores hjernes fortolkning af signaler fra receptorer i vores øjne. Disse receptorer er følsomme overfor lys indenfor en en lille del af lysets spektrum. Så denne måde opfanger nogle af disse receptorer røde nuancer, andre grønne nuancer og andre igen blå. Nogle kvinder har faktisk en ekstra type receptorer der er følsomme indenfor et fjerne område.

I en computer repræsenteres farver typisk *additivt* ved hjælp af RGB tripletter; altså en rød værdi, en grøn værdi og en blå værdi. Oftest bruges én byte til at repræsentere hver af disse. Det giver $2^8 = 256$ forskellige nuancer per farvekanal og $(2^8)^3 = 16777216$ farver. Dette er grunden til at man ofte hører referencer til "16, 7 millioner farver", eller bare "millioner af farver". Disse 3 komponenter – rød, grøn og blå – udspænder et *farverum*. Det kan illustreres som et terning i et 3D koordinatsystem hvor hver akse er navngivet efter en grundfarve.

Men RGB er blot én måde at representere farver på. Andre farvemodeller har dog styrker. I denne opgave ser vi på HSV; Hue-Saturation-Value. Der er igen tale om en triplet. Hue værdien repræsenterer farvens tone, saturation (da: saturering) værdien repræsenterer hvor "farverig" tonen er, og value værdien repræsenterer lysheden. HSV illustreres ofte som en cylinder hvor afstanden fra centrum er saturation, afstanden fra den ene ende er value og hue er vinklen målt langs et tværsnit. Denne model er attraktiv af flere grunde; eksempelvis:

- En farves kontrastfarve fås ved at lægge 180° til hue værdien.
- Der findes teorier om hvordan man fx finder 3 farver der er "pæne sammen". Dette er eksempelvis tilfældet hvis farverne A og B er har en hue værdi der er forskudt k grader i modsat retning i forhold til farven C, og ellers er ens.
- Hvis man ønsker n farver der er lette at skelne fra hinanden kan man fordele dem ligeligt på hue cirklen.

I denne opgave skal vi konvertere mellem RGB og HSV ved at lave et program efter følgende opskrift:

1. Skriv en ny klasse ved navn `RGB` der repræsenterer en farve formuleret efter RGB komponenterne.
   - Klassen skal have 3 `int` attributter med getter metoder[8]: `r`, `g` og `b`.
   - Klassen skal have en `asHSV` metode der opretter et `HSV` objekt der repræsenterer samme farve. Algoritmen kan ses i figur 1.
   - Klassen skal have en passende constructor.
   - Klassen skal have en `toString` metode der returnerer en `String` værdi der repræsenterer instansvariablene.
2. Skriv en ny klasse ved navn `HSV` der repræsenterer en farve formuleret efter HSV komponenterne.

[8] En getter metode for attributten `r` af typen `int` er en parameterløs metode der hedder `getX` med returtypen `int` der returnerer værdien af attributten `r`

- Klassen skal have 3 `int` attributter med getter metoder: `h`, `s` og `v`.
- Klassen skal have en `asRGB` metode der opretter et `HSV` objekt som repræsenterer samme farve. Algoritmen kan ses i figur 2.
- Klassen skal have en passende constructor.
- Klassen skal have en `toString` metode der returnerer en `String` værdi der repræsenterer instansvariablene.

3. Opret en ny klasse ved navn `Test` med en `main` metode der tester funktionaliteten:
   - Opret et array med nogle hardkodede RGB objekter.
   - Iterer igennem dette array. For hver iteration:
     - Print objektets attributter ud.
     - Konvertér objektet til et objekt af typen `HSV`. Print dette objekts attributter ud.
     - Konvertér det nye objekt tilbage til et **nyt** objekt af typen `RGB`. Print dette objekts attributter ud.
4. Kør denne `main` metode og verificér at resultaterne er korrekte. Til dette formål kan I fx finde en RGB-HSV converter på nettet.
   - **Bemærk:** Sammenligning af HSV værdierne er ikke altid trivielt. Der er mange standarder for skalering. H skaleres typisk til 360 (da det måles i grader) og både S og V skaleres typisk til 100 (da det er procenter). Sørg for at konvertere disse værdier til at matche den repræsentation som jeres converter bruger. Eksempelvis kan I for hue gange med 360 og dividere med 256 hvis i ønsker at ændre skalering fra 256 til 360.

11 Arv

11.1 Lagersystem

Delopgave 1

Nedenstående UML diagram viser dele af et lagersystem, hvor arv indgår. I skal implementere klasserne fra diagrammet som de er vist:

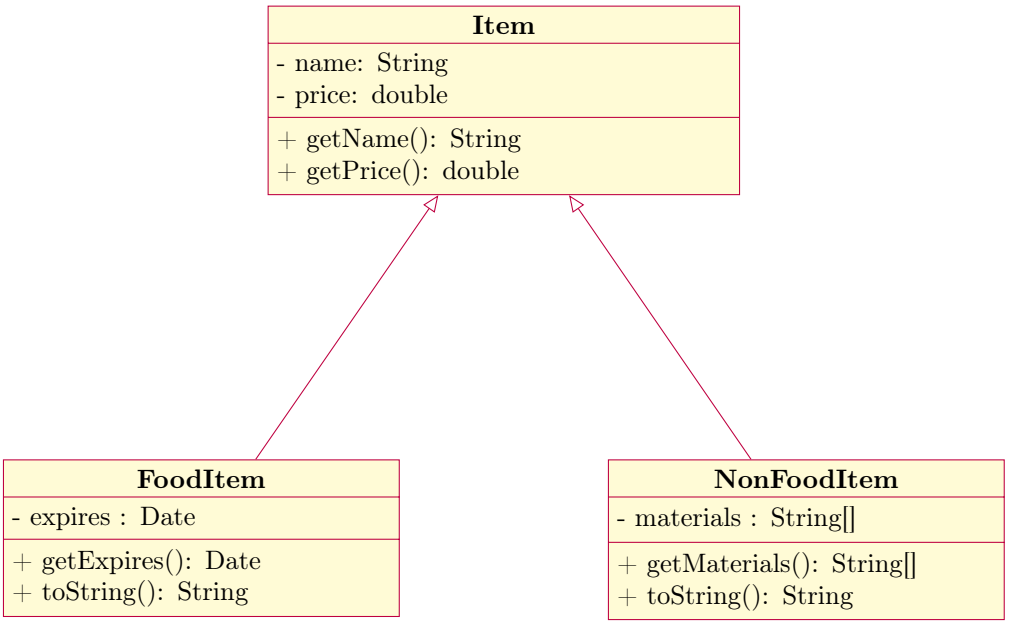
Inputs: r, g, b (alle heltal ≥ 0 og < 256)
Outputs: h, s, v (alle heltal ≥ 0 og < 256)
$cmin \leftarrow \mathbf{min}(r, g, b)$
$cmax \leftarrow \mathbf{max}(r, g, b)$
$diff \leftarrow cmax - cmin$
if $cmax = cmin$ **:**
    $h \leftarrow 0$
else if $cmax = r$ **:**
    $h \leftarrow (60 \cdot \frac{g-b}{diff} + 360) \ \mathbf{mod}\ 360$
else if $cmax = g$ **:**
    $h \leftarrow (60 \cdot \frac{b-r}{diff} + 120) \ \mathbf{mod}\ 360$
else if $cmax = b$ **:**
    $h \leftarrow (60 \cdot \frac{r-g}{diff} + 240) \ \mathbf{mod}\ 360$
$h \leftarrow h \cdot \frac{255}{360}$
if $cmax = 0$ **:**
    $s \leftarrow 0$
else:
    $s \leftarrow \frac{diff}{cmax} \cdot 255$
$v \leftarrow cmax \cdot 255$

Figure 1: Algoritme for at konvertere fra RGB til HSV.

Inputs: h, s, v (alle heltal ≥ 0 og < 256)
Outputs: r, g, b (alle heltal ≥ 0 og < 256)
$h \leftarrow h \cdot 360/256$
$c \leftarrow v \cdot s$
$m \leftarrow v - c$
$x \leftarrow c \cdot \big(1 - \text{abs}\big(\big(\frac{h}{60} \bmod 2\big) - 1\big)\big)$

$$(r, g, b) \leftarrow \begin{cases} (c+m, x+m, m) & \text{if } 0 \leq h < 60 \\ (x+m, c+m, m) & \text{if } 60 \leq h < 120 \\ (m, c+m, x+m) & \text{if } 120 \leq h < 180 \\ (m, x+m, c+m) & \text{if } 180 \leq h < 240 \\ (x+m, m, c+m) & \text{if } 240 \leq h < 300 \\ (c+m, m, x+m) & \text{if } 300 \leq h < 360 \\ (m, m, m) & \text{otherwise} \end{cases}$$

Figure 2: Algoritme for at konvertere fra HSV til RGB.



`toString`-metoderne i hhv. `FoodItem` og `NonFoodItem` skal annoteres med `@Override` da de overrider metoden fra `Object`-klassen (ikke vist på diagrammet).

`toString`-metoden i `FoodItem` skal returnere navn, pris og udløbsdato som en `String`.

`toString`-metoden i `NonFoodItem` skal returnere navn, pris og listen af materialer som en `String`.

Delopgave 2

Lav i jeres `main`-metode et array der kan indeholde 10 `FoodItem`-objekter. Fyld hver plads i arrayet med et `FoodItem`-objekt vha. et loop.

I et andet loop skal i kalde `toString` metoden på hvert af de `FoodItem`-objekter der ligger i jeres array og udskrive det vha. `System.out.println(String)`.

Delopgave 3

Gør det samme som i forrige delopgave, men denne gang for `NonFoodItem`-objekter (undlad at overskrive koden fra forrige delopgave, så dit program gør det for begge typer af objekter).

12 Navngivning

12.1 Killinger

Skriv et program hvori:

- Der er en `Kitten` (da: killing) klasse.

- Objekter af denne type skal have en `cuteness` (da: nuttethed) attribut af typen `double` der initialiseres igennem en constructor.
- En klassevariabel (altså, en statisk variabel) ved navn `count` og type `int` initialiseres til værdien 0 og inkrementeres hver gang en killing oprettes.
- Tilføj en `main` metode der – via et `for`-loop – opretter en række `Kitten` objekter med forskellig nuttethed og derefter udskriver værdien af `count`.
- Kør programmet og verificer at `count` har den værdi som I ville forvente.

12.2 Addition

Skriv et program efter følgende opskrift:

- Opret et nyt projekt med en klasse der hedder `Adder`.
- Opret en statisk metode med navn `solve` der tager to `int` parametre og returnerer en `String`. Denne metode skal konstruere en streng der udtrykker hvad summen af de to parametre er. Dette kunne fx være `"3 + 5 = 8"`.
- Opret en statisk metode med samme navn der tager to `double` parametre og returnerer en `String`. Denne metode skal konstruere en streng der udtrykker hvad summen af de to parametre er. Dette kunne fx være `"3.14 + 5.12 = 8.26"`.
- Opret nu en `main` metode der:
  1. Kalder `solve` med to `int` værdier og skriver resultatet ud på skærmen.
  2. Kalder den med to `double` værdier og skriver resultatet ud på skærmen.
- Oversæt og udfør programmet. Verificér at resultatet er korrekt.
- Hvad sker der hvis man prøver at kalde `solve` med en `int` og en `double`, og hvorfor sker dette?

12.3 Operatorer

Udfør følgende program:

```
class Operator {
    public static int operator (int a, int b, boolean c) {
        return a+b;
    }
    public static int operator (int a, int b, char c) {
        return a-b;
    }
```

```
    public static void main (String[] args) {
        for (int a=0 ; a<5 ; a++) {
            for (int b=0 ; b<5 ; b++) {
                System.out.println(a+" (?1?) "+b+" = "+operator(a, b, true));
                System.out.println(a+" (?2?) "+b+" = "+operator(a, b, false));
                System.out.println(a+" (?3?) "+b+" = "+operator(a, b, 'a'));
                System.out.println(a+" (?4?) "+b+" = "+operator(a, b, 'b'));
                System.out.println(a+" (?5?) "+b+" = "+operator(a, b, '.'));
            }
        }
    }
}
```

Betragt uddata (i.e., dét der er blevet printet ud på skærmen) og forklar hvad der foregår. Specifikt:

1. Hvad repræsenterer `(?1?)`, `(?2?)`, `(?3?)`, `(?4?)` og `(?5?)`?
2. Hvad er forholdet mellem disse og den 3. parameter til `operator` kaldene?
3. Hvad er den vigtige del af den 3. parameter?
4. Er dette en fornuftig grænseflade for `operator`?

12.4 Scope

I følgende program, hvor kan (og bør) variablene `i`, `d`, `tmp`, `sum` og `bonus` deklareres?

```
class Scope {
    // location 0
    bonus = 42;
    // location 1

    static int doubler (int value) {
        // location 2
        d = value * 2;
        // location 3
        return d;
        // location 4
    }

    // location 5

    public static void main (String[] args) {
        // location 6
        sum = 0;
        // location 7
        for (/* location 8 */ i=0 ; /* location 9 */ i<100 ; /* location 10 */ i++) {
```

```
            // location 11
            tmp = doubler(i);
            // location 12
            sum += tmp;
            // location 13
        }
        // location 14
        System.out.println(sum+bonus);
        // location 15
    }

    // location 16
}
```

Hvad sker der hvis en variabel deklareres to gange?

12.5 Indkapsling

Betragt følgende program:

```
class Circle {
    double x, y;
    double r;

    public Circle (double x, double y, double radius) {
        this.x = x;
        this.y = y;
        this.r = radius;
    }
}

class TestCircle {
    public static void main (String[] args) {
        Circle c = new Circle(1.24, 2.83, 12.7);
        System.out.println("x="+c.x+" y="+c.y+" r="+c.r);
        c.r *= 1.37;
        c.x += 0.65;
        System.out.println("x="+c.x+" y="+c.y+" r="+c.r);
    }
}
```

Følg følgende instruktioner:

1. Beskriv sprogligt hvad programmets `main metode` gør. Fokusér på detaljerne.
2. Lav en kopi af dette program. I kommer ikke til at arbejde videre på den gamle udgave.

