INF1000 : Forelesning 2

Enkle feilsituasjoner
Beregning av matematiske og logiske uttrykk
Terminal I/O
Forgreninger

Ole Christian Lingjærde
Biomedisinsk forskningsgruppe
Institutt for informatikk
Universitetet i Oslo

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.

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 = .435;
double vekt = 1.23e-3;
double omkrets = 2 * pi * radius;
```

Vi kan legge et heltall inn i en double-variabel:

```
double radius = 2;
```

men maskinen behandler det da som et desimaltall: 2.0000.....

Moral: hvis det er viktig at tallet behandles som et heltall, bruker du **int.**

Sannhetsverdier

- Variabeltypen *boolean* har bare to mulige verdier **true** og **false**:

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

int x = 3;
b = (x < 3);       // Nå får b verdien false
```

Fullstendige programeksempler

- Noen gir kompileringsfeil, dvs javac protesterer.
- Noen gir kjørefeil, dvs java protesterer.
- Noen gir ingen feilmeldinger.

Det er **ekstremt nyttig** å lære seg å forstå de vanligste feilmeldingene. Da:

- ✓ Sparer du tid (din og andres)
- ✓ Har kontroll (mindre avhengig av ekstern hjelp)
- ✓ Lærer hurtigere å programmere (mange begynnerfeil skyldes feilaktig forståelse av hva programmet gjør)

Eksempel 1

```
class Eksempel1 {
   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

```
Eksempel1.java:6: possible loss of precision
found   : double
required: int
       y = x;
           ^
1 error
```


Eksempel 2

```
class Eksempel2 {
   public static void main (String [] args) {
      boolean b;
      b = 2;   // Her prøver vi å sette b lik et heltall
   }
}
```

FEIL UNDER KOMPILERING

```
Eksempel2.java:4: incompatible types
found   : int
required: boolean
       b = 2;
           ^
1 error
```


Eksempel 3

```
class Eksempel3 {
   public static void main (String [] args) {
      int x, y;
      y = x;  // Høyresiden inneholder en variabel som ikke er initialisert
   }
}
```

FEIL UNDER KOMPILERING

```
Eksempel3.java:4: variable x might not have been initialized
       y = x;
           ^
1 error
```

Eksempel 4

```
class Eksempel4 {
   public static void main (String [] args) {
      int x = 3;
      int y = 0;
      int z = x / y;     // Heltallsdivisjon med null
   }
}
```

FEIL UNDER KJØRING

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
                at Eksempel4.main(Eksempel4.java:5)
```


Eksempel 5

```
class Eksempel5 {
   public static void main (String [] args) {
      double x = 3.0;
      double y = 0.0;
      double z = x / y;
      System.out.println(z);
   }
}
```

KOMPILERING OG KJØRING

```
> javac Eksempel5.java
> java Eksempel5
Infinity
```


Eksempel 6

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

      double radius = 4.0;
      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
```


Eksempel 7

- Anta at vi har utført disse instruksjonene:

```
int første = 65;
int 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

Konvertering int→double

Java konverterer ved behov automatisk (implisitt) heltall til desimaltall.

Eksempler:

```
double x = 7;
// Nå har x verdien 7.000...

int a = 15;
double x = a;
// Nå har x verdien 15.000...


double x = (7 + 14) * 3 - 12;
// Nå har x verdien 51.0...
```


Konvertering double→int

Java konverterer ikke automatisk andre veien:

```
A:  int a = 7.15;                    // Ikke lov!!

B:  double x = 15.6;
    int a = x;                       // Ikke lov!!

C:  int a = 3.14 * 7 – 5.1;          // Ikke lov!!
```

Vi får det til med eksplisitt konvertering:

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

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

C:  int a = (int) (3.14 * 7 – 5.1); // Lovlig!
```


Regning med double er ikke eksakt

- 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 har x og y forskjellig verdi
```

- Verdiene til x og y er nesten like, men det er en forskjell i et av desimalene langt ute. Slike avrundingsfeil betyr noen ganger ingenting, andre ganger kan de være katastrofale.

Eksempel

Et stjerneeksempel på hvor galt det kan gå:

Den europeiske forskningsraketten Ariane 5 ble sendt opp i 1996 fra Fransk Guiana. Den falt ned etter 37 sekunder pga en feil i en konvertering fra desimaltall til heltall. En av de dyreste regnefeil i historien.

int eller double?

