Repetisjon
INF 1000 – våren 2007

Grunnkurs i programmering

Institutt for Informatikk

Universitet i Oslo

Anne Landro, Are Magnus Bruaset og Arild Waaler

Mål for INF1000

- Gi grunnleggende forståelse av noen sentrale begreper, problemstillinger og metoder innen informatikk
- **Lære å programmere**
- Gi noe innsikt i datamaskiners muligheter og begrensninger
- Lære noe om samfunnsmessige konsekvenser av bruk av informasjonsteknologi

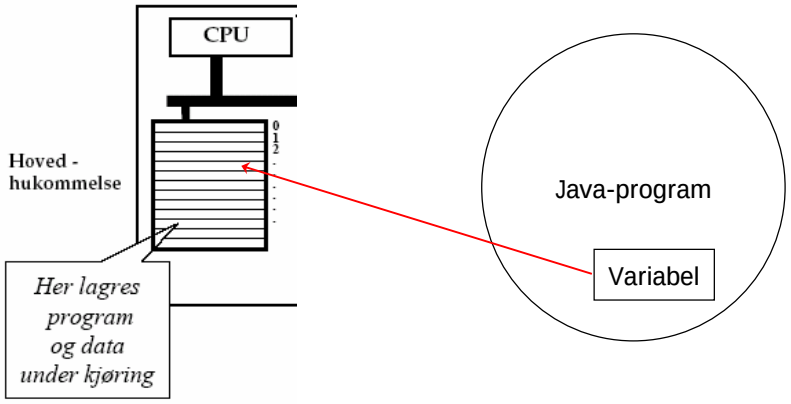
Programvareutvikling - oversikt

1. Først har vi et problem vi skal løse (en oppgave)
2. Finn en fremgangsmåte (=algoritme) for problemet
3. Hvilke data beskriver problemet / algoritmen?
4. Skriv et (Java-)program, syntaktisk korrekt slik at kompilatoren ikke gir feilmeldinger
5. Test ut programmet, sjekk at det gir riktig svar

Representasjon av data

- Variabler - tre egenskaper:
  - Et **navn** (ulikt alle andre variablers)
  - En **type** (tekst, heltall, desimaltall,...)
  - Et **data-innhold,** en **verdi**

Minnelokasjon



Heltall og desimaltall (flyttall)

- Datamaskiner håndterer disse på forskjellig måte
- Heltall er alltid eksakte, mens desimaltall har bare en viss nøyaktighet

Handlingssetninger, tilordning

- Tilordningsetninger (v.s. = h.s;) gir en ny verdi til en variabel
- På venstre side (v.s.) står navnet til en variabel
- Tegnet **=** leses: *"settes lik"*
- På høyre side (h.s.) står et regnestykke, verdien regnes ut

Addisjon (+) har mange virkemåter

- Mellom to heltall gir "+" en heltallsaddisjon, **`2 + 2`**
- Mellom to flytende tall gir "+" en desimaltallsaddisjon, **`2.5 + 3.14`**
- Mellom et heltall og et flytende tall gir "+" en desimaltalls-addisjon: **`2 + 3.14`**
- Mellom to tekststrenger gir "+" en sammenskjøting (konkatenering) av strengene **`"Hallo " + "verden"`**

Konvertering

- Hvis nødvendig vil Java automatisk (implisitt) konvertere heltall til desimaltall
- Eksempler:
  - `double x = 7;`
  - ```
    int a = 15;
    double x = a;
    ```
  - `double x = (7 + 14) * 3 - 12;`

Mer om konvertering

- Java vil **ikke** automatisk konvertere desimaltall til heltall, siden det generelt fører til en endring i verdien:
  - `int a = 7.15;  // Ikke lov!!`
  - ```
    double x = 15.6;
    int a = x;  // Ikke lov!!
    ```
  - `int a = 3.14 * 7 / 5;  // Ikke lov!!`

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)

Heltallsdivisjon

- Legg spesielt merke heltallsdivisjonen:

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.