3. Opdater den nye kopi sådan at attributten `r` (der repræsenterer en radius) erstattes med attributten `d` (der repræsenterer en diameter).
4. Skriv ned hvilke ændringer I lavede på denne kopi?
5. Lav en ny kopi af dette program. I kommer ikke til at arbejde videre på den gamle udgave.
6. Foretag indkapsling af alle attributter. Indkapsling indebærer:
   - At attibutten erklæres `private`.
   - Af der tilføjes de relevante getters og setters.
7. Lav en ny kopi af dette program. I kommer ikke til at arbejde videre på den gamle udgave.
8. Opret en ny klasse ved navn `Coordinate` der repræsenterer et (x, y) koordinat.
9. Opdater det nye program til at benytte `Coordinate` klassen.
10. Skriv ned hvilke ændringer I lavede på denne kopi?
11. Trin 4 og trin 10 repræsenterer lignende ændringer med forskelligt udgangspunkt. Forskellen er om attributterne er indkapslede. Sammenlign nu noterne fra disse trin. Hvad gør indkapsling når i modificerer repræsentationen af `Circle`?

13 Standardbibliotek

13.1 Date

Skriv et program der

1. Opretter et enkelt `Date` objekt.
2. Ved hjælp af `setTime` metoden sættes dette objekts "tid" først til 10^3, dernæst til 10^4, så til 10^5 og til sidst til 10^6.
3. For hver af disse tider udskrives det tidspunkt objektet repræsenterer til skærmen ved hjælp af blandt andet `toString` metoden.
4. Verificer at dette giver mening.

13.2 Tidstagning

Skriv et program, hvori

1. Der er én klasse som hedder `Timing`.

2. Der er i denne klasse en statisk metode som hedder `fun`. Denne metode tager to `double` argumenter (`x` og `y`) og returnerer en `double`. Denne metode er rekursivt defineret således at:

   - Hvis `y` ≤ 1 så returnerer `pow` værdien af variablen `x`.
   - Ellers ($y \neq 1$) returnerer metoden værdien af følgende udtryk: `fun(x,y-1) * fun(x,y-1)`.

3. En `main` metode der for `y = 1.0000001` gennemløber alle `x`-værdier fra og med 1 til og med 32. For hver værdi kombination af `x` og `y` udregnes og udskrives `fun(x,y)` sammen med den tid udregningen tager.

   - Den statiske metode `System.currentTimeMillis()` returnerer et tidsstempel målt i millisekunder i form af en `long`.
   - Ved at trække et sådant tidsstempel fra et andet får man den tid der er gået imellem dem.

Udregner denne implementation `fun(x,-1)` korrekt (begrund dit svar)?

Er dette ellers en god implementation af `fun(x,y)`?

14 Programudvikling

14.1 Indskrivningssystem

Under forelæsningen blev I introduceret for følgende programspecifikation:

The enrollment system contains a list of students and courses. It can display the list of students which are enrolled in a particular course. It is possible to enroll students to a course and remove them from a course.

På baggrund af dette blev der udledt følgende krav:

- ***Funktionelle krav:***

ID	*Navn*	*Beskrivelse*
F01	Vis liste over studerende	Det skal være muligt at få vist en list eaf studerende
F02	Vis liste over kurser	Det skal være muligt at få vist en liste af kurser
F03	Tilføj og fjern tilknytning til kurser	Det skal være muligt at associere studerende til et kursus og at fjerne eksisterende associationer

- ***Non-funktionelle krav:***

ID	*Navn*	*Beskrivelse*
NF01	Kapacitet	Systemet skal have plads til 10 kurser
NF02	Interoperabilitet	Systemet skal kunne integreres med Blackboard

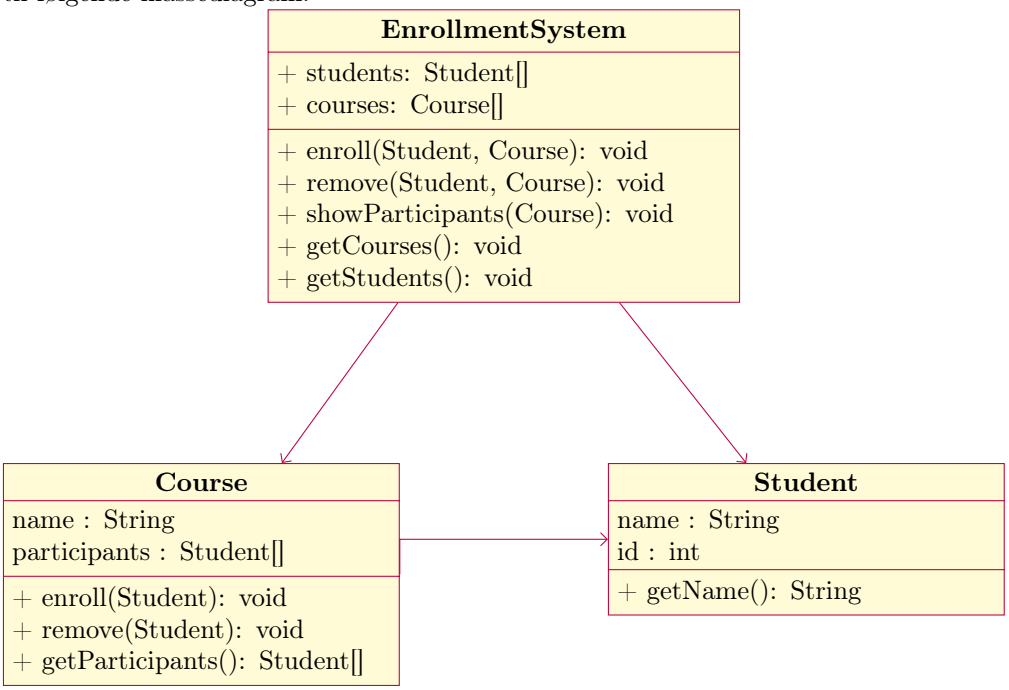
Resultatet af en noun-verb analyse af programspecifikationen:

The `enrollment` system `contains` a `list` of `students` and `courses`. `It` can `display` the `list` of `students` which `are` enrolled in a particular `course`. It is possible to `enroll` `students` to a `course` and `remove` `them` from a `course`.

Dette har ledt til cølgende CRC kort:

EnrollmentSystem	
Responsibilities	*Collaborators*
• Contains list of `Course`-objects • Contains list of `Student`-objects • Can enroll `Student`-objects in `Course`-objects • Can remove `Student`-objects from `Course`-objects • Can display `Student`-objects enrolled in any given `Course`-object	• Student • Course

Baseret på dette – og tilsvarende for de resterende klasser – er man nået frem til følgende klassediagram:



Dette har resulteret i følgende implementation:

```
public class Student
{
    String name;
    int    id;
```

```
    Student (String name, int id) {
        this.name    = name;
        this.id      = id;
    }

    public String getName () {
        return name;
    }
}

public class Course
{
    String name;
    Student[] participants;
    int    id;

    Course (String name) {
        this.name = name;
        this.participants = new Student[10]; // NOTE: This constant is BAD!
    }

    public void enroll (Student student) {
        for (int i=0 ; i<participants.length ; i++) {
            if (participants[i]==null) {
                participants[i] = student;
                return;
            }
        }
    }

    public void remove (Student student) {
        for (int i=0 ; i<participants.length ; i++) {
            if (participants[i]==student) {
                participants[i] = null;
            }
        }
    }

    public Student[] getParticipants () {
        // count number of entries
        int count = 0;
        for (Student student: participants) {
            if (student!=null) {
                count++;
            }
        }

        // make a copy
        Student[] result = new Student[count];
        for (Student student: participants) {
```

```
            if (student!=null) {
                result[count--] = student;
            }
        }

        return result;
    }
}

public class EnrollmentSystem
{
    Student[] students;
    Course[]  courses;

    public void enroll (Student student, Course course) {
        course.enroll(student);
    }

    public void remove (Student student, Course course) {
        course.remove(student);
    }

    public void showParticipants (Course course) {
        for (Student student: course.getParticipants()) {
            System.out.println(student.getName());
        }
    }

    public void getCourses () {
        System.out.println("void for a getter?");
    }

    public void getStudents () {
        System.out.println("void for a getter?");
    }
}
```

Med udgangspunkt i ovenstående, løs følgende opgaver:

1. Den ovenstående programspecifikation har den åbenlyse mangel at den ikke beskriver nogen mulighed for at tilføje eller fjerne hverken kurser eller studerende. Opdatér specifikationen så den beskriver denne funktionalitet.
2. Opdatér kravspecifikationen så den indeholder krav der er dækkende for den nye programspecifikation.
3. Udfør en noun/verb analyse på den opdaterede programspecifikation, og opdater listen over metoder og/eller klasser.
4. Opdatér CRC-kortet så de passer på den nye noun/verb analyse og kravspecifikation. Tilføj kort hvis det er nødvendigt.
5. Opdatér UML diagrammet så det afspejler jeres opdaterede CRC kort.
6. Opdatér implementeringen af programmet således at det afspejler jeres opdaterede UML diagram.

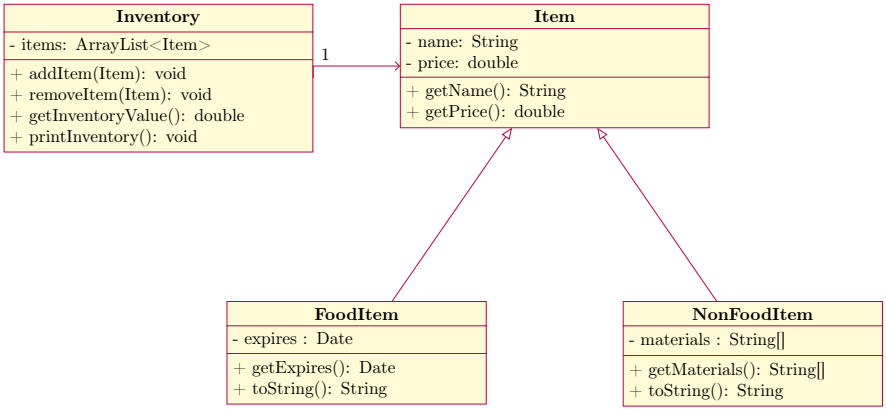
15 Polymorfi

15.1 Lagersystem

Denne opgave bygger videre på opgaven fra afsnit 11.1 hvortil en referenceløsning er givet i afsnit 30.1. Det er op til dig hvorvidt du vil arbejde videre på denne eller din egen løsning.

Delopgave 1

Du skal nu udvide implementationen så den modsvarer UML diagrammet herunder:



Metoden `getInventoryValue` skal løbe alle `Item`-objekter i listen `items` igennem, kalde `getPrice` på hvert af objekterne og lægge værdierne sammen for alle objekterne og returnere denne sum.

`printInventory` skal udskrive tekst-repræsentationen af alle objekterne i listen items ved hjælp af `System.out.println`. Igen skal der her anvendes en løkke.

`addItem` og `removeItem` skal henholdsvis tilføje til og fjerne objekter fra `items`-listen.

Delopgave 2

Opdatér din `main`-metode, så du tilføjer nogle items af både typen `FoodItem` og `NonFoodItem` til en `Items`-liste i en instans af `Inventory`.

Kald `printInventory` og `getInventoryValue` for at validére deres funktionalitet.

Delopgave 3

Attributten `materials` i `NonFoodItem` anvender et array. Erstat typen her med en `ArrayList<String>` og lav de ændringer i din kode der skal til for at den stadig fungerer efter hensigten.

16 Abstrakte Klasser og Interfaces

16.1 Lagersystem

Disse opgaver bygger videre på et løsningsforslag til opgaven fra afsnit 15.1. I vælger selv om I vil løse opgaverne på jeres egen løsning eller anvende referenceløsningen fra afsnit 34.1. Opgaverne vil anvende klassenavne fra løsningsforslaget.

Delopgave 1

For at sikre, at produkter altid instantieres som enten et `FoodItem`-objekt eller et `NonFoodItem`-objekt skal I gøre klassen `Item` abstrakt. Hvad sker der, hvis I derefter forsøger at lave et objekt at typen `Item` ved at kalde constructoren på `Item` direkte? Hvorfor er det tilfældet?

Delopgave 2

I skal nu lave et interface kaldet `Expireable`, med en metode der har følgende signatur:

```
public boolean isExpired();
```

Delopgave 3

Interfacet skal I herefter implementere i klassen `Item`. At implementere et interface vil sige, at I skal anvende `implements`-keywordet i `Item`.

Delopgave 4

Da `FoodItem` og `NonFoodItem` nedarver fra `Item`, nedarver de også `isExpired` metoden. I skal nu *override* denne metode i klassen `FoodItem`, så den udnytter attributten `expireDate` til at afgøre, om produktet er for gammelt.

Hint: `Date` typen i Java frameworket kan anvendes til at repræsentere datoer.

Delopgave 5

I `Inventory` klassen skal I implementere en `void` metode ved navn `removeExpiredFoods` der itererer igennem listen af `Item`-objekter og kalder `isExpired` på hvert af objekterne. Hvis `isExpired` returnerer `true`, skal det pågældende `Item`-objekt fjernes fra listen af `Item`-objekter.

Hint: Da man ikke må modificere i en `ArrayList` mens man itererer over den med en foreach, kan man med fordel oprette en ny `ArrayList`, indsætte de varer der skal blive, og til sidst opdatere varelagerets reference til at pege på den nye `ArrayList`.

17 Objekt-Orienteret Programmering

17.1 Gennemsnitlig Alder

Denne opgave tager udgangspunkt i følgende klasse:

```
class Person {
    double age;

    public Person (double age) {
        this.age = age;
    }
}
```

Tilføj en statisk metode der returnerer en `double` der repræsenterer den gennemsnitlige alder af alle eksisterende `Person` objekter.

Bemærk 1: Mængden af eksisterende `Person` objekter er en delmængde af de oprettede `Person` objekter.

Bemærk 2: Denne opgave er et sted mellem svær og umulig, i hvert fald hvis man ikke ænsker at kortslutte Java's garbage collector.

