Uke 13 – Litt repetisjon

22 november 2005,
Arild Waaler
Inst. for informatikk, UiO

Eksamensrelevant repetisjonsstoff

- datatyper og konvertering mellom datatyper
- returtyper til metoder
- løkkekonstruksjoner: while og do
- statiske (kontra objekt-) metoder/variable
- objekter og ”this”
- modifikatorer: private/protected/public
- String-metoder for tekster
- equals vs. ==

Deklarasjoner og variabeltyper

- Mange grunnleggende, innebygde typer i Java

Type	Forklaring	Eksempel
byte	heltall	`byte b = 23;`
short	heltall	`short s = 256;`
int	heltall	`int i = 76234;`
long	heltall	`long l = 1234567890123L;`
float	desimaltall	`float f = 2.5F;`
double	desimaltall	`double d = 3.14;`
char	tegn	`char c = 'g';`
boolean	sannhetsverdi	`boolean b = true;`

Konstanter

```
final int DUSIN = 12;
final int SNES = 20;
final int GROSS = 144;
final int MAX_ELEVER_PR_KLASSE = 20;
```

- Kan ikke endres etter at de har fått første verdi pga. Java-ordet **final**
- Bruker bare store bokstaver (for å vise at dette er konstanter

Spesialnotasjon for ofte brukte operasjoner

```
j++;     //det samme som j = j+1;
j--;     //det samme som j = j-1;
j *= 22; //det samme som j = j*22;
j += a;  //det samme som j = j+a;
j -= 14; //det samme som j = j-14;
  ................
(flere etter samme mønster)
```


Heltallsregning og blandet flyt&heltall

```
int i = 14, j = 5, k ;

k = i/j;    //verdien av k blir 2 (avrunder nedover)
k = i%j;    //verdien av k = 4, resten etter: i/j


double x;

x = i/j;   // x blir også 2, fordi hele høyresiden
           // består bare av heltall. Først ved = (settes lik)
           // oversettes heltallet "2" til flyttallet "2.0"

x = i*1.0/j; // Nå blir x = 2.8 fordi ett av tallene
             // på høyre side var flyttall, og da oversettes
             // alle tallene først til flyttall før utregning
```


logiske variable og uttrykk

```
boolean c, b = i < 5;  // i er mindre enn 5
c = (j != 5);          // j inneholder ikke verdien 5;
System.out.println("Er i større enn 5" + b);
```

Operator	Beskrivelse	Eksempel
&&	Og (sann hvis begge ledd sanne)	`b = (x<y) && (y<z);`
||	Eller (sann hvis minst ett ledd er sant)	`b = (x<y) || (y<z);`
!	Ikke (snur sannhetsverdien)	`b = !(x<y);`
< og >	Mindre enn, større enn	`b = x < y;`
<= og >=	Mindre enn eller lik, større enn eller lik	`b = x <= y;`
==	Er lik	`b = (x==y);`
!=	Er ikke lik	`b = (x != y);`

Konvertering

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

```
double x = 7;
int a = 15;
double x = a;
double x = (7 + 14) * 3 - 12;
```

- Derimot vil Java 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!!
```

Konvertering *forts.*

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

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

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

  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?

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.

Nøtt

- Hva blir utskriften fra dette programmet?

```
class InkrementOperator {
   public static void main (String [] args) {
       for (int i=0; i<10; i++) {
         int j = i;
         int k = j++ + ++j;
         System.out.println("k = " + k);
       }
   }
}
```


Hva skjedde?

- Første løkkegjennomløp (i = 0):
  - `int j = i;`
    Variabelen j settes lik 0.
  - `int k = j++ + ++j;`
    j++ gir verdien **0** og øker j til 1
    ++j øker j til 2 og gir verdien **2**
    k settes lik 0 + 2, altså 2.
- Andre løkkegjennomløp (i = 1):
  - `int j = i;`
    Variabelen j settes lik 1.
  - `int k = j++ + ++j;`
    j++ gir verdien **1** og øker j til 2
    ++j øker j til 3 og gir verdien **3**
    k settes lik 1 + 3, altså 4.
- Osv.

Automatisk initialisering av arrayer

- Når en array blir opprettet, blir den automatisk initialisert (dvs verdiene er ikke udefinerte når den er opprettet).
  - `int[] k = new int[100];` // Nå er alle k[i] == 0
  - `double[] x = new double[100];` // Nå er alle x[i] == 0.0
  - `boolean[] b = new boolean[100];` // Nå er alle b[i] == false
  - `char[] c = new char[100];` // Nå er alle c[i] == '\u0000'
  - `String[] s = new String[100];` // Nå er alle s[i] == null
- Merk: String-arrayer initialiseres med den spesielle verdien `null`. Dette er *ikke* en tekststreng og må ikke blandes sammen med en tom tekst: `""`.
- For å kunne bruke verdien `s[i]` til noe fornuftig må du først sørge for å gi `s[i]` en tekststreng-verdi, f.eks. `s[i] = "Per"` eller `s[i] = "".`

Array-typer konverterer ikke

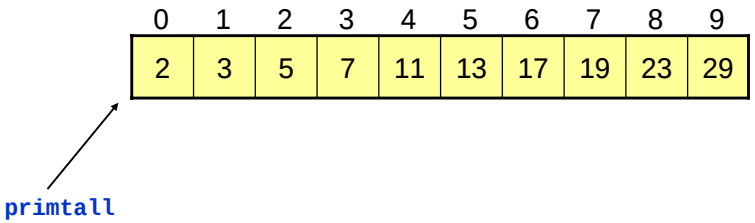
- Lovlig: int i = 4; double d=i;
- Ikke lovlig: int[] a = new int[10]; double[] d = a;

En array-variabel er en adresse

- Når vi deklarerer en array så refererer arraynavnet ikke til selve verdiene i arrayen, men til adressen (i hukommelsen) hvor verdiene ligger lagret.
- Resultatet etter at vi har utført