Tekster og klassen `String`

- En tekststreng er en sekvens av tegn (null, en eller flere), f.eks.

  `""`

  `"*"`

  `"Kristina"`

- Hver tekststreng vi lager er et *objekt* av typen **`String`**

Konkatenering

- Operatoren + har flere betydninger i Java:
  - mellom to tall: addisjon
  - mellom to tekster : tekstkonkatenering
  - mellom tekst og annen type : tekstkonkatenering
- Eksempel på overlasting av metode

Eksempel

Husk at uttrykk i Java beregnes fra venstre mot høyre:

```
class Konkatenering {
  public static void main (String[] args) {
    System.out.println("Sum: " + 2 + 3);
    System.out.println(1 + 2 + 3 + " " + 1 + 2 + 3);
  }
}
```

```
>java Konkatenering
Sum: 23
6 123
```


Deler av en tekststreng

- Generelt:

```
s.substring(index1, index2)
```

Første posisjon som skal være med

Første posisjon som *ikke* skal være med

- Siste del av en tekststreng:

```
String s = "Paris er hovedstaden i Frankrike";
String s1 = s.substring(6);
// Nå er s1 tekststrengen "er hovedstaden i Frankrike"
```

Fra tall til tekst og omvendt

- For å konvertere fra tall til tekst:

```
String s1 = String.valueOf(3.14);
String s2 = String.valueOf('a');
String s3 = String.valueOf(false);

String s4 = "" + 3.14
String s5 = "" + 'a';
String s6 = "" + false;
```

- For å konvertere fra tekst til tall:

```
int k = Integer.parseInt(s);
double x = Double.parseDouble(s);
```
(og tilsvarende for de andre numeriske datatypene)

Enkel if-setning

```
if (logisk uttrykk){
  /* Her kommer de instruksjonene
     som skal utføres når det logiske
     uttrykket er sant (true) */
}
```


If-setning med else-gren

```
if (logisk uttrykk){
    /* Her kommer de instruksjonene
       som skal utføres når det
       logiske uttrykket er sant (true) */
} else {
    /* Her kommer de instruksjonene
       som skal utføres når det
       logiske uttrykket er usant (false) */
}
```


If-setninger kan settes sammen

```
if (x < 0) {
  System.out.print("Tallet er negativt");
} else if (x == 0) {
  System.out.print("Tallet er null");
} else {
  System.out.print("Tallet er positivt");
}
```

Eksempel: Adgang til studier

- Problem:

  For å få adgang til universitetet i Ruritania må man oppfylle begge disse kravene:

  alder ≥ 18

  karaktersnitt > 4.8

  Lag et program som, gitt en persons alder og karaktersnitt, skriver ut på skjerm om personen får adgang til universitetet eller ikke.


```
import easyIO.*;

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

    int    alder;
    double karSnitt;

    Out skjerm = new Out();
    In  tast   = new In ();

    skjerm.out("Alder: ");          alder    = tast.inInt();
    skjerm.out("Karaktersnitt: ");  karSnitt = tast.inDouble();

    if (alder >= 18) {
      if (karSnitt > 4.8)
        skjerm.outln("Personen kan tas opp");
      else
        skjerm.outln("Personen kan ikke tas opp");
    } else
      skjerm.outln("Personen kan ikke tas opp");
  }
}
```

While-løkker

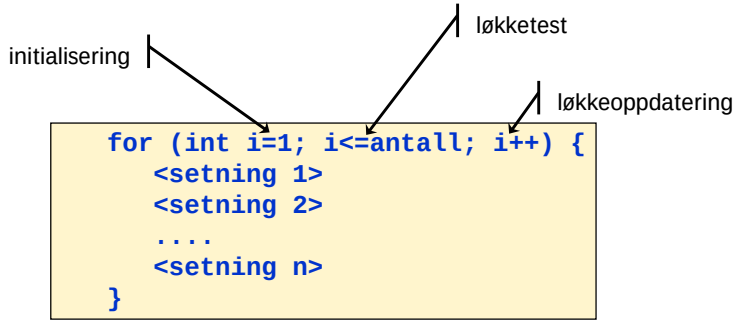
```
while (logisk uttrykk) {
  setning 1;
  setning 2;
  .....
  setning n;
}
```