Bemærk 3: I er *ikke* blevet introduceret til de værktøjer der skal til for at løse opgaven.

17.2 Den Transsylvanske Fårehyrdeprøve

Langt ude i det ydre Transsylvanien, indhyldet i en stadig tåge, lever et fattigt men stolt folkefærd. Gennem utallige generationer har de opbygget en traditionsrig kultur omkring fåret. Fåret giver dem uld til tøj, og både mælk og kød til spise. De kan ikke eksistere uden fåret. Og derfor er dét at blive tildelt en fårehyrdes ansvar det ypperste man kan opnå. Men før man får overrakt ansvaret for folkets fremtid, skal man vise sit værd; man skal bestå den Transsylvanske Fåreprøve.

Denne prøve går ud på at man har to græsmarker der er forbunder af en smal sti. På den ene side af stien er en bar lodret klippe og på den anden side en dyb afgrund. 5 får er på vej fra den ene græsmark til den anden. De går helt

tæt: Hovede til hale. 5 andre får er på vej i den modsatte retning, og de går ligeledes helt tæt. De to flokke står nu med præcist ét fårs afstand til hinanden og er forvirrede over hele situationen. Stien er ikke bred nok til at to får kan krydse hinanden og hvis et får forsøger at vende om eller gå baglæns bliver det forvirret og falder i afgrunden. Fårene er dog nærværende nok til at reagere på kommandoer og stoler betingelsesløst på fårehyrden. For at bestå prøven skal du få de to flokke til at passere hinanden ved at at give korrekte instruktioner til de enkelte får. Du kan enten bede et får om at gå (et fårs længde) frem, eller at hoppe fremad. Når et får hopper fremad kan det lige akkurat hoppe over et andet får.

Bemærk: Det er ikke alle og enhver der kan blive fårehyrde i Transsylvanien, så det er nok urealistisk at du kommer hele vejen igennem opgaven.

Opgaven tager udgangspunkt i følgende klasser. Det er din upgave at udfylde `solve` metoden i `Solver` klassen, sådan at det fulde program løser fårehyrdeprøven. Instansvariablen `path` repræsenterer stien. Her kan man igennem et metodekald til `peek` observere hvilke får der er på de forskellige diskrete positioner. Sådan et får kan man bede om at gå fremad (ved kald til `walk` metoden) eller at hoppe fremad (ved kald til `jump` metoden). Efter hver af disse operationer vil systemet sørge for at printe stiens aktuelle tilstand ud på skærmen. Din opgave er at skrive en algoritmen der ved hjælp af disse operationer får fjernet alle får fra stien. Når et får krydser én af stiens to ender siger vi at den forlader stien. Hovedprogrammet er i `Test` klassen. En god løsning vil også fungere ved andre størrelser af `PROBLEM_SIZE` i `Test`.

```
class Sheep {
    public static final int SHEEP_SIZE = 3;

    Path    path;
    boolean rightbound;
    int position = -1;

    public Sheep (Path path, boolean rightbound) {
        this.path       = path;
        this.rightbound = rightbound;
    }

    public void print () {
        System.out.print( (rightbound ? "-m*" : "*m-") );
    }

    public void setPosition (int i) {
        position = i;
        path.setPosition(this, i);
    }

    public void walk () throws DeadSheepException {
        path.movePosition(position, rightbound ? ++position : --position);
    }
```

```
    public void jump () throws DeadSheepException {
        int old_pos = position;
        position += rightbound ? 2 : -2;
        path.movePosition(old_pos, position);
    }

    public boolean isRightbound () {
        return rightbound;
    }
}

class Path {
    Display display;
    Sheep[] spots;
    int problemsize;
    int spotcount;

    public Path (Display display, int problemsize) {
        this.problemsize = problemsize;
        this.spotcount   = 2*problemsize+1;

        spots = new Sheep[spotcount];

        this.display = display;
        display.setPath(this);
    }

    public void print () {
        for (int i=0 ; i<spots.length ; i++) {
            System.out.print(" ");
            if (spots[i]==null) {
                for (int j=0 ; j<Sheep.SHEEP_SIZE ; j++) {
                    System.out.print(" ");
                }
            } else {
                spots[i].print();
            }
        }
        System.out.println("");

        for (int i=0 ; i<spots.length ; i++) {
            System.out.print("+");
            for (int j=0 ; j<Sheep.SHEEP_SIZE ; j++) {
                System.out.print("-");
            }
        }
        System.out.println("+");
        System.out.println("");
    }
```

```
    public void setPosition (Sheep sheep, int i) {
        if (i<0 || i>=spotcount)
            throw new RuntimeException("Invalid position: "+i);
        spots[i] = sheep;
        display.update();
    }

    public void movePosition (int src, int dst) throws DeadSheepException {
        if (src<0 || src>=spotcount)
            throw new RuntimeException("Invalid source position: "+src);
        if (spots[src]==null)
            throw new RuntimeException("Trying to move null sheep");

        if (dst>=0 && dst<spotcount) {
            if (spots[dst]!=null) throw new DeadSheepException();
            spots[dst] = spots[src];
        }

        spots[src] = null;
        display.update();
    }

    public Sheep peek (int i) {
        if (i<0 || i>=spotcount) return null;
        return spots[i];
    }

    public int getSpotcount () {
        return spotcount;
    }

    public boolean isSolved () {
        for (int i=0 ; i<problemsize ; i++) {
            if (spots[i]!=null) return false;
        }

        return true;
    }
}

class DeadSheepException extends Exception {}

class Display {
    boolean running = false;
    Path path;

    public Display () {
    }

    public void setPath (Path path) {
```

```
        this.path = path;
    }

    public void start () {
        running = true;
    }

    public void update () {
        if (running) {
            path.print();
        }
    }
}

class Test {
    private static final int PROBLEM_SIZE = 5;

    public static void main (String args[]) {
        Display display = new Display();
        Path    path    = new Path(display, PROBLEM_SIZE);
        Solver  solver  = new Solver(path);

        // init puzzle
        for (int i=0 ; i<PROBLEM_SIZE ; i++) {
            Sheep s1 = new Sheep(path, true);
            s1.setPosition(i);
            Sheep s2 = new Sheep(path, false);
            s2.setPosition(PROBLEM_SIZE+1+i);
        }

        // start displaying state
        display.start();
        display.update();

        // perform test
        try {
            solver.solve();

            if (path.isSolved()) {
                System.out.println("You have graduated!");
            } else {
                System.out.println("You have given up!");
            }
        } catch (Exception e) {
            System.out.println("You have failed!");
        }
    }
}

class Solver {
```

```
    Path path;

    public Solver (Path path) {
        this.path = path;
    }

    public void solve () throws DeadSheepException {
    }
}
```

17.3 Hangman

Lav en implementering af Hangman (da: Galgespil)[9].

Bemærk: Dette er en svær opgave hvis løsningen skal være pæn.

17.4 Den Omrejsende Salgsmand

I denne opgave skal vi hjælpe Benny fra Esbjerg. Benny er en salgsperson, og han skal besøge de 15 største byer i Danmark for at sælge nogle meget eftertragtede gaming laptops. For Benny er tid penge, og hans mål er at maksimere profitten. Han ønsker derfor at finde den sekvens af byer der starter (og slutter) i Esbjerg, og som repræsenterer den korteste totale rejse. Selvom Benny er en dygtig salgsperson, er han ikke den skarpeste med et landkort. Det han har brug for er en softwareingeniør! Kan du hjælpe ham ved at lave et program der udregner den korteste rute der besøger disse byer?

Som en hjælp udleveres følgende datastrukturer:

```
String[] citynames = new String[] {
    "Esbjerg",
    "Helsingør",
    "Herning",
    "Horsens",
    "Kolding",
    "København",
    "Næstved",
    "Odense",
    "Randers",
    "Roskilde",
    "Silkeborg",
    "Vejle",
    "Viborg",
    "Aalborg",
    "Århus",
};

// source: https://dk.afstand.org
```

[9] https://en.wikipedia.org/wiki/Hangman_(game)

```
int[][] distances = {
    {  0, 269,  82,  98,  65, 260, 211, 123, 149, 230, 105,  74, 126, 198, 134},
    {269,   0, 226, 173, 206,  40, 105, 157, 166,  55, 191, 196, 207, 200, 150},
    { 82, 226,   0,  63,  79, 230, 202, 121,  76, 202,  36,  59,  45, 117,  77},
    { 98, 173,  63,   0,  48, 172, 139,  62,  68, 142,  40,  26,  75, 132,  40},
    { 65, 206,  79,  48,   0, 196, 148,  59, 114, 165,  77,  25, 110, 176,  88},
    {260,  40, 230, 172, 196,   0,  71, 141, 180,  31, 196, 190, 218, 224, 156},
    {211, 105, 202, 139, 148,  71,   0,  89, 174,  50, 175, 150, 204, 232, 142},
    {123, 157, 121,  62,  59, 141,  89,   0, 121, 110, 102,  64, 136, 186,  86},
    {149, 166,  76,  68, 114, 180, 174, 121,   0, 156,  44,  90,  42,  65,  36},
    {230,  55, 202, 142, 165,  31,  50, 110, 156,   0, 169, 160, 193, 206, 130},
    {105, 191,  36,  40,  77, 196, 175, 102,  44, 169,   0,  53,  35,  99,  41},
    { 74, 196,  59,  26,  25, 190, 150,  64,  90, 160,  53,   0,  86, 151,  65},
    {126, 207,  45,  75, 110, 218, 204, 136,  42, 193,  35,  86,   0,  72,  63},
    {198, 200, 117, 132, 176, 224, 232, 186,  65, 206,  99, 151,  72,   0, 101},
    {134, 150,  77,  40,  88, 156, 142,  86,  36, 130,  41,  65,  63, 101,   0},
};
```

Her indeholder `citynames` et array af bynavne. Disse byers indekser kan bruges i `distances` til at slå distances mellem to by er op. Eksempelvis har Helsingør indekset 1 og Kolding indekset 4. Derfor indeholder `distances[1][4]` distancen mellem Helsingør og Kolding (i fugleflugt).

Dette problem er for øvrigt kendt som "Traveling Salesman Problemet"[10].

Bemærk: Dette er en svær opgave, og det tager tid for programmet at finde løsningen.

18 Exceptions

18.1 Lagersystem

Disse opgaver bygger videre på et løsningsforslag til opgaven fra afsnit 16.1. I vælger selv om I vil løse opgaverne på jeres egen løsning eller anvende referenceløsningen fra afsnit 35.1.

Delopgave 1

I skal implementere en klasse ved navn `ExpiredItemAddedException`, der arver fra `Exception`.

Denne skal i sin constructor overføre strengen *"Attempted to add expired product to database"* til constructuren i `Exception` ved hjælp af `super` keyworded.

Delopgave 2

Constructoren i `FoodItem` skal kaste den exception I lavede i ovenstående delopgave ved hjælp af `throws` keywordet, når der bliver forsøgt at oprette et produkt med en dato, der ligger i fortiden.

Delopgave 3

[10] `https://en.wikipedia.org/wiki/Travelling_salesman_problem`

I skal udføre exception-handling (da: håndtering af denne exception) i klassen med jeres `main` metode, således at programmet fanger `ExpiredProductAddedException` når den bliver kastet og håndterer denne exception hensigtsmæssigt.

Delopgave 4

I stedet for at få constructoren i `FoodItem` til at kaste et `Exception` objekt, kunne man have fået `Inventory` til at kaste denne exception, når man forsøger at tilføje et udløbet `FoodItem` til et `Inventory` objekt.

Overvej hvilke egenskaberne dette designvalg medfører i forhold til dét vi valgte i delopgave 2. Hvilken løsning er mest hensigtsmæssig, og hvorfor?

19 Collections

19.1 CPR Register

Delopgave 1

Implementer klassen fra nedenstående UML diagram i Java.

Person
- name: String - courses: int - cpr: String
+ getName(): String + getAge(): int + getCpr(): String + toString(): String

Hint: Når `toString` metoden er angivet på klassen, så betyder det, at I skal overskrive metoden sådan at den får en meningsfuld implementering i jeres program.

Delopgave 2

Lav en `ArrayList` indeholdende 5 `Person` objekter, som du selv bestemmer attribut-værdier for. Den ene person skal have CPR nummeret `010101-0101`.

Delopgave 3

Iterer igennem (gennemløb) jeres `ArrayList` objekt og find personen med CPR-nummeret `010101-0101` ved at kalde metoden `getCpr` på hvert af objekterne.

Udskriv resultatet af et (eksplicit eller implicit) kald til `toString` metoden på dette objekt.

Delopgave 4

Opret et `HashMap<String,Person>` hashmap og tilføj de samme `Person` objekter til dette map, som du tidligere tilføjede til `ArrayList` objektet i delopgave 2. Brug CPR nummer som *key*.

Delopgave 5

Find personen med CPR-nummeret `010101-0101` i `HashMap` objektet og udskriv værdien returneret fra et kald til `toString` metoden på dette objekt.

19.2 Lagersystem

Disse opgaver bygger videre på et løsningsforslag til opgaven fra afsnit 18.1. I vælger selv om I vil løse opgaverne på jeres egen løsning eller anvende referenceløsningen fra afsnit 37.1.

Delopgave 1

I skal implementere interfacet `Comparable<Item>` i klassen `Item`. Dette kræver, at I implementerer metoden `compareTo(Item it)` i klassen `Item`. Gør dette på en måde, således at metoden returnerer en negativ værdi hvis objektet der sammenlignes på (dét som metoden kaldes igennem) har en pris der er mindre end `it.getPrice()`, positiv hvis den har en pris der er større end `it.getPrice()` og 0 hvis den har en pris der er lig med den værdi som `it.getPrice()` returnerer.

Delopgave 2

Lav en klasse ved navn `ItemNameLengthComparator`, der implementerer interfacet `Comparator<Item>`. I `Compare(Item i1, Item i2)` metoden fra dette interface skal længden på `name` attributen i objekterne `i1` og `i2` sammenlignes, således at der returneres en positiv `int` værdi hvis længden af strengen fra `name` attributten på `i1` er større end længden på `name` attributten i `i2`, en negativ `int` værdi hvis det er omvendt og en værdi på 0 hvis de er lige lange.