```
int[] primtall = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29};
```

kan visualiseres slik:



Å kopiere en array-adresse

- Hvis vi har

```
double [ ]  x = new double [100];
```

så vil

```
double [ ]  y = x;
```

medføre at adressen til arrayen vi opprettet kopieres over til variabelen y (dermed har vi fortsatt bare ett sett med verdier lagret, men vi har to referanser til arrayen: x og y.

Når arrayen blir for liten

- Noen ganger får vi behov for å utvide en array.
- Framgangsmåte for å utvide en array:
  - Deklarer og opprett en ny array **`temp`** som er av ønsket lengde
  - Flytt over alle verdier fra opprinnelig array til **`temp`**
  - Sett opprinnelig array-peker til å peke på **`temp`**
- Programkode:

```
/* Anta at tall er en int-array og at vi ønsker
   å utvide tall til dobbel lengde */

int[] temp = new int[2 * tall.length];
for (int i=0; i<tall.length; i++) {
   temp[i] = tall[i];
}
tall = temp;
```


Kopiering av arrayer

Vi kan ikke lage en kopi av en array x ved å skrive

```
int[] y = x;
```

siden dette bare medfører at adressen til arrayen legges inn i y.

- Skal vi lage en kopi, må vi først opprette en array til (f.eks. y), og så kopiere over verdiene en for en:

```
double[] y = new double[x.length];
for (int i=0; i<x.length; i++) {
    y[i] = x[i];
}
```

- Det finnes også ferdige verktøy i Java for å kopiere en array, f.eks:

```
int[] y = (int[]) x.clone();
```


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*.

Overlasting av metoder

- Flere metoder kan deklareres med samme metodenavn, forutsatt at Java klarer å avgjøre hvilken metode som skal kalles. Krav:
  - metodene har ulikt antall parametre eller
  - metodene har ulik type på noen av parametrene, og slik at Java alltid klarer å finne en entydig match
- Metoden (eller metodenavnet) sies da å være overlastet, og de ulike metodene med samme navn kan ha ulik returtype.
- Eksempel:

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

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

Overlasting - eksempel

```
public static void main (String[] args) {
    skrivUt(2,3);
    skrivUt(2.0,3.0);
    skrivUt(2.0,3);
}

static void skrivUt (double x, int y) {
      System.out.println("double+int: "+x + " , "+y);
}

static void skrivUt (double x, double y) {
      System.out.println("double+double: "+x + " , "+y);
}
```

```
double+int: 2.0 , 3
double+double: 2.0 , 3.0
double+int: 2.0 , 3
```


Klasse-variabel (=statisk variabel)

- Setter vi static foran en variabel, er det er bare **én** felles variabel med det navnet for alle objektene.
- Setter vi **`static`** foran en metode, har den bare utsikt til :
  - sine egne lokale variable og parametere
  - andre statiske variable og metoder
  - klassenavnene
- Statiske metoder og variable kan man få adgang til både
  - via klassenavnet og punktum
  - via peker til et objekt av klassen og punktum