Ferdig program

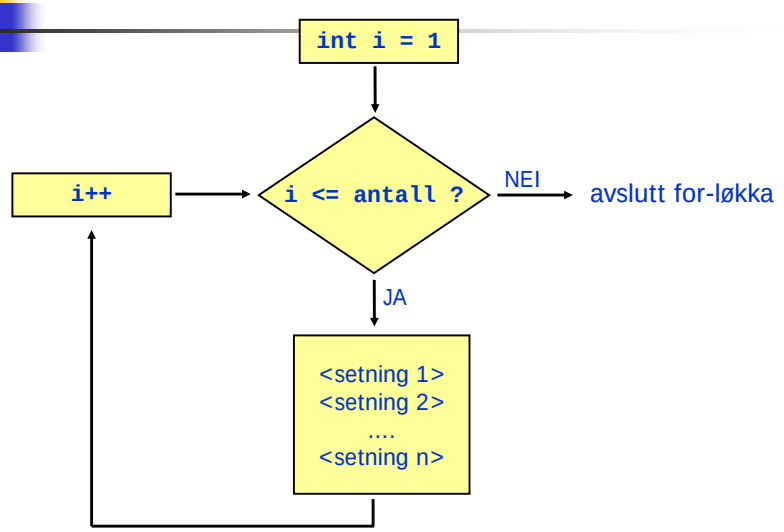
```
class TreGangen {
  public static void main (String[] args) {
    int k=1, svar;

    while (k <= 10) {
      svar = k * 3;
      System.out.println(k + " * 3 = " + svar);
      k = k + 1;
    }

  }
}
```

For-setninger

- En annen måte å få utført en instruksjon (eller blokk) mange ganger er ved hjelp av en for-setning (også kalt for-løkke):



Hvordan for-setningen virker



Variabler

- Hittil har vi sett på variable som kan holde en enkelt verdi:
  - en **`int`**-variabel har plass til ett heltall
  - en **`double`**-variabel har plass til ett desimaltall
  - en **`String`**-variabel har plass til en enkelt tekststreng
  - osv.

Arrayer

- Arrayer er "variable" som kan holde på mange verdier:
  - en **`int`**-array har plass til mange heltall
  - en **`double`**-array har plass til mange desimaltall
  - en **`String`**-array har plass til mange tekststrenger
  - osv.

Arrayer

- Verdiene som ligger i en array har hver sin posisjon (= indeks):
  - 0, 1, 2, ..., K-1 hvor K = lengden til arrayen
- En array med lengde 4 kan visualiseres slik:

Deklarere og opprette arrayer

- Deklarasjon:

```
datatype[] arrayNavn;
```

f.eks. int, double, boolean eller String

- Oppretting:

```
arrayNavn = new datatype[K];  // K er ønsket lengde
arrayNavn = new datatype[]{verdi1, verdi2, …, verdiK};
```

- Kombinert:

```
datatype[] arrayNavn = new datatype[K];
datatype[] arrayNavn = {verdi1, …, verdiK};
```


Verdiene i en array

- Anta at vi har deklarert og opprettet følgende array:

```
int[] tlf = new int[600];
```

- For å få tak i de enkelte verdiene i arrayen:

```
tlf[0], tlf[1], tlf[2], ..., tlf[599]
```

- For å få tak i lengden på arrayen:

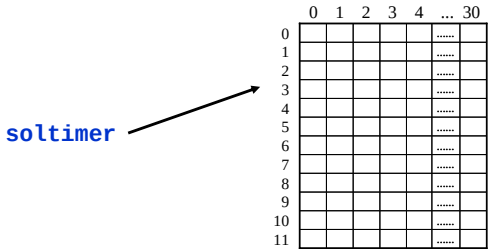
```
tlf.length   // NB: ingen parenteser til slutt
```