Delopgave 3

Implementer en metode i `Inventory` ved navn `sortedByNameLength`, der returnerer en liste af produkter sorteret efter længden af deres navne. Her skal I anvende den *comparator* som I implementerede i delopgave 2, sammen med `Collections.sort(List, Comparator)`.

Delopgave 4

Implementér en metode i `Inventory` ved navn `sortedByPrice`, der returnerer en liste af produkter, sorteret efter deres pris. Her skal I anvende metoden `Collections.sort(List)`.

Delopgave 5

Validér jeres sorteringer ved at lave en test i jeres `main` metode der skriver resultatet af udvalgte kald til metoderne `sortedByNameLength` og `sortedByPrice` ud.

Part II

Vejledende Løsninger

20 Typer

20.1 Hello, World

```
public class Hello
{
    public static void main (String[] args) {
        System.out.println("Hello, World");
    }
}
```

20.2 Måned

Der er 12 forskellige måneder, hvor man kunne sige at 1 repræsenterer Januar, 2 Februar ... Det er ikke klart hvad 1.1 bør betyde, så det giver mening at begrænse sig til heltal. Derfor vil vi naturligt bruge en heltallig type, og da vi vi kun har 12 måneder er `int` mere pladsøkonomisk end `long`.

20.3 Heltallige Grænser

I Java er der 4 (primitive) heltalstyper: `byte`, `short`, `int` og `long`.

En `byte` kan repræsenterer alle heltal fra og md -128 til og med 127. For at finde ud af hvad der sker når vi overskrider det største heltal der kan representeres, skriver vi følgende program:

```
public class IntegerBoundary
{
    public static void main (String[] args) {
        byte b = 127;
        System.out.println(b);
        b += 1;
        System.out.println(b);
    }
}
```

Kører vi dette program får vi:

```
127
-128
```

Vi har altså, at $127 + 1$ giver os -128. Eller med andre ord; når vi lægger én til det størst mulige tal så får vi det mindst mulige tal. Dette er ikke et tilfælde, og gør sig generelt gældende for alle de 4 ovenstående typer.

21 Expressions

21.1 Tildeling

Du kan jo prøve:

```
public class ExprTest
{
    public static void main (String[] args) {
        int var1;
        int var2 = var1 = 42;
        System.out.println(var1);
        System.out.println(var2);
    }
}
```

21.2 Expression vs Statement

Expressions og statements er forskellige syntaktiske konstruktioner. Det betyder at Javas oversætter (nydansk: compiler) forventer expressions nogen steder og statements andre steder. Overholder ens program ikke disse *syntaktiske* regler vil oversætteren give en såkaldt syntaksfejl og opgive.

Den vigtigeste forskel er at et expression – i modsætning til statements – kan evalueres til en værdi. Hovedstrukturen af et stykke logik er udgøres derfor at statements og expressions bruges til at udregne mellemresultater.

22 Variable

22.1 Areal af Cirkler

```
public class CircleArea
{
    public static void main (String[] args) {
        final double pi = 3.14;
        double r;

        r = 1;
        System.out.println("r="+r+" -> area="+(pi*r*r));

        r = 3;
        System.out.println("r="+r+" -> area="+(pi*r*r));
```

```
        r = 5;
        System.out.println("r="+r+" -> area="+(pi*r*r));
    }
}
```

22.2 Inkrementering af Måned

Observation: Da vi er i den niende måned gav jeg 9 som initiel værdi. Denne værdi ændrede sig ikke da jeg lagde en halv til, og heller ikke da jeg gjorde dette igen.

Forklaring: Ved addition af forskellige typer foretager Java en implicit konvertering til den første type. Denne type er heltallige og værdien 0.5 bliver derfor rundet ned til 0. Derfor lægges der i praksis nul til to gange.

22.3 Værdi vs Variabel

En variabel er en symbolsk reference (eller indirektion) til en værdi. Vi siger at variablen *peger* på værdien. Vi kan ændre på hvilken værdi en variabel peger på ved at tildele den en ny værdi. I praksis ligger værdien et sted i hukommelsen og variablen indeholder adressen på dette sted.

Dette er en *abstraktion* der gør det muligt at skrive kode der er uafhængig af specifikke værdier.

22.4 Daglige Differencer

```
public class DailyDiffs
{
    public static void main (String[] args) {
        final double day1 = 21.5;
        final double day2 = 23.7;
        final double day3 = 19.6;
        final double day4 = 22.5;
        final double day5 = 25.3;
        final double day6 = 21.7;
        final double day7 = 18.9;

        System.out.println(day2-day1);
        System.out.println(day3-day2);
        System.out.println(day4-day3);
        System.out.println(day5-day4);
        System.out.println(day6-day5);
        System.out.println(day7-day6);
    }
}
```

22.5 Gennemsnitlig Alder

Når programmet udføres printes følgende ud:

```
Average livespan of a male computer scientist: 76.666664
Average livespan of a female computer scientist: 68.666664
Average livespan of a computer scientist: 72.666664
Males lives this much longer than female: 8.0
```

Programmet har en main metode der består af (i) 6 variabel erklæringer med initielle værdier, (ii) en række variabel erklæringer der initialiseres til at holde resultatet af nogle udregninger, og til sidst udskrives resultaterne.

I den første del erklæres 6 `int` variabler. Hver af disse er navngivet efter en kendt datalog og initialiseres til hvor gamle de blev. For hver af disse er der et udkommenteret link til deres side på Wikipedia. Dernæst kommer der fire udregninger. Hver udregning gemmes i en `float` variabel.

I den første udregning lægges samtlige mandlige variable sammen. Resultatet af denne opsummering castes til en `float` hvorefter det divideres med 3 (antallet) og gemmes i `male_avg` variablen. Tilsvarende udregnes og gemmes gennemsnittet for kvinder i `female_avg`. I den tredje udregning lægges disse to gennemsnit sammen og resulatet af dette divideres med 2 og gemmes i variablen `avg`. Dette er korrekt eftersom der er lige mange mænd og kvinder, og da `male_avg` er af typen `float` er det ikke nødvendigt at caste for at undgår afrundingsfejl i divisionen. I den sidste udregning trækkes `female_avg` fra `male_avg` og resultatet gemmes i variablen `diff`.

Til sidst printes hver af disse 4 udregnede variable ud på skærmen. For hver af dem bruges to statements. I det første statement udskrives en streng der beskriver hvad værdien betyder uden linjebrud. I det andet og sidste statement udskrives værdien med linjebrud.

22.6 Printf

Den fulde test kode:

```
public class Formatting
{
    public static void main (String[] args) {
        int i = 42;
        long l = 56;
        float f = 3.14159F;
        double d = 3.14159*10;
        System.out.printf("i=%d l=%,4d f=%f d=%6.2f", i, l, f, d);
        System.out.println("");
    }
}
```

Vi kan bruge `System.out.printf` til at *padde* værdier med mellemrum og

bestemme hvor mange decimaler vi ønsker efter kommaet. Herved kan vi formatere udskrifter.

23 Booleans

23.1 Definition

En `boolean` er både en type og en variabel af denne type. Det betyder at variablen have have (pege på) én af to værdier; true (sandt) og false (falsk). Dette er en enkelt bit information, men tildeles typisk mere plads i hukommelsen. En række særlige (boolske) operationer er defineret på denne type. Disse kan bruges til at implementere Boolsk logik.

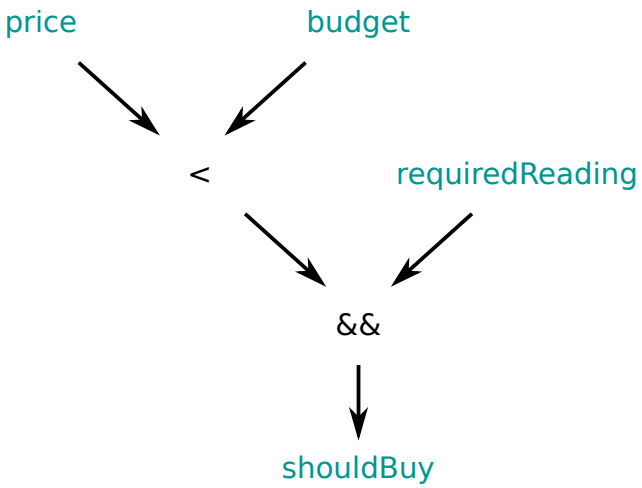
23.2 Oprettelse

Boolske værdier bliver produceret når:

1. Man i koden skriver `true` eller `false`.
2. Som resultatet af en sammenligning.
3. Som resultatet af en boolsk operation.

23.3 Købsbeslutning

Udregningen foregår således:



Variablen `shouldBuy` repræsenterer en beslutning for hvorvidt en bog skal købes. Dette sker når prisen er indenfor budgettet og bogen er en del af læsepensum.

24 Branches

24.1 Juleudsalg

```
public class ChristmasSale
{
    public static void main (String[] args) {
        final int secsPerDay  = 24*60*60;
        final int secsPerYear = 360*secsPerDay;
        final int xmas = (11*30+23)*24*60*60;

//        long epoch = xmas-secsPerDay;
//        long epoch = xmas+secsPerDay;
        long epoch = xmas;

        long years = epoch/secsPerYear;
        long days  = (epoch-years*secsPerYear)/secsPerDay;
        long month = days/30;
        long day   = days%30;

        double price = 599.95 * (month==11 && day==23 ? 0.7 : 1.0);
        System.out.println(price);
    }
}
```

24.2 Ferie

En typisk implementering:

```
public class Holiday
{
    public static void main (String[] args) {
        final int JANURAY  =  0;
        final int FEBRUARY =  1;
        final int March    =  2;
        final int APRIL    =  3;
        final int MAY      =  4;
        final int JUNE     =  5;
        final int JULY     =  6;
        final int AUGUST   =  7;
        final int SEPTEMER =  8;
        final int OCTOBER  =  9;
        final int NOVEMBER = 10;
        final int DECEMBER = 11;

        // choose a test case
        long month = OCTOBER;
//        long month = NOVEMBER;
```

```
        switch ((int)month) {
        case OCTOBER:
            System.out.println("Efterårsferie");
            break;
        case DECEMBER:
            System.out.println("Juleferie");
            break;
        case APRIL:
            System.out.println("Påskeferie");
            break;
        case JULY:
        case AUGUST:
            System.out.println("Sommerferie");
            break;
        default:
            System.out.println("Hårdt arbejde");
            break;
        }
    }
}
```

Karl Koders version:

```
public class KarlsHoliday
{
    public static void main (String[] args) {
        final int JANURAY  =  0;
        final int FEBRUARY =  1;
        final int March    =  2;
        final int APRIL    =  3;
        final int MAY      =  4;
        final int JUNE     =  5;
        final int JULY     =  6;
        final int AUGUST   =  7;
        final int SEPTEMER =  8;
        final int OCTOBER  =  9;
        final int NOVEMBER = 10;
        final int DECEMBER = 11;

        // choose a test case
        long month = OCTOBER;
//        long month = NOVEMBER;

        String message = (month==OCTOBER               ? "Efterårsferie"
                        : (month==DECEMBER             ? "Juleferie"
                        : (month==APRIL                ? "Påskeferie"
                        : (month==JULY || month==AUGUST ? "Sommerferie"
                        : "Hårdt arbejde"))));
        System.out.println(message);
```

```
    }
}
```

25 Loops

25.1 Areal af Cirkler

```
public class CircleArea
{
    public static void main (String[] args) {
        final double pi = 3.14;

        for (double r=1 ; r<=5 ; r+=2) {
            System.out.print("r=");
            System.out.print(r);
            System.out.print(" -> area=");
            System.out.println(pi*r*r);
        }
    }
}
```

25.2 Længden af en Måned

```
public class Months
{
    public static void main (String[] args) {
        final int month = 6;
        int length = -1;

        if (month==1 || month==3 || month==5 || month==7 || month==8 ||
            month==10 || month==12) {
            length = 31;
        } else if (month==2) {
            length = 28;
        } else if (month==4 || month==6 || month==9 || month==11) {
            length = 30;
        }

        if (length==-1) {
            System.out.println("Fejl: Måneden \""+month+"\"er ikke indenfor [1,12]");
        } else {
            System.out.println(length);
        }
    }
}
```

Karl Koder anbefaler dog:

```
public class KarlsMonths
{
    public static void main (String[] args) {
        final int month = 6;
        int length = -1;

        length = (month==1 || month==3 || month==5 || month==7 || month==8 ||
                  month==10 || month==12 ? 31 : 0)
               + (month==2 ? 28 : 0)
               + (month==4 || month==6 || month==9 || month==11 ? 30 : 0);

        if (length==-1) {
            System.out.println("Fejl: Måneden \""+month+"\"er ikke indenfor [1,12]");
        } else {
            System.out.println(length);
        }
    }
}
```

25.3 Primtal

```
public class Primes
{
    public static void main (String[] args) {
        final int last = 1000000;
        int max = -1;

        for (int i=0 ; i<last ; i++) {
            boolean isPrime = true;

            // even numbers over 2 are not prime
            if (i>2 && (i&1)==0) isPrime = false;

            // primes have a remainder when divided by a lower odd number
            for (int j=3 ; j<i ; j++) {
                if (i%j==0)  {
                    isPrime = false;
                    break;
                }
            }

            if (isPrime) {
//                System.out.println(i);
                max = i;
             }
        }

        System.out.println("Highest prime is "+max);
    }
```

```
}
```

26 Arrays

26.1 Definition

Et array er en *datastruktur* hvori et fast antal elementer er placeret fortløbende i hukommelsen. Eftersom alle elementer har samme størrelse kan et opslag foretages effektivt. Bag scenen gør Java noget i stil med:

$$A_{\text{element}} = A_{\text{array}} + \text{index} * \textbf{sizeof}(\textbf{typeof}(\text{element})) \tag{1}$$

26.2 Anvendelse

Arrays løser den problematik hvor alle heltal under et bestemt tal skal associeres (på nydansk siger vi *mappes*) til en individuel værdi.