- Det er åpenbart mest fleksibelt å bruke double, men:
  - Vi har avrundingsproblematikken
  - 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 (men veldig maskinavhengig).
- **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));
  }
}
```


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
2 + 3 = = 2 + 5	boolean

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, kan vi legge inn kommentarer i programteksten. Disse ignoreres av kompilatoren.
- 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.
- I obligatorisk oppgave 2, 3, og 4 må programmene du leverer være kommentert for å bli godkjent.

Når du løser oppgaver

1. **Bestem programmets oppførsel sett utenfra:**
   - Hva skal være inndata (input) til programmet?
   - Hvordan skal programmet få tak i inndataene?
   - Hva skal være utdata (output) fra programmet?
   - Hvordan skal utdataene presenteres for brukeren?
2. **Avgjør hvordan du skal transformere inndata til utdata:**
   - Hvordan skal inn- og utdata representeres (lagres) i programmet?
   - Reduser transformasjonen inndata -> utdata til en sekvens av trinn hvor hvert trinn gjør en enkel ting med dataene og hvor hvert trinn er enkelt å programmere.
3. **Skriv programkode (og test løsningen).**

Eksempel: Celcius og Fahrenheit

- Problem:

  I Norge angis vanligvis temperaturer i Celcius (C), mens man bl.a. i USA benytter Fahrenheit (F). F.eks. svarer 0 C til 32 F.

  Lag et program som lager en tabell som nedenfor (og med temperaturer i Fahrenheit fylt inn):

Celcius	Fahrenheit
-10.0	.....
0.0	.....
37.0	.....
100.0	.....

Hvilke data beskriver problemet?

- Inndata:
  - De fire Celcius-temperaturene -10, 0, 37 og 100 (desimaltall)
  - Vi tenker oss at temperaturene er gitt når vi skriver programmet. Senere skal vi se hvordan programmet kunne ha lest inndata fra terminal (fra brukeren).
- Utdata:
  - De tilsvarende (konverterte) Fahrenheit-temperaturene (desimaltall)
  - Skal skrives ut på skjermen i en tabell

Transformere inndata til utdata

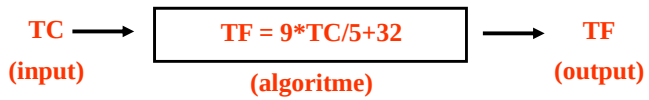
- Vi må kjenne formelen for å regne om fra Celcius til Fahrenheit. La

  TC = Temperatur i Celcius
  TF = Temperatur i Fahrenheit

  Vi finner i et oppslagsverk at omregningsformelen er

  TF = 9 * TC / 5 + 32

  Dermed blir fremgangsmåten slik:



Programskisse

```
class TemperaturKonvertering {
  public static void main (String[] args) {
    <deklarasjoner>

    <Skriv overskrift>

    <sett TC lik -10>
    <regn ut TF>
    <skriv ut>

    <sett TC lik 0>
    <regn ut TF>
    <skriv ut>

    <sett TC lik 37>
    <regn ut TF>
    <skriv ut>

    <sett TC lik 100>
    <regn ut TF>
    <skriv ut>
  }
}
```


Ferdig program

```
class TemperaturKonvertering {
  public static void main (String[] args) {
    double TC, TF;

    System.out.println("Celcius     Fahrenheit");

    TC = -10;
    TF = 9 * TC / 5 + 32;
    System.out.println(TC + "       " + TF);

    TC = 0;
    TF = 9 * TC / 5 + 32;
    System.out.println(TC + "       " + TF);

    TC = 37;
    TF = 9 * TC / 5 + 32;
    System.out.println(TC + "       " + TF);

    TC = 100;
    TF = 9 * TC / 5 + 32;
    System.out.println(TC + "       " + TF);

  }
}
```

Test programmet

Innlesning fra terminal

- Innlesning fra terminal kan gjøres på flere måter i Java. I INF1000 bruker vi pakken easyIO. Du må da skrive i toppen av programmet:

```
import easyIO.*;
```

- Inne i klassen skriver vi følgende før vi kan starte innlesning:

```
In tastatur = new In();
```

- Så kan vi lese inn fra terminal (=tastatur), f.eks. et heltall:

```
int radius;
System.out.print("Oppgi radiusen: ");
radius = tastatur.inInt();
```


Eksempel

Vi må først importere pakken easyIO

Vi oppretter en verktøykasse for lesing fra terminal og lager en variabel tast som blir vårt håndtak til denne verktøykassen

```
import easyIO.*;

class LesFraTerminal {
   public static void main (String [] args) {
      In tast = new In();

      System.out.print("Skriv et heltall: ");
      int k = tast.inInt();
      System.out.println("Du skrev: " + k);
   }
}
```

I verktøykassen ligger det bl.a. en metode for å lese et heltall fra terminal.

Lesemetoder

```
// Opprette forbindelse med tastatur:
In tastatur = new In();

// Lese et heltall:
int k = tastatur.inInt();

// Lese et desimaltall:
double x = tastatur.inDouble();

// Lese et enkelt tegn:
char c = tastatur.inChar();

// Lese et enkelt ord:
String s = tastatur.inWord();

// Lese resten av linjen:
String s = tastatur.inLine();
```


Hvilken lesemetode skal jeg velge?

- Først: importere easyIO og åpne forbindelse til tastaturet
- Lese item for item:
  - For å lese et heltall: inInt() [egentlig: tastatur.inInt()]
  - For å lese et desimaltall: inDouble()
  - For å lese ett ord: inWord()
  - For å lese en hel linje (med minst ett tegn): inLine()
- Lese linje for linje:
  - Bruk readLine()
- Lese tegn for tegn:
  - For å lese neste tegn (også hvite tegn): inChar()

Hvordan lesemetodene virker

- Metodene inInt(), inDouble() og inWord() virker slik:
  - De hopper over eventuelle innledende blanke tegn.
  - De leser så alt fram til neste blanke tegn eller linjeskift. Dersom det som leses ikke er et heltall når inInt() brukes eller et desimaltall når inDouble() brukes, gis det en feilmelding og man får en ny sjanse.
- Metoden inChar() virker slik:
  - Den leser neste tegn, enten det er et blankt tegn eller ikke.
- Metoden inLine() virker slik:
  - Den leser alt fram til slutten av linjen (inkludert blanke tegn), men ignorerer linjer hvor det kun står (igjen) et linjeskift.

Hvordan lesemetodene virker

Terminal-input: `_ _ x y z _ 1 6 1 2 7 5` `_ = blank`

`String s1 = tast.inWord();` `String s2 = tast.inWord();`	`s1: "xyz"` `s2: "161275"`
`String s1 = tast.inWord();` `int x = tast.inInt();`	`s1: "xyz"` `x :161275`
`String s = tast.inLine();`	`s: "  xyz 161275"`

`char c1 = tast.inChar();`
`char c2 = tast.inChar();`
`char c3 = tast.inChar();`
	`c1: ' '`
	`c2: ' '`
	`c3: 'x'`

`int x = tast.inInt();`	`feilmelding`

Eksempel: lese data om en person

- Problem:

  Lag et program som leser fra terminal disse dataene om en person:
  - Navn
  - Yrke
  - Alder

  og som skriver ut dataene på skjermen etterpå.

- Framgangsmåte:
  - Vi bruker inLine() til å lese navn og yrke, og inInt() til å lese alder.

Ferdig program

```
import easyIO.*;

class LesDataOmPerson {
  public static void main (String [] args) {
    String navn, yrke;
    int alder;
    In tast = new In();

    System.out.print("Navn: ");
    navn = tast.inLine();

    System.out.print("Yrke: ");
    yrke = tast.inLine();

    System.out.print("Alder: ");
    alder = tast.inInt();

    System.out.print("Hei " + navn + ", du er " + alder);
    System.out.println(" år gammel og jobber som " + yrke);
  }
}
```

Test programmet

Et eksempel til

```
import easyIO.*;

class LesFraTerminal2 {
  public static void main (String [] args) {
    In tastatur = new In();
    System.out.print("Skriv tre ord: ");
    String s1 = tastatur.inWord();
    String s2 = tastatur.inWord();
    String s3 = tastatur.inWord();
    System.out.println("Du skrev disse ordene:");
    System.out.println("  " + s1);
    System.out.println("  " + s2);
    System.out.println("  " + s3);
  }
}
```

Test programmet

Formatert utskrift til skjerm

- Formatert utskrift vil si at vi angir nøyaktig hvordan utskriften skal se ut og plasseres på skjermen.
- Kan gjøres "manuelt" med System.out.print(...), men det er upraktisk.
- Bedre: bruke en ferdiglaget pakke for slikt. I INF1000 bruker vi pakken easyIO. For å få tilgang til denne pakken må vi som før skrive helt i toppen av programmet vårt (før class):