Flerdimensjonale arrayer

- Vi kan også deklarere todimensjonale (og høyeredimensjonale) arrayer.
- Eksempel:

```
int[][] soltimer = new int[12][31];
```

- Resultat:



Blokker og metoder

- En blokk er en samling instruksjoner omgitt av krøllparenteser:

```
{
 instruksjon 1;
 instruksjon 2;
 ....
 instruksjon n;
}
```

- Alle steder i et Java-program hvor det kan stå en instruksjon, kan vi om ønskelig i stedet sette inn en blokk

Metoder

- En metode er essensielt en navngitt blokk med instruksjoner som vi kan få utført hvor som helst i et program ved å angi metodens navn
- Beskrivelsen av hva metoden skal hete og hvilke instruksjoner som skal ligge i metoden kalles en metode-deklarasjon.

Metoder

3 typer variable: Klassevariable

- Variable som er deklarert på klassenivå, utenfor metoden
- (Også objektvariable)

3 typer variable: Lokale variable

- Variable som deklareres inne i en metode
- Slike variable er definert fra og med der deklarasjonen gjøres og til slutten av blokken de er deklarert i

3 typer variable: Parametre

- Variable som deklareres i hodet på metoden
- Slike variable er definert i hele metodekroppen

Viktig detalj

Ved gjentatte kall på en metode er det et

nytt sett med lokale variable og parametre

som lages hver gang

Metode uten parametre og returverdi

Følgende metode skriver ut en ordremeny på skjermen:

```
static void skrivMeny () {
   System.out.println("Lovlige kommandoer: ");
   System.out.println("--------------------");
   System.out.println("1  Registrer ny student");
   System.out.println("2  Søk etter student");
   System.out.println("3  Lag liste");
   System.out.println("4  Avslutt");
   System.out.println("--------------------");
}
```

Merk: vi kan hvor som helst i metoden gi instruksjonen

return;

som avslutter utførelsen av metoden og returnerer eksekveringen til kallstedet

Metode med returverdi

Følgende metode leser et positivt tall fra terminal og returner det til kallstedet:

```
static double lesPositivtTall () {
  In tastatur = new In();
  double x;
  do {
    System.out.print("Gi et positivt tall: ");
     x = tastatur.inDouble();
  } while (x <= 0);

  return x;
}
```

Merk: vi kan hvor som helst i metoden gi instruksjonen

return <uttrykk>;

som avslutter utførelsen av metoden og returnerer til kallstedet med verdien til det angitte uttrykket (verdien må være av typen **double** i dette tilfellet)

Parametre og argumenter

```
class Eksempel {

  public static void main (String[] args) {
    minMetode(3.14, 365);
  }

  static void minMetode (double x, int y) {
    .....
  }

}
```

Argumenter

Parametre

Merk: et annet navn for argumenter er *aktuelle parametre*, og et annet navn for parametre er *formelle parametre*

Metode med parameter og returverdi

Følgende metode finner summen av elementene i en double-array:

```
static double finnSum (double[] x) {
   double sum = 0.0;
   for (int i=0; i<x.length; i++) {
      sum += x[i];
   }
   return sum;
}
```


Overlasting av metoder

```
static int sum (int x, int y) {
   return x + y;
}


static double sum (double x, double y) {
   return x + y;
}
```

Objektorientering - oppsummering

- Klasser og objekter
  - Hva er klasser og objekter?
  - Pekere og tilgang til objekter
  - Klasse- og objektvariable
  - Konstruktører
  - Beskyttelse
  - Klassene ArrayList og HashMap
- Filer
- UML
- Det aller viktigste

Klasser og objekter

- **Klasser**
  - En klasse er en mal/mønster for objekter. Det finnes alltid nøyaktig ett eksemplar av hver klasse når programmet kjøres.
  - **Eksempel:** `class Kjøretøy {. . . }`
- **Objekter**
  - Et objekt er en instans laget utifra en klasse. Når programmer startes, finnes ingen objekter (med unntak av systemobjekter). Vi kan lage så mange objekter vi vil.
  - **Eksempel** `Kjøretøy minBil = new Kjøretøy()`
  - Objekter som ingen bryr seg om (dvs peker på), blir fjernet automatisk.

