INF1000 : Forelesning 2

Variable og tilordninger
Heltall, desimaltall og sannhetsverdier
Kompilering og kjøring
Utskrift på skjerm

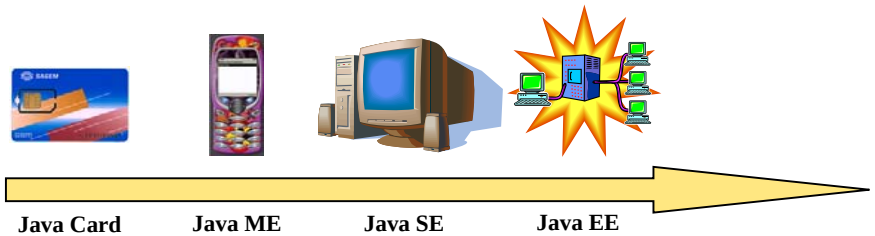
Ole Christian Lingjærde
Gruppen for bioinformatikk
Institutt for informatikk
Universitetet i Oslo

Java

- **Java**:
  - programmeringsspråk med grammatiske regler, semantikk, osv
  - programsystemet som må være installert for å kompilere og kjøre Java-programmer.
- Het opprinnelige Oak og skulle brukes for å styre digitale "dingser" som spillkonsoll, digital kabel-TV (*video on demand*), osv. Dette ble ingen suksess.
- På denne tiden kom Internett for fullt, og Java-teamet brukte Oak/Java teknologien til å lage en nettleser i 1994 som de kalte WebRunner (etter filmen Blade Runner), senere omdøpt til HotJava. Første nettleser med dynamisk innhold og animasjon!
- Java er gratis og kan lastes ned fra http://java.sun.com (og fra Ifi-CD'en).
- Versjonen som brukes nå kalles *Java 2 plattformen.*


James Gosling

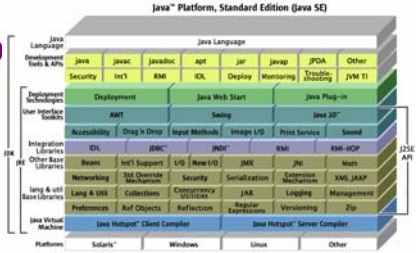
Ulike varianter for ulike behov



Java Standard Edition (Java SE)

To sentrale begreper:

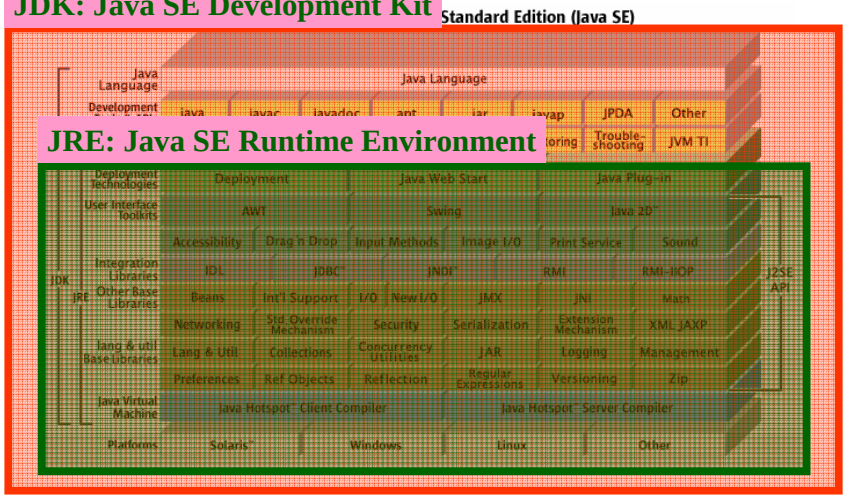
- **Java SE Runtime Environment (JRE)**
  System for å kjøre kompilerte Java-programmer.
- **Java SE Development Kit (JDK)**
  JRE + programmer for å kompilere, feilsøke ("debugge") og dokumentere Java-programmer.



I tillegg til dette kan man legge til egne "pakker" som gir økt funksjonalitet.
I INF1000 brukes det en slik pakke, med navn easyIO.