  ```
  import easyIO.*;
  ```

- Inne i klassen skriver vi følgende før vi kan starte formatert utskrift:

  ```
  Out skjerm = new Out();
  ```

- Så kan vi skrive ut det vi ønsker, f.eks.:

  ```
  double pi = 3.1415926;
  skjerm.out(pi, 2, 6);   // Skriv ut pi med 2 desimaler,
                             høyrejustert på 6 plasser.
  ```

Eksempel

Vi må først importere pakken easyIO

```
import easyIO.*;

class FormatertUtskrift {
   public static void main (String [] args) {
      Out skjerm = new Out();
      int BREDDE1 = 15;
      int BREDDE2 = 30;
      skjerm.out("NAVN", BREDDE1);
      skjerm.outln("ADRESSE", BREDDE2);
      skjerm.out("Kristin Olsen", BREDDE1);
      skjerm.outln("Vassfaret 14, 0773 Oslo", BREDDE2);
   }
}
```

Vi oppretter en verktøykasse for skriving til terminal

Test programmet

I verktøykassen ligger det bl.a. verktøy (på java-språk: *metoder*) for å skrive til skjerm med og uten linjeskift til slutt.

Tre måter å skrive ut på

- Uten formatering:

```
skjerm.out("Per Hansen");
skjerm.out(12345);
skjerm.out(3.1415, 2);
```

- Angi utskriftsbredde:

```
skjerm.out("Per Hansen", 15);      // Bredde 15 tegn
skjerm.out(12345, 15);             // Bredde 15 tegn
skjerm.out(3.1415, 2, 15);         // Bredde 15 tegn
```

- Angi utskriftsbredde og justering:

```
skjerm.out("Per Hansen", 15, Out.RIGHT);   // Høyrejuster
skjerm.out(12345, 15, Out.CENTER);         // Senterjuster
skjerm.out(3.1415, 2, 15, Out.LEFT);       // Venstrejuster
```


Programmer med forgreninger

- En svært nyttig programmeringsteknikk er å bruke forgreninger, dvs forskjellige instruksjoner utføres i ulike situasjoner.
- Vi kan få til dette med en if-setning:

```
if (logisk uttrykk)
{
   <instruksjoner>
}
else
{
   <instruksjoner>
}
```

f.eks. x < y eller et annet uttrykk som enten er true eller false

denne blir utført når det logiske uttrykket er sant (true)

denne blir utført når det logiske uttrykket er usant (false)

- Eksempel:

```
if (x > 0) {
   System.out.println("Tallet er positivt");
} else {
   System.out.println("Tallet er ikke positivt");
}
```


Programmer med forgreninger

- Else-delen kan utelates, slik som her:

```
if (pris > 1500) {
  System.out.println(”Det er for dyrt”);
}
```

- Vi kan legge if-setninger inni if-setninger:

```
if (lønn < 400000) {
  if (ferieuker < 8) {
    System.out.println(”Ikke søk på jobben”);
  }
}
```

- Vi kan lage sammensatte if-setninger av typen

```
if (a < 10) { // a er positivt heltall
  System.out.println(”Ett siffer”);
} else if (a < 100) {
  System.out.println(”To siffer”);
} else {
  System.out.println(”Mer enn to siffer”);
}
```

Eksempel på bruk av if-setning

Program som avgjør hvem av to personer som er høyest:

```
import easyIO.*;

class Hoyde {
  public static void main (String[] args) {
    In tastatur = new In();
    double høyde1, høyde2;

    System.out.print("Høyden til Per: ");
    høyde1 = tastatur.inDouble();
    System.out.print("Høyden til Kari: ");
    høyde2 = tastatur.inDouble();

    if (høyde1 > høyde2) {
      System.out.println("Per er høyere enn Kari");
    } else if (høyde1 < høyde2) {
      System.out.println("Kari er høyere enn Per");
    } else {
      System.out.println("Kari og Per er like høye");
    }
  }
}
```

Test programmet