Pekere til objekter

- For å få tilgang til objektene må vi ha pekere:

```
class Stasjon {
  String navn;
  int moh; // meter over havet
}

Stasjon maj, nat;

maj = new Stasjon();
nat = new Stasjon();
```


Tilgang til innmaten i objekter

- Når vi har en peker til et objekt, kan vi få tilgang til innmaten (dvs data og metoder) med et ".":

```
nat.navn = "Nationaltheateret";
nat.moh = 4;
```

Objekt- og klassevariable

- Variable i en klassedeklarasjon vil forekomme i alle objekter. De er **objektvariable**.
- Unntak: Om det står static foran, vil de kun finnes i ett eksemplar, nemlig i *klassen*. De er **klassevariable**.

```
class Scenic {
  static int antLaget;
  String eier;
}
```

- **Spesielle "variable"** (finnes i alle klasser):
  - **this** betegner objektet selv.
  - **null** betegner "ingenting".

Konstruktører

- Med konstruktører kan man lage objekter som initieres med parametre:

```
class Dato {
  int dag, måned, år;
  Dato (int d, m, å) {
    dag = d; måned = m; år = å;
  }
}
Dato idag = new Dato(24, 5, 2004);
```

- Uten en slik konstruktør vil Java sette inn en tom en:

```
Dato ( ) { }
```


Beskyttelse

- Alt i en klasse (og objekt) kan beskyttes:
  - **private** Elementer er usynlige utenfra.
  - **protected** Elementer er kun synlig i **subklassser**. (Omtales ikke i dette kurset.)
  - **public** Elementer er synlig overalt.
  - ***ingenting*** Elementer er synlig i samme **pakke**. (Omtales ikke i dette kurset.)

Klassen ArrayList

- Klassen ArrayList fungerer *nesten* som vanlige array-er.
  - Vi må hente inn klassen fra biblioteket:

```
import java.util.ArrayList;
import java.util.*;
```

  - Objekter opprettes med **new**:

```
ArrayList a = new ArrayList();
```

Klassen ArrayList

- Vi legger inn objekter med metoden **add**:

```
a.add(p);
a.add(n, p);
```

- **NB!** Vi får bare legge inn i posisjoner som er i bruk, eller én høyere!
- Vi henter elementer med metoden **get**:

```
p = (MyClass)a.get(n);
```

- **NB!** Vi må angi objektets klasse.
- Antall elementer får man med metoden **size**:

```
len = a.size();
```


Klassen HashMap – java 1.4

- Klassen importeres med

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

- Objekter opprettes med

```
HashMap map = new HashMap();
```

- Elementer legges inn med

```
map.put(nøkkel, objekt);
```

- En nøkkel kan være et vilkårlig object; i INF1000 vil det alltid være en String.
- Vi henter ut objekter med

```
obj = (MyClass)map.get(nøkkel);
```

- **NB!** Vi må angi objektets klasse.

Klassen HashMap – java 1.5

- **Importere HashMap:**

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

- **Opprette HashMap:**

```
HashMap <Nøkkeltype, Objekttype> map = new HashMap <Nøkkeltype, Objekttype>();
```

  - **Angir hvilken type nøklene og objektene tilhører**
- **Legge inn i HashMap:**

```
map.put(nøkkel, objekt);
```

- **Hente ut fra HashMap:**

```
Myclass m = map.get(nøkkel);
```

  - **Trenger ikke angi objektets klasse**

Løkke gjennom elementer

- I en array kan man gå gjennom alle elementene med en for-løkke:

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

- Det sammen kan man med en ArrayList:

```
for (int i = 0; i < a.size(); ++i)
  ... a.get(i) ...
```

- Med en HashMap bruker man klassen *Iterator*:

```
Iterator it = map.keySet().iterator();
while (it.hasNext()) {
  ... (Xxxx)it.next() ...
}
```