Java Standard Edition (Java SE)

JDK: Java SE Development Kit



Installasjon av Java på egen maskin

Hvis maskinen din mangler Java, må du gjennom to steg:

1) **Installere Java SE Development Kit**

   Kan lastes ned fra Ifi-CD'en eller fra nettet. I siste tilfelle går du til siden http://java.sun.com/javase/downloads/index.jsp og velger nedlasting av JDK 5.0 Update N (N = versjonsnr).

2) **Installere INF1000-pakken easyIO**

   Dette gir Java tilleggsfunksjonalitet som benyttes i undervisningen. Kan lastes ned fra Ifi-CD'en eller fra nettet. I siste tilfelle går du til siden http://www.universitetsforlaget.no/java

   Merk: varianten av easyIO som ligger på Ifi-CD'en inneholder noen tillegg som du kan komme til å møte i undervisningen.

Lagre, kompilere, kjøre

- Anta at vi har skrevet et Java-program. Eksempel:

```
class MittProgram {
  public static void main (String [] args) {
    int u;
    u = 2;
  }
}
```

- For å få datamaskinen til å utføre programmet for oss, må vi

1) Lagre programmet som **`MittProgram.java`**

2) Kompilere programmet:

   > **`javac MittProgram.java`**

   Sjekker du innholdet i filkatalogen etterpå, ser du at maskinen har laget en ny fil som heter `MittProgram.class` (det kompilerte programmet).

3) Kjøre programmet:

   `> java MittProgram`

Kompilere

```
class MittProgram {
  public static void main  (String [ ] args) {
     .... osv
}
```

Java programtekst - filen må ha samme navn som den første klassen på filen

Filnavn: MittProgram.java

Kompilert Java-program

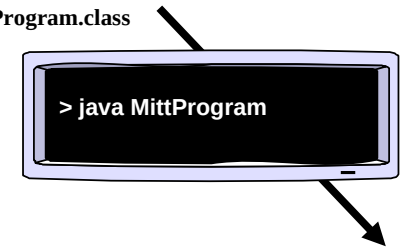
```
Êþº¾? ?-?
? ? ?
 ?  ? <init> ? ()V ? Code ? LineNumberTable ? main ? ([Ljava/lang/String;)V ?
SourceFile ? MittProgram.java? ?  ?
MittProgram ? java/lang/Object?
? ? ????? ??? ? ? ? ??? ? ? ??? *·? ±??? ? ??? ? ??? ?      ? ?
                ? ? ??? ? ? ???  <±??? ? ???
? ??? ? ? ? ?
??? ?
```

Filnavn: MittProgram.class

Kjøre

```
Êþº¾? ?-?
? ? ?
 ?  ? <init> ? ()V ? Code ? LineNumberTable ? main ? ([Ljava/lang/String;)V ?
SourceFile ? MittProgram.java? ?  ?
MittProgram ? java/lang/Object? ? ? ????? ??? ? ? ? ??? ? ? ??? *·? ±??? ? ??? ? ??? ?
          ? ?         ? ? ??? ? ? ???  <±??? ? ???
? ??? ? ? ? ?
??? ?
```

Filnavn: MittProgram.class



Det kompilerte programmet lastes inn i maskinens primærlager og eksekveres

Programmet utføres

Kompilere og kjøre i Unix, Windows, MacOS

- **Unix:**
  - Start et terminalvindu (xterm-vindu)
  - Endre filområde (directory) til der programfilen ligger
  - For å kompilere: javac MittProgram.java
  - For å kjøre: java MittProgram
- **Windows:**
  - Start et kommandovindu ved å gå inn i Start-menyen og velge
    - `MS-DOS Prompt` (Window 95 og 98)
    - `Command Prompt` (Windows NT, 2000 og XP)

    (hvis skjult, se under All Programs / Accessories eller tilsvarende)
  - Resten som for Unix-baserte plattformer.
- **Mac OS:**
  - Se nettsiden http://java.sun.com/docs/books/tutorial/getStarted/TOC.html

Bestanddelene i et Java-program

Alle programmer må starte med **class** (det kan stå public foran)

Dette er programmets navn og kan velges fritt av oss

Her starter innmaten i programmet

```
class MittProgram {
  public static void main (String[] args) {
    int u;
    u = 2;
  }
}
```

