INF1000 – Uke 3

Innlesing fra terminal, formatert utskrift og forgreininger

Oversikt

- Litt repetisjon
  - Program
  - Variabler og Uttrykk
  - Presedens
  - Matematiske funksjoner
- Innlesing
- Formatert utskrift

Repetisjon – Program

- Program skrives i et programmeringsspråk
- Imperativ programmering: **Setninger** utføres i **sekvens**, ovenfra og nedover
- Variable **deklareres**
- Variable **tilordnes** verdier i setninger som kan inneholde **uttrykk**
- Program skrives i en fil, kompileres og kjøres.

Repetisjon – Program

```
class Repetisjon {
   public static void main(String[] args){
        final double PI = 3.14;  // final betyr konstant
        double radius = 2.0;
        double areal;
        String FORTEKST =
               "Arealet til en sirkel med radius ";

        // Utregning
        areal = PI * radius * radius;

        System.out.println(FORTEKST + radius +
               " ER " + areal + ".");
   }
}
```

Alt inne i klasser

Prosedyren ”main”

En kommentar

Setninger avsluttes med semikolon

UNIVERSITETET I OSLO

Repetisjon – Variable og uttrykk

- En variabel er en plass i maskinens lager (minne)
- Variable må ha **navn**
  - Slik at vi kan angi i programmet vårt hvilken plass i lageret vi snakker om
- Variable må ha **type**
  - Så vi vet hvor stor plass variabelen tar og hva det er som ligger der

UNIVERSITETET I OSLO

Utrykk

- Aritmetiske uttrykk:
  - (2 + sum) / antall; // sum og antall er int-variable
- Logiske uttrykk
  - ((sum < 14) && (verdi == 43));

UNIVERSITETET I OSLO

Presedensregler for logiske uttrykk

- *Presedens* er regler for hvilken rekkefølge utregninger skal skje i et uttrykk
- Vi snakker om høyere og lavere presedens
- Ordning av operatorene:
  - Høyest: Metodekall
  - ! (ikke)
  - <, <=, >, >=
  - ==, !=
  - && (og)
  - || (eller)

UNIVERSITETET I OSLO

Aritmetisk uttrykk – skjulte parenteser

```
2.0*3  + Math.ceil(7.23)    +  (2 + 3) * 3.5

2.0*3  + (Math.ceil(7.23))  +  (2 + 3) * 3.5

(2.0*3) + (Math.ceil(7.23))  +  (2 + 3) * 3.5

(2.0*3) + (Math.ceil(7.23))  + ((2 + 3) * 3.5)

((2.0*3) + (Math.ceil(7.23))) + ((2 + 3) * 3.5)

(((2.0*3) + (Math.ceil(7.23))) + ((2 + 3) * 3.5))
```

UNIVERSITETET I OSLO

Aritmetisk uttrykk – utregning

```
(((2.0*3) + (Math.ceil(7.23))) + ((2 + 3) * 3.5))
 ((6.0    + (Math.ceil(7.23))) + ((2 + 3) * 3.5))

 ((6.0    +  8.0)              + ((2 + 3) * 3.5))

  (14.0                        + ((2 + 3) * 3.5))

  (14.0                        + ( 5      * 3.5))

  (14.0                        +   17.5)

   31.5
```


UNIVERSITETET I OSLO

Eksempel – Logisk uttrykk

```
int u=1,v=1;
boolean b = true;

boolean resultat =
    u>2 || v<2 &&  b == !(++u == v++);

// Hva blir resultat?
```


UNIVERSITETET I OSLO

Logisk uttrykk – skjulte parenteser

```
 u>2  ||   v<2  &&  b ==  !( ++u  ==  v++)
 u>2  ||   v<2  &&  b ==  !((++u) == (v++))
 u>2  ||   v<2  &&  b == (!((++u) == (v++)))
(u>2) ||   v<2  &&  b == (!((++u) == (v++)))
(u>2) ||  (v<2) &&  b == (!((++u) == (v++)))
(u>2) ||  (v<2) && (b == (!((++u) == (v++))))
(u>2) || ((v<2) && (b == (!((++u) == (v++)))))
((u>2) || ((v<2) && (b == (!((++u) == (v++))))))
```


UNIVERSITETET I OSLO

Logisk uttrykk – utregning

```
((u>2)  || ((v<2) && (b == (!((++u) == (v++))))))
( false || ((v<2) && (b == (!((++u) == (v++))))))
( false || (true  && (b == (!((++u) == (v++)))))) u: 1, v: 1
( false || (true  && (b == (!(    2 == (v++)))))) u: 2, v: 1
( false || (true  && (b == (!(    2 == 1 ))))) u: 2, v: 2
( false || (true  && (b == (! false ))))
( false || (true  && (b == true )))
( false || (true  && true ))
( false || true)
  true
```