En hyppig variant af dette scenarie er at alle heltal *mellem* to tal skal associeres til en bestemt værdi. Her vælger man ofte at forskyde samtlige elementer ved at trække det mindste af tallene fra alle indekseringer. Så slipper man for at spilde plads i hukommelsen.

26.3 Type af Indhold vs Type af Array

Alle elementer i et array skal have samme type. Et arrays type er afledt af denne type. Ønsker man eksempelvis elementer af typen `int` så skal ens array have typen `int[]`.

Note: I en senere forelæsning vil I se hvordan man – i nogen grad – kan omgå den første af disse regler ved brug af objektorienterede konstruktioner.

26.4 Størst i Array

Hvis vi har en garanti for at der kun eksisterer en enkel instans af den største værdi så vil følgende virke:

```
public class Index
{
    public static void main (String[] args) {
        int[] a = {1, 7, 4, 1, 23, 54, 8, 34, 54, 12, 11, 8, 25};

        int max  = a[0];
        int maxi = 0;

        for (int i=0 ; i<a.length ; i++) {
```

```
            int value = a[i];
            if (value>max) {
                max  = value;
                maxi = i;
            }
        }

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

Ellers er vi nødt til at gøre noget i stil med:

```
public class IndexAll
{
    public static void main (String[] args) {
        int[] a = {1, 7, 4, 1, 23, 54, 8, 34, 54, 12, 11, 8, 25};

        // find max
        int max  = a[0];
        for (int value: a) {
            if (value>max) {
                max  = value;
            }
        }

        // locate occurences and output them
        for (int i=0 ; i<a.length ; i++) {
            int value = a[i];
            if (value==max) {
                System.out.println(i);
            }
        }
    }
}
```

26.5 Deklaration af Størrelse

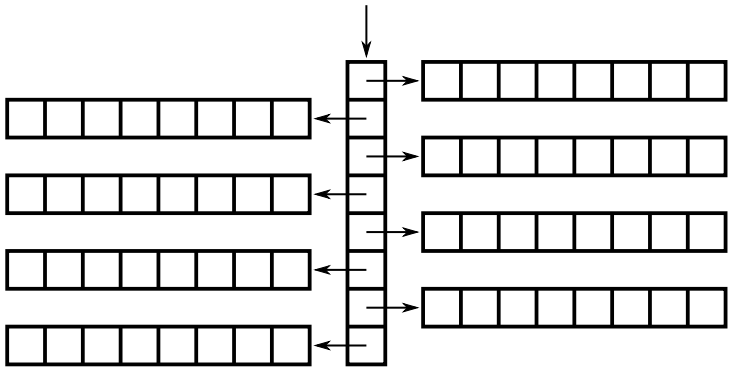
Når vi opretter et array uden eksplicit at definere dets indhold skal vi fortælle hvor stort dette array skal være (ellers ved Java jo ikke hvor meget hukommelse der skal allokeres). I Java gøres dette ved hjælp af et expression der skal evaluere til et heltal. Det betyder at størrelsen fx kan være en konstant værdi, indholdet af en variabel eller resultatet af en udregning.

26.6 Sudoku Plade

En sudoku plade er en to-dimensionel størrelse med $9 \cdot 9$ felter. Dette passer perfekt til et 2D array af heltal. I Java har vi teknisk set ikke 2D arrays. I stedet

har Java arrays af arrays, og det er i praksis næsten lige så godt. Vi snakker her om 9 indre arrays (af `int`) der er indlejret i en ydre array (af `int[]`). Vi siger at de indre arrays er *nested* i det ydre.

Strukturelt er en sudoku et nested array lagt således ud:



Eftersom hukommelse har endimensionelle adresser i en computer vil de enkelte arrays blive lagt mere eller mindre i serie. Man er dog nødt til at følge to referencer for at komme til et felts data, og sådanne referencer kan omdefineres.

Dette betyder også at `System.arraycopy` kun kan bruges til at kopiere det yderste array. Resultatet af en sådan operation vil være at man har to forskellige ydre arrays der hver peger på (delte) indre arrays.

26.7 Areal af Cirkler

```
public class CircleArea
{
    public static void main (String[] args) {
        final int[] radiuses = {1, 3, 5};
        final double pi = 3.14;

        for (int r: radiuses) {
            System.out.println("r="+r+" -> area="+(pi*r*r));
        }
    }
}
```

26.8 Primtal

```
public class Primes
{
    public static void main (String[] args) {
        final int last = 1000000;
        boolean[] sieve = new boolean[last];
```

```
        // initialize sieve
        for (int i=1 ; i<sieve.length ; i++) sieve[i] = true;

        // fill out sieve
        for (int i=2 ; i<java.lang.Math.sqrt(sieve.length) ; i++) {
            if (sieve[i]) {
                for (int j=i*2 ; j<last ; j+=i) {
                    sieve[j] = false;
                }
            }
        }

//        // verify
//        for (int i=1 ; i<sieve.length ; i++) {
//            if (sieve[i]) System.out.println(i);
//        }

        // find and print out largest prime
        int max = -1;
        for (int i=last-1 ; i>0 ; i--) {
            if (sieve[i]) {
                max = i;
                break;
            }
        }
        System.out.println("Highest prime is "+max);
    }
}
```

26.9 Kalender Prettyprinting

```
public class Calendar
{
    public static void main (String[] args) {
        final String[] days = {
            "Mandag",
            "Tirsdag",
            "Onsdag",
            "Torsdag",
            "Fredag",
            "Lørdag",
            "Søndag",
        };
        final String[] months = {
            "Januar",
            "Februar",
            "Marts",
            "April",
            "Maj",
```

```
            "Juni",
            "Juli",
            "August",
            "September",
            "Oktober",
            "November",
            "December",
        };
        final int[] monthsNormal = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
        final int[] monthsLeap =   { 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };

        final int year = 2019;
        int firstDay = 1; // index for days

        // find the correct month lenghts
        boolean leap = year%4==0 && year%400!=0;
        int[] monthLengths = leap ? monthsLeap : monthsNormal;

        // build base data structure
        String[][] calendar = new String[12][];
        int dayNumber = firstDay;
        for (int month=0 ; month<calendar.length ; month++) {
            int monthLength = monthLengths[month];
            calendar[month] = new String[monthLength];
            for (int day=0 ; day<monthLength ; day++) {
                calendar[month][day] = days[dayNumber++ % days.length];
            }
        }

        for (int month=0 ; month<calendar.length ; month++) {
            for (int day=0 ; day<calendar[month].length ; day++) {
                String monthName = months[month];
                String dayName   = calendar[month][day];

                System.out.printf("%d %2d. ", year, day+1);
                System.out.println(monthName+": "+dayName);
            }
        }
    }
}
```

26.10 Sudoku Checker

```
public class SudokuVerifier
{
    public static void main (String[] args) {
        // test cases
        final int[][] puzzle = {
          {7, 3, 6, 4, 5, 2, 9, 8, 1},
```

```
            {1, 9, 8, 6, 3, 7, 4, 5, 2},
            {4, 2, 5, 9, 8, 1, 3, 7, 6},
            {3, 6, 4, 5, 2, 8, 1, 9, 7},
            {9, 5, 2, 7, 1, 4, 6, 3, 8},
            {8, 1, 7, 3, 9, 6, 2, 4, 5},
            {2, 8, 9, 1, 7, 3, 5, 6, 4},
            {6, 7, 3, 2, 4, 5, 8, 1, 9},
            {5, 4, 1, 8, 6, 9, 7, 2, 3},
        };
//        final int[][] puzzle = {
//          {7, 3, 6, 4, 5, 2, 9, 8, 1},
//          {1, 9, 8, 6, 3, 7, 4, 5, 2},
//          {4, 2, 5, 9, 8, 4, 3, 7, 6},
//          {3, 6, 4, 5, 2, 8, 1, 9, 7},
//          {9, 5, 2, 7, 1, 4, 6, 3, 8},
//          {8, 1, 7, 3, 9, 6, 2, 4, 5},
//          {2, 8, 9, 1, 7, 3, 5, 6, 4},
//          {6, 7, 3, 2, 4, 5, 8, 1, 9},
//          {5, 4, 1, 8, 6, 9, 7, 2, 3},
//        };

        boolean correct = true;

        // check rows
        for (int y=0; y<9 ; y++) {
            for (int x=0; x<9 ; x++) {
                for (int x2=x+1 ; x2<9 ; x2++) {
                    if (puzzle[y][x]==puzzle[y][x2]) correct = false;
                }
            }
        }

        // check columns
        for (int x=0; x<9 ; x++) {
            for (int y=0; y<9 ; y++) {
                for (int y2=y+1 ; y2<9 ; y2++) {
                    if (puzzle[y][x]==puzzle[y2][x]) correct = false;
                }
            }
        }

        // check groups
        for (int x=0; x<3 ; x++) {
            for (int y=0; y<3 ; y++) {
                for (int i=0 ; i<9 ; i++) {
                    int ix = i%3;
                    int iy = i/3;
                    for (int i2=i+1 ; i2<9 ; i2++) {
                        int i2x = i2%3;
                        int i2y = i2/3;
```

```
                        if (puzzle[y*3+iy][x*3+ix]==puzzle[y*3+i2y][x*3+i2x]) {
                            correct = false;
                        }
                    }
                }
            }
        }

        System.out.println("Puzzle is "+(correct ? "correct" : "incorrect"));
    }
}
```

27 Metoder

27.1 Sum

```
public class Adder
{
    public static int add (int a, int b) {
        return a+b;
    }

    public static void main (String[] args) {
        for (int a=0 ; a<=10 ; a++) {
            for (int b=0 ; b<=10 ; b++) {
                System.out.printf("%3d", add(a,b));
            }
            System.out.println("");
        }
    }
}
```

27.2 Egen Kvadratrod

```
public class Sqrt
{
    public static double sqrt (double x) {
        double candidate = 0.0;
        for (double pointer = 1000000000.0; pointer>0.000000001 ; pointer /=10) {
            candidate += 10*pointer;
            for (int i=0 ; i<10; i++) {
                candidate -= pointer;
                if (candidate*candidate<x) {
                    break;
                }
            }
        }
```

```
        return candidate;
    }

    public static void main (String[] args) {
        for (double x=1 ; x<=10 ; x++) {
            System.out.println("sqrt("+x+")="+sqrt(x));
        }
    }
}
```

27.3 Matematisk Ækvivalent

En matematisk funktion matcher er metode utroligt godt. Den store forskel er at metoder kan have side-effekter. Det betyder at de har adgang til tilstand (hukommelse) Igennem variable kan metoder kan læse fra og skrive til hukommelsen. Herved kan man gøre en metodes opførsel afhængig af hukommelsens tilstand.

27.4 Fakultet

```
public class Factorial
{
    public static int fac (int i) {
        if (i<0) {
            System.out.println("That was not very nice of you!");
        }

        if (i>0) {
            return i * fac(i-1);
        } else {
            return 1;
        }
    }

    public static void main (String[] args) {
        for (int i=1 ; i<10 ; i++) {
            System.out.println(" fac("+i+") = "+fac(i));
        }
    }
}
```

28 Grundlæggende Programmering

28.1 Sudoku Løser

En referenceløsning vil blive gjort tilgængelig uafhængig af dette dokument.

28.2 Game of Life

En referenceløsning vil blive gjort tilgængelig uafhængig af dette dokument.

28.3 8 Dronninge Problemet

Følgende implementation udprinter der første løsning:

```
public class Queens
{
    public static void print (boolean[][] board) {
        String line = "+-+-+-+-+-+-+-+-+";
        System.out.println(line);
        for (boolean[] row: board) {
            System.out.print("|");
            for (boolean cell: row) {
              System.out.print((cell?"Q":" ")+"|");
            }
            System.out.println("");
            System.out.println(line);
        }
    }

    public static boolean check (boolean[][] board) {
        boolean failure = false;

        for (int y=0 ; y<8 ; y++) {
            for (int x=0 ; x<8 ; x++) {
                boolean cell = board[y][x];
                if (!cell) continue;

                // row
                for (int i=0 ; i<8 ; i++) {
                    if (i==y) continue;
                    failure |= board[i][x];
                }

                // column
                for (int i=0 ; i<8 ; i++) {
                    if (i==x) continue;
                    failure |= board[y][i];
```

```
            }

            // diagonal: falling
            for (int i=0 ; i<8 ; i++) {
                int xderived = i;
                int yderived = y-x+i;
                if (yderived<0 ||
                    xderived<0 ||
                    yderived>7 ||
                    xderived>7 ||
                    (yderived==y && xderived==x)) continue;
                failure |= board[yderived][xderived];
            }

            // diagonal: rising
            for (int i=0 ; i<8 ; i++) {
                int xderived = i;
                int yderived = y+x-i;
                if (yderived<0 ||
                    xderived<0 ||
                    yderived>7 ||
                    xderived>7 ||
                    (yderived==y && xderived==x)) continue;
                failure |= board[yderived][xderived];
            }
        }
    }

    return !failure;
}

public static void fill (boolean[][] board) {
    fill(board, 0);
}

private static boolean fill (boolean[][] board, int y) {
    if (y==8) {
        return check(board);
    } else {
        for (int x=0 ; x<8 ; x++) {
            board[y][x] = true;
            boolean success = fill(board, y+1);
            if (success) {
              return true;
            }
            board[y][x] = false;
        }
        return false;
    }
}
```

```
    public static void main (String[] args) {
        boolean[][] board = new boolean[8][8];

        fill(board);
        print(board);
    }
}
```

29 Objekter

29.1 Kunder

```
public class Customer
{
    String name;
    int    id;
    double balance;

    Customer (String name, int id, double balance) {
        this.name    = name;
        this.id      = id;
        this.balance = balance;
    }

    Customer (String name, int id) {
        this(name, id, 0);
    }

    void deposit (double amount) {
        balance += amount;
    }

    void withdraw (double amount) {
        balance -= (balance > amount ? amount : 0);
    }

    double getBalance () {
        return balance;
    }

    public static void main (String[] args) {
        Customer aCustomer;

        aCustomer = new Customer("Donald Knuth", 42, 0);
        System.out.println(aCustomer.getBalance());
        aCustomer.deposit(1000);
        System.out.println(aCustomer.getBalance());
```