Forteller at programmet er kjørbart

Her slutter innmaten i programmet

Her er instruksjonene i programmet

Mer om denne linjen senere - men merk at vi alltid trenger den

Eksempel

- Instruksjon til en datamaskin (i programmeringsspråket Java):

```
class Kalkulator {
  public static void main(String [] args) {
    System.out.println("Beløp: 500 NOK");
    System.out.println("Kurs : 6.5");
    System.out.println("Beløp: " + 500/6.5 + " USD");
  }
}
```

Det er ikke meningen at du skal forstå Java-eksemplet i detalj nå, men merk at:

- setninger i Java har en helt annen utforming enn setninger i norsk
- instruksjoner i Java må følge en presis syntaks (=setningsbygning), og en eneste trykkfeil (f.eks. at det står **clas** i stedet for **class**) vil gjøre at det ikke blir forstått

Utskrift på skjermen

- Skrive ut verdien til en variabel u (int, double, boolean, ...) på skjermen i det vinduet programmet startes fra:

```
System.out.print(u);      // Skriver ut verdien til u
System.out.println(u);    // Skriver ut verdien til u og starter ny linje
```

- Skrive ut tekst på skjermen:

```
System.out.print("Velkommen til INF 1000");
System.out.println("Velkommen til INF 1000");
```

- Skrive ut tekst i et eget vindu på skjermen:

```
JOptionPane.showMessageDialog(null, "Velkommen til INF 1000");
```

Denne krever at det i starten av programmet (før linjen med class) står

```
import javax.swing.*;
```


Hva slags instruksjoner kan vi gi?

- Begrenset vokabular: Java har et høyst begrenset vokabular, og hver enkelt instruksjon får typisk datamaskinen til å gjøre en veldig enkel ting.
- Derfor: for å få datamaskinen til å løse komplekse oppgaver må vi i de fleste tilfeller gi veldig mange instruksjoner.
  - Eksempel: en moderne kopimaskin kan inneholde programmer med hundretusener av instruksjoner
  - I dette kurset kommer vi ikke så langt, men vi skal etterhvert lage programmer med mange hundre linjer
- Hva slags instruksjoner kan vi så gi?

Eksempel: ta vare på og regne med tall

- Sette av (reservere) plass i datamaskinens hukommelse til et heltall:

```
int lengde;
```

slutt på instruksjonen

sett av plass til et heltall (engelsk: **int**eger)

gi plassen navnet "lengde"

- Fylle plassen med en verdi (et tall):

```
lengde = 14;
```

leses "settes lik" eller "gis verdien"

- Avlese/bruke verdien:

```
int svar;
svar = lengde * 2;
```


Variabler

- Når vi har gitt instruksjonen

```
int lengde;
```

så kan vi endre verdien på denne plassen så mange ganger vi vil, f.eks.:

```
lengde = 14;
lengde = 434;
lengde = lengde + 2;
lengde = lengde;
```

Siden verdien kan variere over tid, kalles `lengde` en variabel.

- Hva skjer egentlig når vi skriver `lengde = lengde + 2;` ? I detalj:
  1. Verdien som ligger i variabelen `lengde` hentes fram (f.eks. 434)
  2. En ny verdi regnes ut ved å legge til 2 (434 + 2 = 436)
  3. Variabelen `lengde` gis den nye verdien (436)

Variabel-deklarasjoner

- Instruksjoner av typen

```
int alder;
int vekt;
int personnummer;
```

kalles variabel-deklarasjoner.

- Vi kunne erstattet de tre instruksjonene ovenfor med:

```
int alder, vekt, personnummer;
```
(NB: komma mellom variablene)

Normalt er det ryddigere å bruke den første varianten, men du møter begge i kurset.

Variabeldeklarasjoner

- Variable kan deklareres hvor som helst i et program, og de kan endres hvor som helst etter at de er deklarert.
- Variable har ingen verdi rett etter en deklarasjon:

```
int lengde;
lengde = lengde + 1;        // Ulovlig!
```

- Vi kan gi variable en verdi når vi deklarerer dem:

```
int lengde = 4;
lengde = lengde + 1;        // Lovlig
```

- Vi kan også vente med å gi en variabel verdi:

```
int lengde;
.....
lengde = 4;
lengde = lengde + 1;        // Lovlig
```


Hvis du glemmer å initialisere en variabel

Forsøk på å kompilere et program med en slik feil:

```
C:\Eksempel.java:4: variable lengde might not have been
   initialized
    lengde = lengde + 1;
             ^
1 error

Tool completed with exit code 1
```

Vanlig feil, så lær deg å kjenne igjen denne feilmeldingen.