UNIVERSITETET I OSLO

Eksempel – String

```
int x = 2, y = 3;

String tekst = "2+3==" + (x + y) + "!=" + x + y;

System.out.println("tekst: " + tekst);

// Hva blir skrevet ut?
```


UNIVERSITETET I OSLO

String – skjulte parenteser

```
      "2+3==" + (x + y)  + "!="  + x + y

(     "2+3==" + (x + y)) + "!="  + x + y

((    "2+3==" + (x + y)) + "!=") + x  + y

(((  "2+3==" + (x + y)) + "!=”) + x) + y

(((("2+3==" + (x + y)) + "!=”) + x) + y)
```


UNIVERSITETET I OSLO

String – utregning av verdi

```
((((("2+3==" + (x + y)) + "!=”) + x) + y)

((((("2+3==" +  5     ) + "!=”) + x) + y)

(((  "2+3==5”           + "!=”) + x) + y)

((   "2+3==5!=”                 + x) + y)

(    "2+3==5!=2”                   + y)

     "2+3==5!=23”
```


UNIVERSITETET I OSLO

Presedens – Oppsummert

- Bruk parenteser for mye heller enn for lite
- Helst ikke bruk ++ og -- inne i lengre uttrykk
- Se på eksempler og oppgaver og øv på bruk av uttrykk

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 vi int og når det er naturlig å bruke desimaltall bruker vi double

Matematiske funksjoner - 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));
  }
}
```


Resultat

```
$ javac Avrunding.java
$ java Avrunding
0
1
1
$
```


Kompileringsfeil og kjøretidsfeil

- Kompileringsfeil
  - Feil som oppdages av **javac**
  - Feilformulerte setninger
  - Feil type
  - Programmet blir ikke kompilert
  - Husk: Tidligere kompilerte utgaver kan ligge der
- Kjøretidsfeil
  - Feil som oppdages av **java**
  - Feil vi ikke kunne vite om før programmet ble kompilert
  - Programmet ”krasjer”
- Designfeil
  - Bruk av feil formel eller fremgangsmåte. Resultatet blir feil.

Kompileringsfeil

```
class FeilType {
    public static void main(String[ ] args){
        int i = 123456;
        boolean b;
        b = i;
```

```
$ javac FeilType.java
FeilType.java:5: incompatible types
found   : int
required: boolean
        b = i;
            ^
1 error
```


Kjøretidsfeil

```
class DivNull {
    public static void main(String[ ] args){
        int x = 7;
        int y = 0;
        int z = x / y;
    }
}
```

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


Hvordan løse oppgaver

1. **Se oppgaven utenfra:**
   1. Hva skal være inndata (input) til programmet?
   2. Hvordan skal programmet få tak i inndataene?
   3. Hva skal være utdata (output) fra programmet?
   4. Hvordan skal utdataene presenteres for brukeren?
2. **Hvordan transformere inndata til utdata?**
   1. Hvordan skal representeres (lagres)?
   2. Spesifiser en sekvens av trinn der:
      - hvert trinn gjør en enkel ting med dataene
      - hvert trinn er enkelt å programmere
3. **Skriv programkode (og test løsningen)**

Eksempel: Celsius og Fahrenheit

- Problem:
  I Norge angis vanligvis temperaturer i Celsius (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	.....

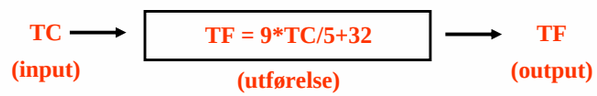
Hvilke data beskriver problemet?

- 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 – ”Pseudokode”

```
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);
   }
}
```

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 importerer pakken easyIO.

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

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

```
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);
  }
}
```


Resultat

```
$ javac LesFraTerminal.java
$ java LesFraTerminal
Skriv et heltall: 123
Du skrev: 123

$
```


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 (3 sjanser).
- 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);
   }
}
```


Et eksempel til

```
import easyIO.*;

class LesInnOrd {
   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);
   }
}
```


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. I toppen av programmet (**før** class) skriv:

```
import easyIO.*;
```

- Inne i klassen skriver vi så:

```
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 = 20;
        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

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

Dukket objekter opp her?

Resultat

```
$ javac FormatertUtskrift.java
$ java FormatertUtskrift
NAVN                ADRESSE
Kristin Olsen       Vassfaret 14, 0773 Oslo
```


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,
                              // 2 desimaler
```