```
        aCustomer.withdraw(500);
        System.out.println(aCustomer.getBalance());
    }
}
```

29.2 Kunde Database

```
class CustomerDatabase {
    Customer[] customers;

    public CustomerDatabase () {
        customers = new Customer[10];
    }

    boolean add (Customer customer) {
        for (int i=0 ; i<customers.length ; i++) {
            if (customers[i]==null) {
                customers[i] = customer;
                return true;
            }
        }
        return false;
    }

    void remove (int id) {
        for (int i=0 ; i<customers.length ; i++) {
            if (customers[i]!=null && customers[i].id==id) {
                customers[i] = null;
            }
        }
    }

    Customer[] get () {
        return customers.clone();
    }

    void print () {
        for (Customer c: customers) {
            if (c==null) continue;
            System.out.println(c.name);
        }
    }

    public static void main (String[] args) {
        CustomerDatabase db = new CustomerDatabase();

        System.out.println("Result:"+db.add(new Customer("Alan Turing", 1912)));
        System.out.println("Result:"+db.add(new Customer("Ada Lovelace", 1815)));
        System.out.println("Result:"+db.add(new Customer("Charles Babbage", 1791)));
```

```
        System.out.println("Result:"+db.add(new Customer("Donald Knuth", 1938)));
        System.out.println("Result:"+db.add(new Customer("Grace Hopper", 1906)));
        System.out.println("Result:"+db.add(new Customer("Edsger Dijkstra", 1930)));
        System.out.println("Result:"+db.add(new Customer("Tony Hoare", 1934)));
        System.out.println("Result:"+db.add(new Customer("Hedy Lamarr", 1914)));
        System.out.println("Result:"+db.add(new Customer("Michael Stonebraker", 1943)));
        System.out.println("Result:"+db.add(new Customer("Jim Gray", 1944)));
        System.out.println("Result:"+db.add(new Customer("Douglas Engelbart", 1925)));
        System.out.println("");

        db.print();
    }
}
```

Bemærk: `get` metoden returnere en kopi af datastrukturen for ikke at give den kaldende metode mulighed for at navngive de originale data igennem en reference.

29.3 Farver

```
class RGB {
    int r, g, b;

    public RGB (int r, int g, int b) {
        this.r = r;
        this.g = g;
        this.b = b;
    }

    public int getR () {
        return r;
    }

    public int getG () {
        return g;
    }

    public int getB () {
        return b;
    }

    // https://www.geeksforgeeks.org/program-change-rgb-color-model-hsv-color-model/
    public HSV asHSV () {
        double r = (double) this.r / 255;
        double g = (double) this.g / 255;
        double b = (double) this.b / 255;

        double cmin = min(r, g, b);
        double cmax = max(r, g, b);
```

```
        double diff = cmax - cmin;

        double h = -1;
        double s = -1;
        double v = -1;

        if        (cmax==cmin) {
            h = 0;
        } else if (cmax==r) {
            h = (60 * ((g - b) / diff) + 360) % 360;
        } else if (cmax==g) {
            h = (60 * ((b - r) / diff) + 120) % 360;
        } else if (cmax==b) {
            h = (60 * ((r - g) / diff) + 240) % 360;
        }
        h *= 255.0/360;

        if (cmax==0) {
            s = 0;
        } else {
            s = (diff/cmax) * 255;
        }

        v = cmax * 255;

        return new HSV((int)h, (int)s, (int)v);
    }

    public String toString () {
        return "RGB("+r+","+g+","+b+")";
    }

    private double min (double a, double b, double c) {
        return (a<b ? (a<c ? a : c) : (b<c ? b : c));
    }
    private double max (double a, double b, double c) {
        return (a>b ? (a>c ? a : c) : (b>c ? b : c));
    }
}

class HSV {
    int h, s, v;

    public HSV (int h, int s, int v) {
        this.h = h;
        this.s = s;
        this.v = v;
    }

    public int getH () {
```

```
    return h;
}

public int getS () {
    return s;
}

public int getV () {
    return v;
}

// http://dystopiancode.blogspot.com/2012/06/hsv-rgb-conversion-algorithms-in-c.html
public RGB asRGB () {
    double h = (double)(this.h*360.0/255); // scale to 360.0
    double s = (double)this.s/255.0;       // scale to 1.0
    double v = (double)this.v/255.0;       // scale to 1.0

    double c = v*s;
    double m = v-c;
    double x = c*(1.0-abs(((h/60) % 2)-1));

    double r, g, b;
    if        (h>=0 && h<60) {
        r = c+m;
        g = x+m;
        b = m;
    } else if (h>=60 && h<120) {
        r = x+m;
        g = c+m;
        b = m;
    } else if (h>=120 && h<180) {
        r = m;
        g = c+m;
        b = x+m;
    } else if (h>=180 && h<240) {
        r = m;
        g = x+m;
        b = c+m;
    } else if (h>=240 && h<300) {
        r = x+m;
        g = m;
        b = c+m;
    } else if (h>=300 && h<360) {
        r = c+m;
        g = m;
        b = x+m;
    } else {
        r = m;
        g = m;
        b = m;
```

```
        }

        return new RGB((int)(r*255), (int)(g*255), (int)(b*255));
    }

    public String toString () {
        return "HSV("+h+","+s+","+v+")";
    }

    private double abs (double input) {
        return (input<0 ? -input : input);
    }
}

class Test {
    public static void main(String[] args) {
        RGB[] tests = {
            new RGB(255,   0,   0),
            new RGB(  0, 255,   0),
            new RGB(  0,   0, 255),
            new RGB(  0,   0,   0),
            new RGB(255, 255, 255),
            new RGB( 18,  52,  86),
            new RGB( 88, 154, 188),
        };

        for (RGB test: tests) {
            HSV hsv = test.asHSV();
            RGB rgb = hsv.asRGB();

            System.out.println(test+" -> "+hsv+" -> "+rgb);
        }
    }
}
```

30 Arv

30.1 Lagersystem

```
class Item {
    private String name;
    private double price;

    Item (String name, double price) {
        this.name  = name;
        this.price = price;
    }
```

```
    public String getName () {
        return name;
    }

    public double getPrice () {
        return price;
    }
}

import java.util.Date;

class FoodItem extends Item {
    private Date expires;

    FoodItem (String name, double price, Date expires) {
        super(name, price);
        this.expires = expires;
    }

    public Date getExpires () {
        return expires;
    }

    @Override
    public String toString () {
        return "FoodItem name='"+getName()
             + "' price='"+getPrice()
             + "' expires='"+getExpires()+"'";
    }

    public static void main (String[] args) {
        FoodItem[] items = new FoodItem[10];

        for (int i=0 ; i<items.length ; i++) {
            items[i] = new FoodItem("Item "+i, 12.3*i,
                                    new Date(i*1000*60*60*24));
        }

        for (FoodItem item: items) {
            System.out.println(item);
        }
    }
}

class NonFoodItem extends Item {
    private String[] materials;

    NonFoodItem (String name, double price, String[] materials) {
        super(name, price);
        this.materials = materials;
```

```
    }

    public String[] getMaterials () {
        return materials;
    }

    @Override
    public String toString () {
        String m = "[";
        for (int i=0 ; i<materials.length ; i++) {
            m += (i==0 ? "" : ",")+materials[i];
        }
        m += "]";
        return "NonFoodItem name='"+getName()
             + "' price='"+getPrice()
             + "' materials='"+m+"'";
    }

    public static void main (String[] args) {
        NonFoodItem[] items = new NonFoodItem[10];

        for (int i=0 ; i<items.length ; i++) {
            items[i] = new NonFoodItem("Item "+i, 12.3*i,
                                       new String[] {"butter", "cream"});
        }

        for (NonFoodItem item: items) {
            System.out.println(item);
        }
    }
}
```

31 Navngivning

31.1 Killinger

```
class Kitten {
    double cuteness;
    static int count = 0;

    public Kitten (double cuteness) {
        this.cuteness = cuteness;
        if (cuteness>9000) {
            System.out.println("The cuteness of this kitten is over 9000!");
        }
        count++;
    }
```

```
    public static void main (String[] args) {
        for (int i=0 ; i<9002 ; i++) {
            new Kitten(i);
        }
        System.out.println("There are "+count+" kittens.");
    }
}
```

31.2 Operatorer

Der er to operatorer; `+` og `-`. De fem kald til `operator` metoden mapper således til disse operatorer:

$$\begin{aligned}(?1?) &\mapsto + \\ (?2?) &\mapsto + \\ (?3?) &\mapsto - \\ (?4?) &\mapsto - \\ (?5?) &\mapsto -\end{aligned}$$

Af disse er `+` implementeret af den første deklaration af `operator` metoden, og `-` implementeret af den sidste. Disse deklarationer adskilder sig naturligvis ved at have forskellige kroppe (eng: body), men de adskilder sig også ved deres signatur. Her er typen af tredje parameter forskellig. Parameterens værdi bliver ikke brugt til noget, men parameterens type gør de to signaturer unikke og derved bruges kan den enkelt implementation udpeges unikt.

Det gode ved denne implementation er at man sikrer sig at valget af implementation af `operator` metode udføres på oversættelsestidspunktet. En konsekvens af dette er at der ikke bruges tid på det på udførelsestidspunktet. Det dårlige ved denne implementation er at den er svær at læse. Mange vil nok ligefrem sige at det er misbrug af typesystemet. I praksis vil de fleste oversættere tilmed være dygtige nok til at oversætte følgende kode lige så effektivt:

```
class Operator {
    public static int operator (int a, int b, boolean plus) {
        if (plus) {
            return a+b;
        } else {
            return a-b;
        }
    }

    public static void main (String[] args) {
        for (int a=0 ; a<5 ; a++) {
            for (int b=0 ; b<5 ; b++) {
                System.out.println(a+" (?1?) "+b+" = "+operator(a, b, true));
                System.out.println(a+" (?2?) "+b+" = "+operator(a, b, true));
                System.out.println(a+" (?3?) "+b+" = "+operator(a, b, false));
                System.out.println(a+" (?4?) "+b+" = "+operator(a, b, false));
```

```
                System.out.println(a+" (?5?) "+b+" = "+operator(a, b, false));
            }
        }
    }
}
```

Derudover kan det diskuteres om det overhovedet er god stil at sende en operator med som parameter.

31.3 Scope

Følgende tabel viser hvor hver af de fem variable må deklareres og bør deklareres (markeret med understregning).

Lokation	i	d	tmp	sum	bonus
Location 0	×	×	×	×	×
Location 1	×	×	×	×	
Location 2		×			
Location 3					
Location 4					
Location 5	×	×	×	×	
Location 6	×		×	×	
Location 7	×		×		
Location 8	×		×		
Location 9					
Location 10			×		
Location 11			×		
Location 12					
Location 13					
Location 14					
Location 15					
Location 16	×	×	×	×	

En variabel skal deklareres før første anvendelse, og den bør deklareres i det snævrest mulige scope. Deklarerer man den i et bredere scope skal man desuden sikre sig at den ikke kan holde en utilsigtet værdi på tværs af flere eksekveringer af indre scopes.

Selvom rækkefølgen på klasseniveau er ligegyldig er reglen af deklaration skal være på samme linje som tildeling. Derudover dikterer konvention at dette bør foregå i top.

Java tillader ikke at man redeklarerer en variabel. Forsøger man dette vil programmet ikke kunne oversættes.

32 Standardbibliotek

32.1 Date

```
import java.util.Date;

class DateTest {
    public static void main (String[] args) {
        Date d = new Date();

        for (long elapsed=1000 ; elapsed<=1000000 ; elapsed *= 10) {
            d.setTime(elapsed);
            System.out.println(d.toString());
        }
    }
}
```

Ved kørsel fås:

```
Thu Jan 01 01:00:01 CET 1970
Thu Jan 01 01:00:10 CET 1970
Thu Jan 01 01:01:40 CET 1970
Thu Jan 01 01:16:40 CET 1970
```

Målt i sekunder er der en faktor 10 imellem hver af disse linjer. Det betyder at Java's tidsregning foretages i millisekunder og starter den 1. Januar 1970 kl 01:00 om natten.

32.2 Tidstagning

```
class Timing {
    static double base = 1.0000001;

    static double fun (double x, double y) {
        if (y<=1) {
            return x;
        } else {
            return fun(x, y-1)*fun(x, y-1);
        }
    }

    public static void main (String[] args) {
        for (double i=1 ; i<32 ; i++) {
            long   t0 = System.currentTimeMillis();
            double f  = fun(base, i);
            long   t1 = System.currentTimeMillis();