Java 1.5

- **Java 1.5 og iterator:**

```
Iterator <String> it = map.keySet().iterator();
while (it.hasNext()) {
  ... it.next() ...
}
```

- **Java 1.5 og forløkker:**

```
for(Myclass m:map.values()){
      … m er her peker til objektet
}
```


Pakken easyIO

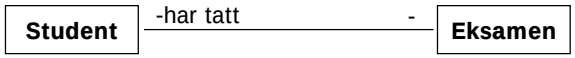
- **In** er en klasse for å lese fra tastatur eller fil.
  - **inLine()** leser neste linje.
  - **inChar()** leser neste tegn.
  - **endOfFile()** sjekker om slutt på filen.
  - **inInt** leser et heltall.
  - **inDouble()** leser et flyt-tall.
  - **inWord()** leser neste ord (omgitt av blanke).
  - **inText()** leser alle ord igjen på linjen.
  - **lastItem()** sjekker om flere ord å lese.
  - **close()** lukker filen.
- **Out** er en klasse for skriving til skjerm eller fil.
  - **out(. . .)** skriver ut en tekst (som kan være satt sammen av mange deler med +).
  - **outln(. . .)** er som out men avslutter linjen etterpå.
  - **out(*x*, *w*)** skriver double-tallet x med w desimaler.
  - **close()** lukker filen.

UML - klassediagrammer

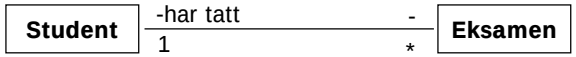
- Tegnet foran et element angir beskyttelsen:
  - – en private definisjon
  - + en public definisjon
  - # en protected definisjon
  - ~ en pakke-definisjon

UML – forhold mellom klasser 1

- Objekter laget av ulike klasser kan ha et visst forhold til hverandre:



- Vi kan også angi et *mengdeforhold*:



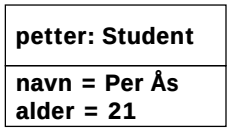
- Lovlige tall er
  - ***** vilkårlig mange (0, 1, 2, . . .)
  - **3** nøyaktig 3
  - **3..5** 3, 4, 5
  - **1..*** minst 1 (1, 2, 3, . . .)

UML – forhold mellom klasser 2

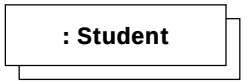
- Hvordan finner man det riktige tallet?
- Anta at du står i *ett* tilfeldig objekt av klassen og ser (langs forbindelsen) på den andre klassen:
  - Hvor mange objekter av det andre klassen ser du da *på et gitt tidspunkt?*
  - Dette tallet noteres *lengst fra deg.*
- Så flytter du deg til den andre klassen og gjentar det hele.

Objektdiagrammer 1

- Vi kan også legge inn opplysninger om objektvariablene og deres verdi:

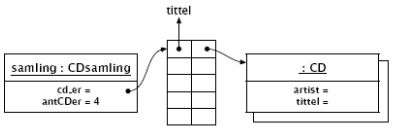


- Vi kan angi at det finnes mange objekter av samme klasse:



Objektdiagrammer 2

- **Pekere**
  - Vi kan angi hvilke pekere som peker hvor:
- **Arrayer**
  - Vi kan også angi arrayer (med pekere):
- **Tabeller**
  - Det finnes et eget UML-symbol for tabeller (HashMap-er).



Pensum V2007

- *Rett på Java*, 2005;
  - Kap 1-9, 12
- På nettet:
  - Arne Maus: Informasjonsteknologi, vitenskap og samfunnsmessige virkninger
  - *Personopplysningsloven*, http://www.datatilsynet.no/

Det viktigste

- Det viktigste vi håper dere har lært i INF1000:
  - Forstå hvordan man programmerer en datamaskin.
  - Forstå hvordan man bygger opp et program med klasser og objekter og kunne illustrere dette med UML-diagram.
  - Kunne skrive enkle programmer i Java.
  - Kunne finne algoritme for å løse enkle problemer og så formulere algoritmen i Java.