```
class B {
       static int i = 0;
       double x = 0.0;
}
class A
{      int k;

    public static void  main ( String[] args)  {
           B b1 = new B(), b2 = new B();
           // endre klassevariable (det er bare en felles)
           System.out.println("b1.i :"+ b1.i+", b2.i:" + b2.i);
           b1.i = 4;
           System.out.println("b1.i :"+ b1.i+", b2.i:" + b2.i);
           // endre objektvariabel (en kopi i hvert objekt)
           System.out.println("b1.x :"+ b1.x+", b2.x:" + b2.x);
           b1.x = 2;
           System.out.println("b1.x :"+ b1.x+", b2.x:" + b2.x);
      }
}
```

```
>java A
b1.i :0, b2.i:0
b1.i :4, b2.i:4
b1.x :0.0, b2.x:0.0
b1.x :2.0, b2.x:0.0
```

```
class A2
{       int k;  // objektvariabel 'k'
    public static void  main ( String[] args)  {
            k = 1;
 }}
```

```
>javac a2.java
a2.java:6: non-static variable k cannot be referenced from a static context
         k = 1;
         ^
1 error
```

```
class A2
{      int k;

    public static void  main ( String[] args)  {
            A2 aa = new A2();
            aa.k = 1;
      }
}
```

```
>javac A2.java

>
```

this

- Av og til trenger vi en peker til det objektet metoden vi utfører er inne i. Java-ordet this gir oss alltid det.
- Brukes i to situasjoner:
  - Vi har en konstruktor, og parametrene til denne heter det samme som objekt-variable i objektet. Eks:

```
class A {
   int antall;
   A (int antall ){
       this.antall = antall;
   }
} // end A
.. A apek = new A(12);
```

  - Vi skal kalle en metode i et annet objekt (gjerne av en annen klasse). Da kan vi bruke this for å overføre en parameter til denne metoden om hvilket objekt kallet kom fra.

Ikke alt i et objekt bør være synlig fra resten av programsystemet - innkapsling

- Vi ønsker ofte at resten av systemet bare skal se deler av et objekt
  - eks: **`int saldo`** i Konto1-objektet bør være skjult, resten av programmet skal bare bruke **`settInn() og taUt()`** metodene.
- Vi kan regulere tilgangen til variable og metoder ved å sette enten :
  - **`private`**
  - **`public`**
  - **`protected`**
- foran en metode eller deklarasjonen av en variabel

For 'små¨' systemer hvor alle .java filene ligger på samme filområde, gjelder:

- Skriver vi:
  - **ingenting** foran en deklarasjon/metode, så er slike deklarasjoner fullt tilgjengelige for alle annen kode kompilert på samme filområde, men usynlig /sperret for kode kompilert på andre filområder.
  - **`private`** foran en deklarasjon/metode, så er den bare synlig fra kode i metoder deklarert i *samme klasse*, usynlig/sperret for all annen kode
  - **`protected`** foran en deklarasjon/metode, så er den synlig i samme klasser og subklasser og synlig i klassene på samme filområdet, men usynlig/sperret i andre klasser (på andre filområder).
  - **`public`** så er metoden/variabelen synlig for all annen kode.
- Slik delvis sperring av adgang til særlig variable, sikrer oss at vi kan bestemme fullt ut selv i en klasse hvordan en variabel skal endres.

To måter å programmere på

Statisk programmering:

- Var fokus i starten av kurset
- Vi lager ikke objekter av klassene
- Alle variable og metoder er deklarert som static
- Begrepsmessig enkelt, men lite egnet for større programmer

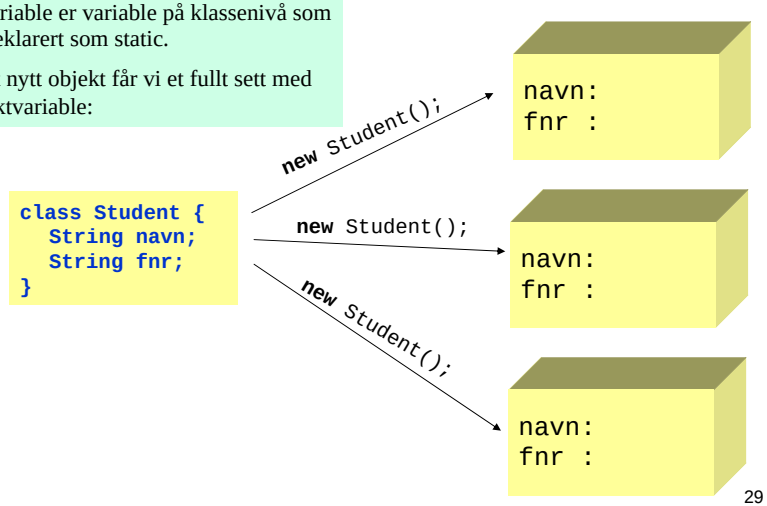
Programmering med objekter:

- Er fokus for resten av kurset (og eksamen).
- Vi lager objekter av klassene (noen eller alle)
- Variable og metoder er vanligvis *ikke* deklarert som static
- Begrepsmessig noe mer komplisert, men mye bedre egnet for større programmer

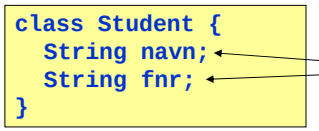
Objektvariable

Objektvariable er variable på klassenivå som *ikke* er deklarert som static.

For hvert nytt objekt får vi et fullt sett med nye objektvariable:



Objektvariablenes levetid



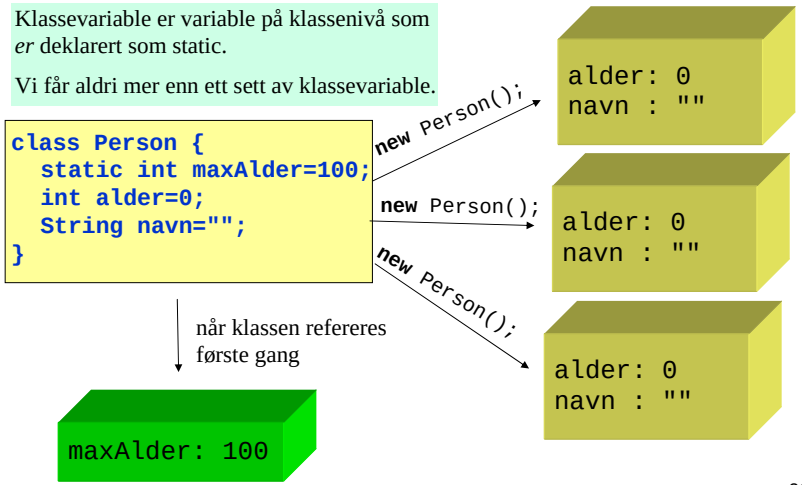
Disse variablene blir deklarert når vi lager et objekt av klassen ved å skrive **`new Student(),`** og de lever så lenge objektet lever.

- Objektvariablene blir til når objektet blir til (med new)
- Objektvariablene lever så lenge objektet finnes

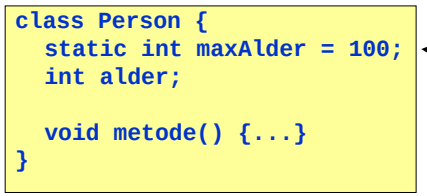
Klassevariable

Klassevariable er variable på klassenivå som *er* deklarert som static.

Vi får aldri mer enn ett sett av klassevariable.



Klassevariablenes levetid



Denne variabelen blir deklarert når klassen Person *blir referert til for første gang* under kjøringen av programmet.

Variabelen lever helt til programmet avsluttes.

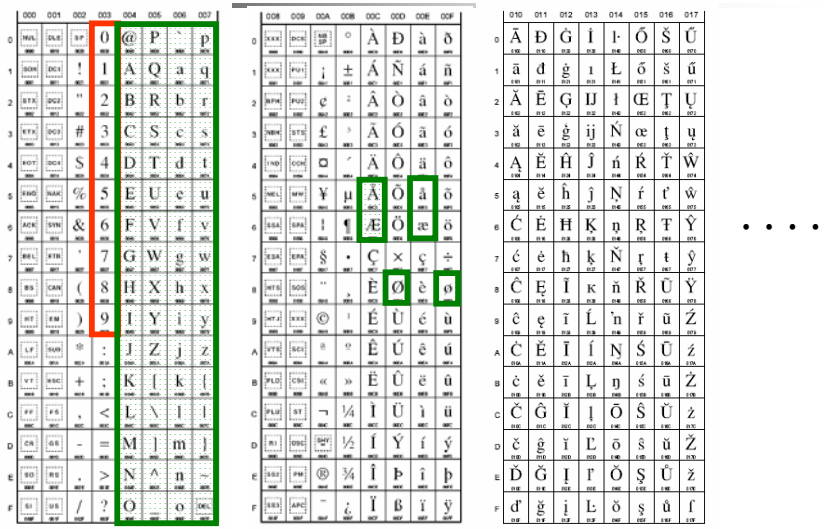
Første gang klassen Person blir referert til = første gang programeksekveringen "møter på" Person-klassen, f.eks. :