            System.out.println("fun("+base+","+i+") = "+f+
```

```
                                    " (calculation took "+(t1-t0)+"ms)");
        }
    }
}
```

Ja, denne implementation returnerer værdien af variablen `x` når den kaldes med som `fun(x,-1)`.

Men dette er ikke den eneste grund til at det er en dårlig algoritme. Hvis vi ser lidt nærmere på hvad der sker hvis vi skal udregne `fun(5,4)`, så udregnes `fun(5,3)` to gange. For hver af disse udregnes `fun(5,2)` to gange (det er 4 gange i alt), og for hver af disse "udregnes" `fun(5,1)` to gange (det er 8 gange i alt). Dette ville være endnu værre ved en større eksponent. Dette havde været fint hvis resultatet af disse udregninger afhang af noget eksternt der ikke kunne forudsiges, men i dette tilfælde er det åbenlyst at udregningen af `fun(5,3)` resulterer i den samme værdi hver gang. Man ville derfor se en gevaldig forbedring af ydelsen ved at udregne `fun(x,y-1)` en enkelt gang og gange denne værdi med sig selv.

33 Programudvikling

33.1 Indskrivningssystem

Den nye programspecifikation:

The `enrollment` system `contains` a `list` of `students` and `courses`. `It` can `display` the `list` of `students` which `are` enrolled in a particular `course`. It is possible to `enroll` `students` to a `course` and `remove` `them` from a `course`. It is possible to `add` and `remove`, both `students` and `courses`, to/from the `enrollment` system.

Den nye kravspecifikation med verb-noun analyse:

- ***Funktionelle krav:***

ID	*Navn*	*Beskrivelse*
F01	Vis liste over studerende	Det skal være muligt at få vist en list eaf studerende
F02	Vis liste over kurser	Det skal være muligt at få vist en liste af kurser
F03	Tilføj og fjern tilknytning til kurser	Det skal være muligt at associere studerende til et kursus og at fjerne eksisterende associationer
F04	Tilføj og fjern studerende	Det skal være muligt både at oprette studerende i og at fjerne dem fra systemet
F05	Tilføj og fjern kurser	Det skal være muligt både at oprette kurser i og at fjerne dem fra systemet

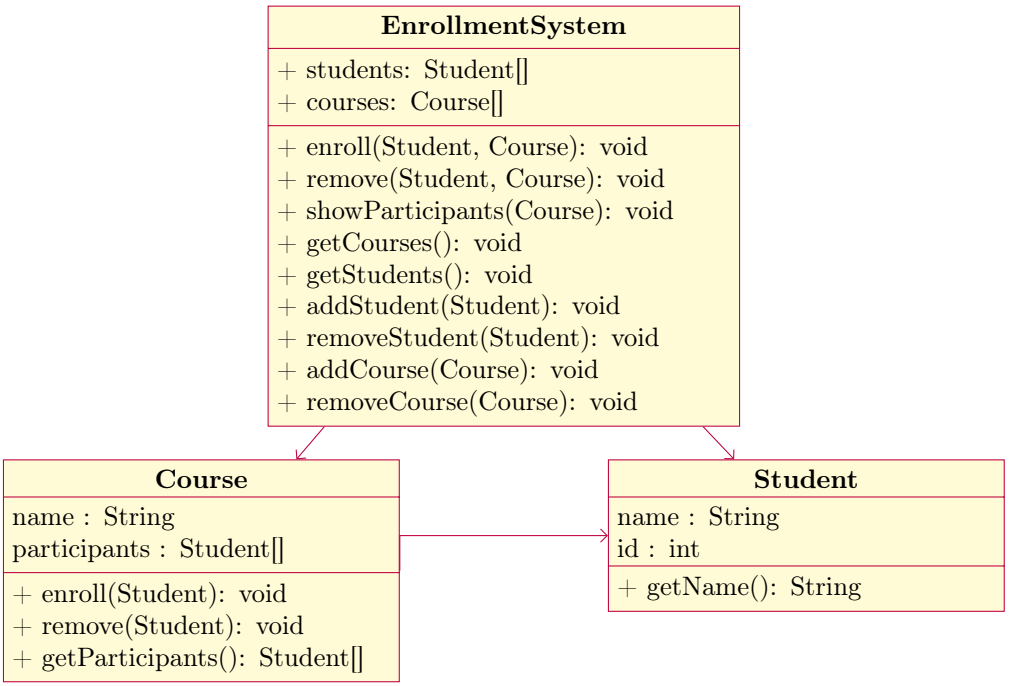
- ***Non-funktionelle krav:***

ID	*Navn*	*Beskrivelse*
NF01	Kapacitet	Systemet skal have plads til 10 kurser
NF02	Interoperabilitet	Systemet skal kunne integreres med Blackboard

Dette har ledt til cølgende CRC kort:

EnrollmentSystem	
Responsibilities	*Collaborators*
• Contains list of `Course`-objects • Contains list of `Student`-objects • Can enroll `Student`-objects in Course-objects • Can remove `Student`-objects from Course-objects • Can display `Student`-objects enrolled in any given `Course`-object • Can add `Student`-objects to the list of `Student`-objects • Can remove `Student`-objects from the list of `Student`-objects • Can add `Course`-objects to the list of `Course`-objects • Can remove `Course`-objects from the list of `Course`-objects	• Student • Course

Baseret på dette – og tilsvarende for de resterende klasser – er man nået frem til følgende klassediagram:



Og den endelige kode er det kun `EnrollmentSystem` der er ændret:

```
public class EnrollmentSystem
{
    public Student[] students;
    public Course[]  courses;

    public void enroll (Student student, Course course) {
        this.students = new Student[0];
        this.courses  = new Course[0];
        course.enroll(student);
    }

    public void remove (Student student, Course course) {
        course.remove(student);
    }

    public void showParticipants (Course course) {
        for (Student student: course.getParticipants()) {
            System.out.println(student.getName());
        }
    }

    public void getCourses () {
        System.out.println("void for a getter?");
    }

    public void getStudents () {
```

```
    System.out.println("void for a getter?");
}

public void addStudent (Student student) {
    // avoid duplicates
    for (Student entry: students) {
        if (entry==student) {
            return;
        }
    }

    // create new list
    Student[] newlist = new Student[students.length+1];
    for (int i=0 ; i<students.length ; i++) {
        newlist[i] = students[i];
    }
    newlist[students.length] = student;

    // override old reference
    students = newlist;
}

public void removeStudent (Student student) {
    int i;

    // find index of student
    for (i=0 ; i<students.length ; i++) {
        if (students[i] == student) {
            break;
        }
    }

    // guard: student not in list
    if (i==students.length) {
        return;
    }

    // create new list
    Student[] newlist = new Student[students.length-1];
    int n = 0;
    for (int o=0 ; o<students.length ; o++) {
        if (o!=i) {
            newlist[n++] = students[o];
        }
        o++;
    }
    newlist[students.length] = student;

    // override old reference
    students = newlist;
```

```
    }
}
```

34 Polymorfi

34.1 Lagersystem

```
class Item {
    private String name;
    private double price;

    Item (String name, double price) {
        this.name  = name;
        this.price = price;
    }

    public String getName () {
        return name;
    }

    public double getPrice () {
        return price;
    }
}

import java.util.Date;

class FoodItem extends Item {
    private Date expires;

    FoodItem (String name, double price, Date expires) {
        super(name, price);
        this.expires = expires;
    }

    public Date getExpires () {
        return expires;
    }

    @Override
    public String toString () {
        return "FoodItem name='"+getName()
             + "' price='"+getPrice()
             + "' expires='"+getExpires()+"'";
    }

    public static void main (String[] args) {
        FoodItem[] items = new FoodItem[10];
```

```
        for (int i=0 ; i<items.length ; i++) {
            items[i] = new FoodItem("Item "+i, 12.3*i,
                                    new Date(i*1000*60*60*24));
        }

        for (FoodItem item: items) {
            System.out.println(item);
        }
    }
}

import java.util.ArrayList;
import java.util.Arrays;

class NonFoodItem extends Item {
    private ArrayList<String> materials;

    NonFoodItem (String name, double price, ArrayList<String> materials) {
        super(name, price);
        this.materials = materials;
    }
    NonFoodItem (String name, double price, String[] materials) {
        super(name, price);
        this.materials = new ArrayList<String>(Arrays.asList(materials));
    }

    public ArrayList<String> getMaterials () {
        return materials;
    }

    @Override
    public String toString () {
        String m = "[";
        for (int i=0 ; i<materials.size() ; i++) {
            m += (i==0 ? "" : ",")+materials.get(i);
        }
        m += "]";
        return "NonFoodItem name='"+getName()
             + "' price='"+getPrice()
             + "' materials='"+m+"'";
    }

    public static void main (String[] args) {
        NonFoodItem[] items = new NonFoodItem[10];

        for (int i=0 ; i<items.length ; i++) {
            items[i] = new NonFoodItem("Item "+i, 12.3*i,
                                       new String[] {"butter", "cream"});
        }
```

```
        for (NonFoodItem item: items) {
            System.out.println(item);
        }
    }
}

import java.util.ArrayList;
import java.util.Date;

class Inventory {
    private ArrayList<Item> items;

    Inventory (ArrayList<Item> items) {
        this.items = items;
    }
    Inventory () {
        this(new ArrayList<Item>());
    }

    public void addItem (Item item) {
        if (!items.contains(item)) {
            items.add(item);
        }
    }

    public void removeItem (Item item) {
        items.remove(item);
    }

    public double getInventory () {
        double total = 0.0;

        for (Item item: items) {
            total += item.getPrice();
        }

        return total;
    }

    public void printInventory () {
        System.out.println("Inventory:");
        for (Item item: items) {
            System.out.println(" - "+item);
        }
    }

    public static void printStatus (Inventory inventory) {
        inventory.printInventory();
        System.out.println("Total: "+inventory.getInventory());
```

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

    public static void main (String args[]) {
        Inventory inventory = new Inventory();
        Item i1 = new Item("chocolate", 19.95);
        Item i2 = new Item("coffee", 24.95);
        Item i3 = new FoodItem("Milk", 12.95, new Date(12*1000*60*60*24));
        Item i4 = new NonFoodItem("USB Charger", 17.45,
                                  new String[] {"plastic", "stuff"});
        Item[] items = new Item[] {i1, i2, i3, i4};

        printStatus(inventory);
        for (Item item: items) {
            inventory.addItem(item);
            printStatus(inventory);
        }

        inventory.removeItem(i1);
        printStatus(inventory);
    }
}
```

35 Abstrakte Klasser og Interfaces

35.1 Lagersystem

Delopgave 1

```
abstract class Item {
    private String name;
    private double price;

    Item (String name, double price) {
        this.name  = name;
        this.price = price;
    }

    public String getName () {
        return name;
    }

    public double getPrice () {
        return price;
    }
}
```

Hvis vi forsøger at kalde constructoren for `Item` direkte så vil Jave give en fejl

på oversættelsestidspunktet. Dette sker fordi java på dette tidspunkt checker at reglen om at man ikke må instantiere en abstrakt klasse overholdes.

Delopgave 2

```
interface Expirable {
    public boolean isExpired ();
}
```

Delopgave 3

```
abstract class Item implements Expirable {
    private String name;
    private double price;

    Item (String name, double price) {
        this.name  = name;
        this.price = price;
    }

    public String getName () {
        return name;
    }

    public double getPrice () {
        return price;
    }
}
```

Delopgave 4

```
import java.util.Date;

class FoodItem extends Item {
    private Date expires;

    FoodItem (String name, double price, Date expires) {
        super(name, price);
        this.expires = expires;
    }

    public Date getExpires () {
        return expires;
    }

    @Override
    public String toString () {
        return "FoodItem name='"+getName()
             + "' price='"+getPrice()
```

```
                + "' expires='"+getExpires()+"'";
    }

    @Override
    public boolean isExpired () {
        return expires.compareTo(new Date()) < 0;
    }

    public static void main (String[] args) {
        FoodItem[] items = new FoodItem[10];

        for (int i=0 ; i<items.length ; i++) {
            items[i] = new FoodItem("Item "+i, 12.3*i,
                                    new Date(i*1000*60*60*24));
        }

        for (FoodItem item: items) {
            System.out.println(item);
        }
    }
}
```

Delopgave 5

```
import java.util.ArrayList;
import java.util.Date;

class Inventory {
    private ArrayList<Item> items;

    Inventory (ArrayList<Item> items) {
        this.items = items;
    }
    Inventory () {
        this(new ArrayList<Item>());
    }

    public void addItem (Item item) {
        if (!items.contains(item)) {
            items.add(item);
        }
    }

    public void removeItem (Item item) {
        items.remove(item);
    }

    public double getInventory () {
        double total = 0.0;
```

```
        for (Item item: items) {
            total += item.getPrice();
        }

        return total;
    }

    public void printInventory () {
        System.out.println("Inventory:");
        for (Item item: items) {
            System.out.println(" - "+item);
        }
    }

    public static void printStatus (Inventory inventory) {
        inventory.printInventory();
        System.out.println("Total: "+inventory.getInventory());
        System.out.println("");
    }

    public void removeExpiredFoods () {
        ArrayList<Item> copy = new ArrayList<Item>();
        for (int i=0 ; i<items.size() ; i++) {
            Item item = items.get(i);
            boolean expired = item.isExpired();
            if (!expired) {
                copy.add(item);
            }
        }
        items = copy;
    }

    public static void main (String args[]) {
        Inventory inventory = new Inventory();
        Item i1 = new FoodItem("chocolate", 19.95, new Date(52L*365*1000*60*60*24));
        Item i2 = new FoodItem("coffee"   , 24.95, new Date(53L*365*1000*60*60*24));
        Item i3 = new FoodItem("Milk"     , 12.95, new Date(    12*1000*60*60*24));
        Item i4 = new NonFoodItem("USB Charger", 17.45,
                                  new String[] {"plastic", "stuff"});
        Item[] items = new Item[] {i1, i2, i3, i4};

        printStatus(inventory);
        for (Item item: items) {
            inventory.addItem(item);
            printStatus(inventory);
        }

        inventory.removeItem(i1);
        printStatus(inventory);
```

```
        inventory.removeExpiredFoods();
        printStatus(inventory);
    }
}
```

36 Objekt-Orienteret Programmering

36.1 Gennemsnitlig Alder

Hvis du finder en løsning der *ikke* bryder med hvad der betragtes som god kodestil i Java må du godt henvende dig til forfatterne af denne opgavesamling.