Avsluttende om variable

- Unngå i størst mulig utstrekning å samle mange variabeldeklarasjoner på en linje:

```
int år, måned, dag, alder;
```
Uoversiktlig

```
int år;    // Fødselsår
int måned; // Fødselsmåned (1-12)
int dag;   // Fødselsdato (1-31)
int alder; // Alder i antall år
```
Oversiktlig

- Deklarer variable først når du trenger dem – ingen grunn til å samle alle variabeldeklarasjoner ett sted med mindre de naturlig hører sammen.

Fullstendige programeksempler

Vi skal nå se på endel programeksempler.

- Noen av programmene gir kompileringsfeil, dvs vi får en feilmelding når vi forsøker å kompilere programmet med javac.
- Noen av programmene gir kjørefeil, dvs vi får en feilmelding når vi forsøker å kjøre programmet med java.
- Noen av programmene virker som de skal.

NB: alle som programmerer opplever fra tid til annen å få kompileringsfeil og kjørefeil. Det er nyttig å lære seg å lese feilmeldingene man får, som ofte (men ikke alltid!) gir en klar pekepinn om hva man har gjort feil.

Eksempel 1: kompileringsfeil

```
class IkkeRiktig {
   public static void main (String [] args) {
      double x;
      int y;
      x = 2;
      y = x;    // Her prøver vi å sette y lik et desimaltall
   }
}
```

```
FEIL UNDER KOMPILERING
IkkeRiktig.java:6: possible loss of precision
found   : double
required: int
       y = x;
           ^
1 error
```


Eksempel 2: kompileringsfeil

```
class MerFeil {
   public static void main (String [] args) {
      boolean b;
      b = 2;    // Her prøver vi å sette b lik et heltall
   }
}
```

```
FEIL UNDER KOMPILERING
MerFeil.java:4: incompatible types
found   : int
required: boolean
       b = 2;
           ^
1 error
```


Eksempel 3: kompileringsfeil

```
class EndaMerFeil {
   public static void main (String [] args) {
      int x, y;
      y = x;    // Her prøver vi å sette y lik en udefinert verdi
   }
}
```

```
FEIL UNDER KOMPILERING
EndaMerFeil.java:4: variable x might not have been initialized
       y = x;
           ^
1 error
```

Eksempel 4: kjørefeil

```
class Kjorefeil {
   public static void main (String [] args) {
      int x = 3, y = 0, z;
      z = x / y;  // Her prøver vi å dele på null
   }
}
```

FEIL UNDER KJØRING

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
                at Kjorefeil.main(Kjorefeil.java:4)
```


Variabel-tilordninger

- Instruksjoner av typen

```
alder = 3;
```

kalles variabel-tilordninger (eller bare tilordninger).

- Generell form:

```
variabel = uttrykk;
```

her må det stå navnet på en variabel som er deklarert

her må det stå en verdi eller et regneuttrykk. To eksempler:
345
(56+36) * 14 - 3 + 53

- Først utføres regnestykket på høyresiden av = , og deretter settes variabelen på venstre side av = lik den utregnede verdien

Vi kan ha mange variable

- I et program kan vi deklarere så mange variable vi vil, f.eks.

```
int alderKari;
int alderPer;
int alderOla;
int sumAlder;
alderKari = 20;
alderPer = 10 + alderKari;
alderOla = 10 + alderKari + alderPer;
sumAlder = alderKari + alderPer + alderOla;
```

- Hvilken verdi har de fire variablene når alle instruksjonene ovenfor er utført?

alderKari:	20
alderPer:	30
alderOla:	60
sumAlder:	110

Ting å passe på

- Vi kan ikke deklarere flere variable med samme navn. Dette er ulovlig:

```
int alder;
int alder;      (Ulovlig - variabelen alder er allerede deklarert!)
```

- En variabel kan hete hva som helst, men bruk bare bokstaver og tall, og begynn alltid navnet med en bokstav. Eksempler:

```
int etVeldigLangtVariabelNavn;      (Denne en lovlig)
int år2001;                         (Denne er også lovlig)
int 2001år;                         (Denne er ikke lovlig)
```

Eksempel: bytte om verdien til variable

- Anta at vi har disse instruksjonene:

```
int første, andre;
første = 65;
andre = 77;
```

- Hvordan kan vi bytte om verdiene i de to variablene?

  Vi forsøker dette:

```
første = andre;
andre = første;
```

  Vil dette virke?

  Se boksen til høyre →

Når vi har utført...	så er verdien til:
`første = 65;` `andre = 77;`	`første:` 65 `andre :` 77
`første = andre;`	`første:` 77 `andre :` 77
`andre = første;`	`første:` 77 `andre :` 77

Løsning

- Problemet var at vi mistet den opprinnelige verdien til første når vi utførte

```
første = andre;
```

- Vi kan løse problemet ved å ta vare på den opprinnelige verdien i en tredje variabel. Alle instruksjonene:

```
int første, andre, hjelpevar;
første = 65;
andre = 77;

hjelpevar = første;
første = andre;
andre = hjelpevar;
```

- Vi sjekker at det virker →

Når vi har utført...	så er verdien til:
`første = 65;` `andre = 77;`	`første:` 65 `andre :` 77 `hjelpevar:` ---
`hjelpevar=første;` `første = andre;`	`første:` 77 `andre:` 77 `hjelpevar:` 65
`andre = hjelpevar;`	`første:` 77 `andre :` 65 `hjelpevar:` 65

Desimaltall

- Variable av typen int kan bare holde heltallsverdier (...-2, -1, 0, 1, 2, ...)
- Hvis vi ønsker å lagre desimaltall (også kalt flyttall) kan vi bruke double:

```
double pi = 3.14;
double radius = 0.332;
double omkrets = 2 * pi * radius;
```

- Vi kan godt gi et heltall som verdi til en double-variabel:

```
double radius = 2;
```

  ...men inne i datamaskinen vil det bli lagret med desimaler: 2.0000.....

- Eksempel:

```
int radius = 2;                 (Tallet 2 som heltall)
double nyradius = radius;       (Tallet 2 som desimaltall)
```

- Desimaltall kan angis på flere måter:

```
-10.5    .435    15.     1.23e5   1.23e+4   1.15e-3
```


Datatyper vi kommer til å benytte

Datatype	Beskrivelse	Eksempel
int	heltall	`int k = 3;`
double	desimaltall	`double x = 3.14;`
boolean	sannhetsverdi	`boolean b = true;`
char	tegn	`char c = '@';`
String	tekst	`String s = "Hei på deg";`

+ noen flere (short, long, byte, float)

De numeriske datatypene

- int og double er eksempler på numeriske datatyper
- Java har ialt seks numeriske datatyper:

Datatype	Lovlige verdier
byte	{-128, -127, ..., 127}
short	{-32768, ..., 32767}
int	$\{-2^{31}, ..., 2^{31}-1\}$
long	$\{-2^{63}, ..., 2^{63}-1\}$
float	(-3.4e38, 3.4e38)
double	(-1.7e308, 1.7e308)

I praksis er det disse to du trenger i INF 1000

- Antall signifikante siffer er 6-7 med float og 14-15 med double.

Sannhetsverdier

- I programmering har vi ofte behov for å vite om noe er sant (= true) eller usant (= false), f.eks. om det er sant eller usant at x > 0.
- Derfor finnes det en egen variabeltype som bare kan holde de to verdiene true og false. Denne typen heter boolean:

```
boolean b;
b = true;
```

```
boolean pos;
int u = -3;
pos = (u > 0);
```

- Hva skjer egentlig i siste linje i det høyre eksemplet ovenfor? I detalj:
  - Datamaskinen "regner ut" verdien til (u > 0). Siden u > 0 er usant, blir verdien til (u > 0) lik false.
  - Til slutt blir variabelen pos satt lik den utregnede verdien
  - Derfor har pos verdien false når alle instruksjonene er utført.

Aritmetiske og logiske uttrykk

- Aritmetiske uttrykk:

  2 * (3+4) / 1.5

  2 / (12 + 34 - 2.3)

- Logiske uttrykk:

Uttrykket	har verdien true hvis	....og verdien false ellers
x < y	x mindre enn y	
x <= y	x mindre enn eller lik y	
x = = y	x lik y	
x != y	x ikke lik y	
x > y	x større enn y	
x >= y	x større enn eller lik y	
! (x < y)	ikke x mindre enn y	
b1 && b2	både b1 og b2 sann	
b1 || b2	b1 eller b2 (eller begge) sann	

Utregning av sammensatte uttrykk

- * og / binder sterkere enn + og -:

  2 + 5 * 4 - 3 **er lik** 2 + (5*4) - 3

- && binder sterkere enn || :

  b1 || b2 && b3 **er lik** b1 || (b2 && b3)

- Både && og || binder sterkere enn = = :

  b1 && b2 = = b3 || b4 **er lik** (b1 && b2) = = (b3 || b4)

- Punktene ovenfor er eksempler på presedensregler, dvs regler for hvilke operatorer som har presedens (fortrinn, førsterett) ved utregning av sammensatte uttrykk. F.eks. har * presedens over +.

Oppgave

```
boolean b1, b2;
double x, y;
int z;

x = 45.33;
y = x + 1;
z = 0;

b1 = (x < y) && (z == 0);
b2 = false || b1;
```

Hva er verdien til b1 og til b2 etter at setningene over er utført?

b1: **true**

b2: **true**

Greit å vite

- Multiplikasjon må alltid angis eksplisitt med *:
  - `int prod = 10 a;` // feil!!
  - `int prod = 10 * a;` // riktig
- Det er forskjell på = og = = :
  - `=` brukes for å sette verdien til en variabel
  - `==` brukes for å sammenlikne to verdier
- Hvis vi har variabelen `boolean b` så er det ingen forskjell på
  - `b == true`
  - `b`
- Ekstra parenteser kan øke leseligheten for mennesker:
  - `b = x == y;` betyr det samme som `b = (x == y);`

Kommentarer i programmer

- For å lage programmene mer forståelige, legger vi inn kommentarer i programteksten
- Kommentarer oversettes ikke: kompilatoren hopper over dem
- To typer kommentarer:

```
// Her er en kommentar som varer ut linja

/* Her er en kommentar som
   varer
   helt til hit */
```

- Gode programmer har kommentarer, men ikke på hver linje – bruk kommentarer når det er ting dere ønsker å rette oppmerksomheten mot, f.eks. sentrale punkter i programmet eller spesielt vanskelige ting.
- Det er en forutsetning at dere kommenterer programmene dere leverer som besvarelse på de obligatoriske oppgavene (oblig 2-4).

Eksempel: arealberegning

```
class Areal {
   public static void main (String [] args) {

      int radius = 4;
      double areal = 3.14 * radius * radius;

      System.out.println(areal);

   }
}
```

System.out.println(areal) skriver her ut på skjermen verdien til variabelen areal

KOMPILERING OG KJØRING

```
> javac Areal.java
> java Areal
50.24
```

Gangetabell

```
class Gangetabell {
   public static void main (String [] args) {
      System.out.println(1 * 8);
      System.out.println(2 * 8);
      System.out.println(3 * 8);
      System.out.println(4 * 8);
      System.out.println(5 * 8);
   }
}
```

KOMPILERING OG KJØRING

```
> javac Gangetabell.java
> java Gangetabell
8
16
24
32
40
```


Konvertering

- Når det er nødvendig vil Java automatisk (implisitt) konvertere heltall til desimaltall, som f.eks. i disse tre tilfellene:

```
a) double x = 7;

b) int a = 15;
   double x = a;

c) double x = (7 + 14) * 3 - 12;
```

- Derimot vil Java ikke automatisk konvertere desimaltall til heltall, siden det generelt fører til en endring i verdien:

```
a) int a = 7.15;              // Ikke lov!!

b) double x = 15.6;
   int a = x;                 // Ikke lov!!

c) int a = 3.14 * 7 / 5;      // Ikke lov!!
```


Konvertering *forts.*

- Dersom vi virkelig ønsker å konvertere et desimaltall til et heltall, må vi eksplisitt be om det:

```
a) int a = (int) 7.15;              // Lovlig!

b) double x = 15.6;
   int a = (int) x;                 // Lovlig!

c) int a = (int) 3.14 * 7 / 5;      // Lovlig!
```

- I noen tilfeller - når tallene allikevel er hele - spiller det ingen rolle om man bruker int eller double. Så hvorfor ikke alltid bruke double?

Hvorfor ikke alltid bruke double?

- Mens regning med heltall alltid er eksakt, er regning med desimaltall ikke det - maskinen kan gjøre avrundingsfeil, slik som her:

```
double x = 0.1;
double y = (x + 1) - 1;
// Nå er verdien til x == y false!
```

- Verdiene til x og y er nesten like, men fordi det er en forskjell i et av desimalene langt ute blir x==y false. Slike avrundingsfeil betyr ofte veldig lite, men du kan ikke stole på at alle desimalene er korrekte når du regner med double.
- Det tar mer plass i hukommelsen å holde en double-verdi enn å holde en int-verdi.
- Det kan ta mer tid å gjøre beregninger med desimaltall enn med heltall.
- Konklusjon: når det er naturlig å bruke heltall bruker du int og når det er naturlig å bruke desimaltall bruker du double!

Avrunding

- Konvertering fra desimaltall til heltall involverer normalt en avrunding.

```
class Avrunding {
  public static void main (String [] args) {
    double x = 0.53;

    // Avrund nedover:
    System.out.println((int)Math.floor(x));

    // Avrund oppover:
    System.out.println((int)Math.ceil(x));

    // Avrund til nærmeste heltall:
    System.out.println((int)Math.round(x));
  }
}
```

Test programmet

Heltallsdivisjon

- Java konverterer ikke fra heltall til desimaltall når to heltall adderes, subtraheres, multipliseres eller divideres:
  - 234 + 63 : heltall (int)
  - 235 - 23 : heltall (int)
  - 631 * 367 : heltall (int)
  - 7 / 2 : heltall (int)
- Legg spesielt merke til siste punkt ovenfor:

Når to heltall divideres på hverandre i Java blir resultatet et heltall, selv om vanlige divisjonsregler tilsier noe annet. Dette kalles heltallsdivisjon, og resultatet er det samme som om vi fulgte vanlige divisjonsregler og så avrundet nedover til nærmeste heltall. Dvs (7/2) = = (int) (7.0/2.0) = = 3.

Konkatenering av tekst

Det er ofte nyttig å slå sammen (=konkatenere) flere tekstbiter til en stor tekstbit før vi skriver ut på skjerm. Det kan vi gjøre i Java med + slik som i dette eksemplet:

```
class SkrivPaaSkjerm {
  public static void main (String [] args) {
    double hastighet = 90.5;
    double avstand = 360.2;
    System.out.println("Kjørelengde: " + avstand + " km");
    System.out.println("Hastighet: " + hastighet + " km/t");
    System.out.println("Kjøretid: " + avstand/hastighet + " timer");
  }
}
```

NB: husk at + også brukes til å addere tall. Det er forskjell på

```
System.out.println("2" + "3");      (utskriften blir: 23)
System.out.println("2 + 3");        (utskriften blir: 2 + 3)
System.out.println(2 + 3);          (utskriften blir: 5)
```


Oppgave

- Avgjør i hvert tilfelle hvilken datatype resultatet har:

Uttrykk	Datatype
2 + 6 * 3	int
14.2 + 6	double
3/2 + 4	int
"Vekt: " + 25 + " kg"	String
"" + 17.4	String
(int) 5.3 + 3.25	double

Hva er vitsen med class?

- En setning av typen

```
class {
  <...masse rart...>
}
```

  kalles en klassedeklarasjon (eller bare klasse).

- Tenk på en klasse som en samling data (tall, tekst, bilder, osv) og operasjoner som vi ønsker å kunne utføre på dataene.

- Senere i kurset kommer hvert program til å bestå av mange klasser. Hver klasse har sitt ansvarsområde: å utføre visse oppgaver, håndtere visse typer data, eller begge deler. Dette gjør bl.a. programmene oversiktlige og gjør det lettere å bruke biter av programmet på nytt i andre sammenhenger.