```
.... new Person() ....
.... Person.maxAlder ....
.... Person.metode() .....
```

Tekster: bruk av spesialtegn

- Både i char-uttrykk og String-uttrykk kan vi ha mange ulike typer tegn
- Alle Unicode-tegn er tillatt
- Unicode er en standard som tillater tusenvis av tegn (ulike varianter fins; den som støttes av Java tillater 65536 ulike tegn)
- Alle tegnene kan angis som '\uxxxx' hvor hver x er en av
  0, 1, 2, ..., 9, A, B, C, D, E, F
  Eksempel: '\u0041' er tegnet 'A'
- Noen spesialtegn har egen forkortelse:
  - `\t` tabulator
  - `\n` linjeskift
  - `\"` dobbelt anførselstegn
  - `\'` enkelt anførselstegn
  - `\\` bakslash

Unicode (http://www.unicode.org)



Teste om to tekster er like

- For å teste om to tekststrenger er like, brukes equals:

```
// Anta at s og t er tekstvariable (og at s ikke har verdien null)
if (s.equals(t)) {
   System.out.println("Tekstene er like");
} else {
   System.out.println("Teksten er forskjellige");
}
```

- Bruk av == virker av og til, men ikke alltid:

```
String s = "abc";
String t = "def";
String tekst1 = s + t;
String tekst2 = s + t;
```

Nå er `tekst1.equals(tekst2)` true, mens `tekst1 == tekst2` er false.

De enkelte tegnene i en tekststreng

- Tegnene i en tekststreng har posisjoner indeksert fra 0 og oppover

0	1	2	3
'k'	'a'	'k'	'e'

- Vi kan få tak i tegnet i en bestemt posisjon:

```
String s = "kake";
char c = s.charAt(1);
// Nå er c == 'a'
```

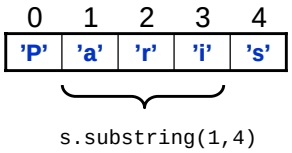
- Vi kan erstatte alle forekomster av et tegn med et annet tegn:

```
String s1 = "kake";
String s2 = s1.replace('k', 'r');
// Nå er s2 en referanse til tekststrengen "rare"
```

Deler av en tekststreng

- Vi kan trekke ut en del av en tekststreng:

```
String s = ”Paris”;
String s1 = s.substring(1,4);
// Nå er s1 tekststrengen ”ari”
```



- Generelt:

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

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”
```


Konvertere mellom små og store bokstaver

- Vi kan konvertere fra små til store bokstaver:

```
String s = ”Jeg ER 18 år”;
String s2 = s.toUpperCase();
// Nå er s2 tekststrengen ”JEG ER 18 ÅR”
```

- Vi kan konvertere fra store til små bokstaver:

```
String s = ”Jeg ER 18 år”;
String s2 = s.toLowerCase();
// Nå er s2 tekststrengen ”jeg er 18 år”
```

- Det finnes tilsvarende metoder for å konvertere char-verdier:

```
char c = ’x’;
char c2 = Character.toUpperCase(c);
char c3 = Character.toLowerCase(c);
```


Lag stor forbokstav i hvert ord av en tekststreng

```
import easyIO.*;

class StorForbokstav {
  public static void main (String[] args) {
    In tast = new In();
    System.out.print(”Skriv en tekst: ”);
    do {
      String s = tast.inWord();
      String t = ””;
      if (s.length() > 0) {
            char c = Character.toUpperCase(s.charAt(0));
            t = c + s.substring(1);
      }
      System.out.print(t + ” ”);
    } while (!tast.lastItem());
  }
}
```


Alfabetisk ordning

- Anta at s og t er tekstvariable (og at s ikke har verdien null)
- Er s foran t i alfabetet?

```
int k = s.compareTo(t);

if (k < 0) {
  System.out.println("s er alfabetisk foran t");
} else if (k == 0) {
  System.out.println("s og t er like");
} else {
  System.out.println("s er alfabetisk bak t");
}
```

Inneholder en tekst en annen?

- Anta at s og t er tekstvariable (og at s ikke har verdien null)
- Inneholder s teksten t?

```
int k = s.indexOf(t);

if (k < 0) {
  System.out.println("s inneholder ikke t");
} else {
  System.out.println("s inneholder t");
  System.out.println("Posisjon i s: " + k);
}
```


Starter en tekst med en annen?

- Anta at s og t er tekstvariable (og at s ikke har verdien null)
- Starter s med teksten t?

```
boolean b = s.startsWith(t);

if (b) {
  System.out.println("s starter med t");
} else {
  System.out.println("s starter ikke med t");
}
```


Slutter en tekst med en annen?

- Anta at s og t er tekstvariable (og at s ikke har verdien null)
- Slutter s med teksten t?

```
boolean b = s.endsWith(t);

if (b) {
  System.out.println("s ender med t");
} else {
  System.out.println("s ender ikke med t");
}
```