36.2 Den Transsylvanske Fårehyrdeprøve

```
class Solver {
    static final int WINDOW_SIZE = 5;

    static final int N = 0;
    static final int R = 1;
    static final int L = 2;

    static final int M2 = 0b11 << (2*0);
    static final int M1 = 0b11 << (2*1);
    static final int N0 = 0b11 << (2*2);
    static final int P1 = 0b11 << (2*3);
    static final int P2 = 0b11 << (2*4);

    static final int M2N = encodeValues(N, 0);
    static final int M2R = encodeValues(R, 0);
    static final int M2L = encodeValues(L, 0);
    static final int M1N = encodeValues(N, 1);
    static final int M1R = encodeValues(R, 1);
    static final int M1L = encodeValues(L, 1);
    static final int N0N = encodeValues(N, 2);
    static final int N0R = encodeValues(R, 2);
    static final int N0L = encodeValues(L, 2);
    static final int P1N = encodeValues(N, 3);
    static final int P1R = encodeValues(R, 3);
    static final int P1L = encodeValues(L, 3);
    static final int P2N = encodeValues(N, 4);
    static final int P2R = encodeValues(R, 4);
    static final int P2L = encodeValues(L, 4);

    Path path;

    public Solver (Path path) {
        this.path = path;
```

```
}

static private int encodeValues (int state, int local_position) {
    return state << (2*local_position);
}

private int encodePosition (int global_position, int local_position) {
    Sheep sheep = path.peek(global_position);
    int state = (sheep==null ? 0 : (sheep.isRightbound() ? 1 : 2));
    return encodeValues(state, local_position);
}

private int encodeWindow (int offset) {
    int result = 0;
    for (int i=0 ; i<WINDOW_SIZE ; i++) {
        result += encodePosition(offset+i, i);
    }
    return result;
}

static private void printEncoding (int e) {
    System.out.print("[");
    for (int i=0 ; i<WINDOW_SIZE ; i++) {
        int entry = (e & (0b11 << (2*i))) >> 2*i;
        System.out.print(entry==0 ? "N" : (entry==1 ? "R" : "L"));
    }
    System.out.println("]");
}

public void solve () throws DeadSheepException {
    boolean done = false;
    while (!done) {
        boolean change = false;

        for (int i=1-WINDOW_SIZE ; i<path.getSpotcount() ; i++) {
            int e = encodeWindow(i);
            if (false) {
            } else if ((e&M1)==M1R && (e&N0)==NON && (e&P1)==P1L && (e&P2)==P2L) {
                Sheep sheep = path.peek(i+3);
                sheep.walk();
                change = true;
                break;
            } else if ((e&M2)==M2R && (e&M1)==M1L && (e&N0)==NON && (e&P1)==P1L) {
                Sheep sheep = path.peek(i+0);
                sheep.jump();
                change = true;
                break;
            } else if ((e&M2)==M2R && (e&M1)==M1N && (e&N0)==N0L && (e&P1)==P1R) {
                Sheep sheep = path.peek(i+0);
                sheep.walk();
```

```
                    change = true;
                    break;
                } else if ((e&M1)==M1R && (e&N0)==N0N && (e&P1)==P1R && (e&P2)==P2L) {
                    Sheep sheep = path.peek(i+4);
                    sheep.jump();
                    change = true;
                    break;
                } else if ((e&M2)==M2N && (e&N0)==N0N && (e&P1)==P1R && (e&P2)==P2L) {
                    Sheep sheep = path.peek(i+4);
                    sheep.jump();
                    change = true;
                    break;
                } else if ((e&M2)==M2N && (e&N0)==N0N && (e&P1)==P1L) {
                    Sheep sheep = path.peek(i+3);
                    sheep.walk();
                    change = true;
                    break;
                } else if ((e&M2)==M2R && (e&N0)==N0N && (e&P2)==P2N) {
                    Sheep sheep = path.peek(i+0);
                    sheep.jump();
                    change = true;
                    break;
                } else if (false) {
                }
            }

            if (!change) done = true;
        }
    }
}
```

Næste opgave er så at forstå hvordan den virker ;-)

36.3 Hangman

En referenceløsning vil blive gjort tilgængelig uafhængig af dette dokument.

36.4 Den Omrejsende Salgsmand

Nedenstående program bruger – på min maskine – ca. 70s på at finde den bedste løsning, og printer den derefter ud som:

```
Best solution found, at 864km:
  0. Esbjerg
  1. Herning
  2. Silkeborg
  3. Viborg
  4. Aalborg
```

```
 5. Randers
 6. Århus
 7. Helsingør
 8. København
 9. Roskilde
10. Næstved
11. Odense
12. Horsens
13. Vejle
14. Kolding
15. Esbjerg
```

Undervejs bliver metoden `TSP.solve` kaldt 270709035 (i.e., 270 millioner) gange. Dette er på en 2.4GHz CPU. Over de 70s er der derfor blevet udført omkring 168 milliarder klokcykler, eller i gennemsnit ca. 620 klokcykler per kald til metoden `TSP.solve`.

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

public class City
{
    private String name;
    private Map<City, Integer> distances;

    public City (String name) {
        this.name = name;
        this.distances = new HashMap<City, Integer>();
    }

    public void connect (City city, int distance) {
        distances.put(city, distance);
    }

    public long getDistance (City city) {
        return distances.get(city);
    }

    public String getName () {
        return name;
    }
}
```

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

class Network {
    Map<String, Integer> name2index;
    Map<Integer, String> index2name;
```

```
City[] cities;

public Network () {
    String[] citynames = new String[] {
        "Esbjerg",
        "Helsingør",
        "Herning",
        "Horsens",
        "Kolding",
        "København",
        "Næstved",
        "Odense",
        "Randers",
        "Roskilde",
        "Silkeborg",
        "Vejle",
        "Viborg",
        "Aalborg",
        "Århus",
    };

    cities = new City[citynames.length];
    name2index = new HashMap<String, Integer>();
    index2name = new HashMap<Integer, String>();
    for (int i=0 ; i<citynames.length ; i++) {
        name2index.put(citynames[i], i);
        index2name.put(i, citynames[i]);
        cities[i] = new City(citynames[i]);
    }

    // source: https://dk.afstand.org
    int[][] distances = {
        {  0, 269,  82,  98,  65, 260, 211, 123, 149, 230, 105,  74, 126, 198, 134},
        {269,   0, 226, 173, 206,  40, 105, 157, 166,  55, 191, 196, 207, 200, 150},
        { 82, 226,   0,  63,  79, 230, 202, 121,  76, 202,  36,  59,  45, 117,  77},
        { 98, 173,  63,   0,  48, 172, 139,  62,  68, 142,  40,  26,  75, 132,  40},
        { 65, 206,  79,  48,   0, 196, 148,  59, 114, 165,  77,  25, 110, 176,  88},
        {260,  40, 230, 172, 196,   0,  71, 141, 180,  31, 196, 190, 218, 224, 156},
        {211, 105, 202, 139, 148,  71,   0,  89, 174,  50, 175, 150, 204, 232, 142},
        {123, 157, 121,  62,  59, 141,  89,   0, 121, 110, 102,  64, 136, 186,  86},
        {149, 166,  76,  68, 114, 180, 174, 121,   0, 156,  44,  90,  42,  65,  36},
        {230,  55, 202, 142, 165,  31,  50, 110, 156,   0, 169, 160, 193, 206, 130},
        {105, 191,  36,  40,  77, 196, 175, 102,  44, 169,   0,  53,  35,  99,  41},
        { 74, 196,  59,  26,  25, 190, 150,  64,  90, 160,  53,   0,  86, 151,  65},
        {126, 207,  45,  75, 110, 218, 204, 136,  42, 193,  35,  86,   0,  72,  63},
        {198, 200, 117, 132, 176, 224, 232, 186,  65, 206,  99, 151,  72,   0, 101},
        {134, 150,  77,  40,  88, 156, 142,  86,  36, 130,  41,  65,  63, 101,   0},
    };
    for (int src=0 ; src<cities.length ; src++) {
        for (int dst=0 ; dst<cities.length ; dst++) {
```

```
                if (dst==src) continue;
                cities[src].connect(cities[dst], distances[src][dst]);
            }
        }
    }

    public City[] getCities () {
        return cities;
    }
}

class TSP {
    private static boolean contains (City[] cities, City city) {
        for (City candidate: cities) {
            if (candidate == city) return true;
        }

        return false;
    }

    private static void print (long distance, City[] solution) {
        System.out.println("Best solution found, at "+distance+"km:");
        for (int i=0 ; i<solution.length ; i++) {
            System.out.printf(" %2d. %s", i, solution[i].getName());
            System.out.println("");
        }
    }

    public static long solve (City[] cities, int depth,
                              City[] best_path, long best_distance,
                              City[]      path, long      distance) {
        if (depth==path.length-1) {
            distance += path[depth-1].getDistance(path[depth]);
            if (distance<best_distance) {
                best_distance = distance;
                for (int i=0 ; i<path.length ; i++) best_path[i] = path[i];
            }
        } else {
            for (City city: cities) {
                if (contains(path, city)) continue;

                long stepsize = path[depth-1].getDistance(city);

                // add
                path[depth] = city;
                distance += stepsize;
                depth++;

                // recurse
                if (distance<best_distance) {
```

```
                    best_distance = solve(cities, depth,
                                          best_path, best_distance,
                                          path, distance);
                }

                // remove
                depth--;
                distance -= stepsize;
                path[depth] = null;
            }
        }
        return best_distance;
    }

    public static void main (String[] args) {
        Network network = new Network();
        City[] cities = network.getCities();

        // initialize any solution
        City[] solution   = new City[cities.length+1];
        for (int i=0 ; i<solution.length ; i++) {
            solution[i] = cities[i%cities.length];
        }
        long best_distance = 0;
        for (int i=0 ; i<cities.length ; i++) {
            best_distance += solution[i].getDistance(solution[i+1]);
        }

        // initialize scratchpad
        City root = cities[0];
        City[] scratchpad = new City[cities.length+1];
        scratchpad[0] = root;
        scratchpad[cities.length] = root;

        // solve
        best_distance = solve(cities, 1, solution, best_distance, scratchpad, 0);

        // present result
        print(best_distance, solution);
    }
}
```

37 Exceptions

37.1 Lagersystem

Delopgave 1

```
class ExpiredItemAddedException extends Exception {
    public ExpiredItemAddedException () {
        super("Attempted to add expired product to database");
    }
}
```

Delopgave 2

```
import java.util.Date;

class FoodItem extends Item {
    private Date expires;

    FoodItem (String name, double price, Date expires) throws ExpiredItemAddedException {
        super(name, price);
        if (expires.compareTo(new Date())<0) {
            throw new ExpiredItemAddedException();
        }
        this.expires = expires;
    }

    public Date getExpires () {
        return expires;
    }

    @Override
    public String toString () {
        return "FoodItem name='"+getName()
             + "' price='"+getPrice()
             + "' expires='"+getExpires()+"'";
    }

    @Override
    public boolean isExpired () {
        return expires.compareTo(new Date()) < 0;
    }

    public static void main (String[] args) {
        FoodItem[] items = new FoodItem[10];

        for (int i=0 ; i<items.length ; i++) {
            try {
                items[i] = new FoodItem("Item "+i, 12.3*i,
                                        new Date((50L+i)*365*1000*60*60*24));
            } catch (ExpiredItemAddedException e) {
                items[i] = null;
                System.out.println(e.getMessage());
            }
        }
```

```
        for (FoodItem item: items) {
            System.out.println(item);
        }
    }
}
```

Delopgave 3

```
import java.util.ArrayList;
import java.util.Date;

class Inventory {
    private ArrayList<Item> items;

    Inventory (ArrayList<Item> items) {
        this.items = items;
    }
    Inventory () {
        this(new ArrayList<Item>());
    }

    public void addItem (Item item) {
        if (!items.contains(item)) {
            items.add(item);
        }
    }

    public void removeItem (Item item) {
        items.remove(item);
    }

    public double getInventory () {
        double total = 0.0;

        for (Item item: items) {
            if (item==null) continue;
            total += item.getPrice();
        }

        return total;
    }

    public void printInventory () {
        System.out.println("Inventory:");
        for (Item item: items) {
            if (item==null) continue;
            System.out.println(" - "+item);
        }
    }
```

```
    public static void printStatus (Inventory inventory) {
        inventory.printInventory();
        System.out.println("Total: "+inventory.getInventory());
        System.out.println("");
    }

    public void removeExpiredFoods () {
        ArrayList<Item> copy = new ArrayList<Item>();
        for (int i=0 ; i<items.size() ; i++) {
            Item item = items.get(i);
            if (item==null) continue;
            boolean expired = item.isExpired();
            if (!expired) {
                copy.add(item);
            }
        }
        items = copy;
    }

    private static FoodItem safeFoodItem (String name, double price, Date expires) {
        try {
            return new FoodItem(name, price, expires);
        } catch (ExpiredItemAddedException e) {
            return null;
        }
    }

    public static void main (String args[]) {
        Inventory inventory = new Inventory();
        Item i1 = safeFoodItem("chocolate", 19.95, new Date(53L*365*1000*60*60*24));
        Item i2 = safeFoodItem("coffee"   , 24.95, new Date(54L*365*1000*60*60*24));
        Item i3 = safeFoodItem("Milk"     , 12.95, new Date(    12*1000*60*60*24));
        Item i4 = new NonFoodItem("USB Charger", 17.45,
                                  new String[] {"plastic", "stuff"});
        Item[] items = new Item[] {i1, i2, i3, i4};

        printStatus(inventory);
        for (Item item: items) {
            inventory.addItem(item);
            printStatus(inventory);
        }

        inventory.removeItem(i1);
        printStatus(inventory);

        inventory.removeExpiredFoods();
        printStatus(inventory);
    }
}
```

I denne løsning er der introduceret en `safeFoodItem` metode som *konverterer* den mulige exception til en `null` reference.

Der er mindst to måder hvormed vi kan undgå at følge `null` referencer. Den ene er at sørge for at de aldrig bliver tilføjet til vores `inventory` i `main` metoden. Den anden er at sørge for at de bliver sprunget over. Den sidste er valgt i ovenstående.

Delopgave 4

Hvis `FoodItem` skal kunne bruges udenfor `Inventory` til at repræsentere for gamle madvarer så vil det være en desideret fejl at kaste en exception i constructoren for `FoodItem`.

Man kunne som et designvalg placere sin sikkerhedsbarriere i `Inventory`. Men hvis checket udføres her, kan man begynde at overfeje om løsningen overhovedet skal indeholde en exception.

Udgangspunktet for denne diskussion må være:

- Skal det være muligt at repræsentere madvarer der er for gamle?
- Hvor har vi mulighed for at beslutte om en madvare er for gammel?
- Hvor er der behov for at vide at en madvare er for gammel?
- Hvor kan den nødvendige sikkerhed implementeres uden at inføre problematiske restriktioner?
- Hvor vil det være overskueligt at implementere den nødvendige sikkerhed?

Den rigtige løsning ligger i overlappet af svarene til disse spørgsmål.