- 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
```


Blokker

- En **programblokk** er en samling med programsetninger omsluttet av krøllparenteser
- Setningene i main-metoden ligger inne i en blokk
- Blokker kan **nøstes** inne i hverandre, slik at vi kan ha blokker inne i blokker
- En variabel som er deklarert inne i en blokk er kun definert (”synlig”) fra stedet den er deklarert til slutten av blokken. Vi kaller det **skopet** til variabelen.

Skop – Lovlig eksempel

```
class SkopLovlig {
    public static void main(String args[]){
        int k = 15;
        {
            int n = 10;
            System.out.println(k + n);
        }
        // Her er ikke n definert
        System.out.println(k);
    }
}
```

Skop – Ikke lovlig eksempel

```
class SkopIkkeLovlig {
    public static void main(String args[]){
        int k = 15;
        {
            int n = 10;
            int k = 200; // Ikke lov.
                         // k er allerede
                         // definert.
        }
  }
}
```


Skop – Ikke lovlig eksempel

```
$ javac SkopIkkeLovlig.java
SkopIkkeLovlig.java:6: k is already defined in
  main(java.lang.String[])
            int k = 200; // Ikke lov.
                ^
1 error
$
```


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** (pseudokode):

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

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

Den første blokken (og bare den) blir utført hvis det logiske uttrykket er sant (true)

Den andre blokken (og bare den) blir utført hvis det logiske uttrykket er usant (false)

- Eksempel:

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


Varianter av if-setninger

- Else-delen kan utelates:

```
if (pris > 1500) {System.out.println(”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:

```
if (a < 10) { // a ikke 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 {
            System.out.println("Per er ikke høyere enn Kari");
        }
    }
}
```


Eksempel: Body Mass Index

Oppgave:

Body Mass Index (BMI) er et mål som kan regnes ut fra høyden og vekten til en person. Ifølge verdens helseorganisasjon (WHO)[1] :

BMI	Vektstatus
Under 18.5	Undervekt
18.5 – 24.9	Normal vekt
25.0 – 29.9	Overvekt
30.0 eller høyere	Fedme

Vi skal lage et program som beregner BMI ut fra høyde og vekt og gir melding om hvilken vektstatus (se tabellen) det tilsvarer.

[1]Se http://www.who.int/hpr/NPH/docs/gs_obesity.pdf

Inndata og utdata

- **Inndata:**
  - Personens **høyde** (i m)
  - Personens **vekt** (i kg)
  - Leses fra terminal
- **Utdata:**
  - **BMI**
  - Skrives ut på skjerm, sammen med en av beskjedene
    - ***Undervekt*** (hvis BMI <= 18.4)
    - ***Normal vekt*** (hvis 18.5 <= BMI <= 24.9)
    - ***Overvekt*** (hvis 25.0 <= BMI <= 29.9)
    - ***Fedme*** (hvis BMI >= 30.0)

Transformere inndata til utdata

- Vi må kjenne formelen for å regne ut BMI. La

  vekt = personens vekt i kg

  hoyde = personens høyde i m

- Da er

  BMI = vekt / (hoyde*hoyde)

Ferdig program

```
import easyIO.*;
class BodyMassIndex {
    public static void main (String[] args) {
        In tast = new In();
        System.out.print("Vekt (i kg): ");
        double vekt = tast.inDouble();
        System.out.print("Høyde (i cm): ");
        double høyde = tast.inDouble()/100;
        double bmi = vekt / (høyde * høyde);
        System.out.println("BMI = " + bmi);

        if (bmi <= 18.4) {
            System.out.println("Vektstatus: undervekt");
        } else if (bmi <= 24.9) {
            System.out.println("Vektstatus: normalvekt");
        } else if (bmi <= 29.9) {
            System.out.println("Vektstatus: overvekt");
        } else {
            System.out.println("Vektstatus: fedme");
        }
    }
}
```


Alternativ til if-else: switch

- En sammensetning av flere if-setninger kan i noen tilfeller erstattes med en switch-setning:

```
switch (uttrykk) {
    case verdi1:
        <instruksjoner>
        break;
    ....

    case verdiN:
        <instruksjoner>
        break;
    default:
        <instruksjoner>
}
```

- Nøkkelordet **break** avbryter utførelsen av switch-setningen. Når **break** mangler, fortsetter utførelsen på neste linje (det er sjelden ønskelig).

Eksempel

```
class BrukAvSwitch {
   public static void main (String [] args) {
        char c = 'x';
        switch(c) {
        case 'a':
               System.out.println("Tegnet var en a");
               break;
        case 'b':
               System.out.println("Tegnet var en b");
               break;
        default :
               System.out.println(
                      "Tegnet var ikke a eller b");
        }
   }
}
```