# Forkurs i Java, in105 - jan 1999

**Arne Maus**
**Inst for Informatikk**
**Univ. i Oslo**

## Oversikt

1. Hva er Java - ett språk mange bruksformer
2. Java og objekter
3. Vanlige setninger
4. Initialisering og opprydding
5. Pakker og grensesnitt
6. Polymorfisme
7. Gjenbruk/subklasser
8. Arrayer og 'mengder'
9. Feil-håndtering
10. I/O (filer)
13. AWT - vinduer og knapper
14. Tråder og parallellitet
15. Java og nettet - litt CORBA

©Arne Maus, jan. 1999

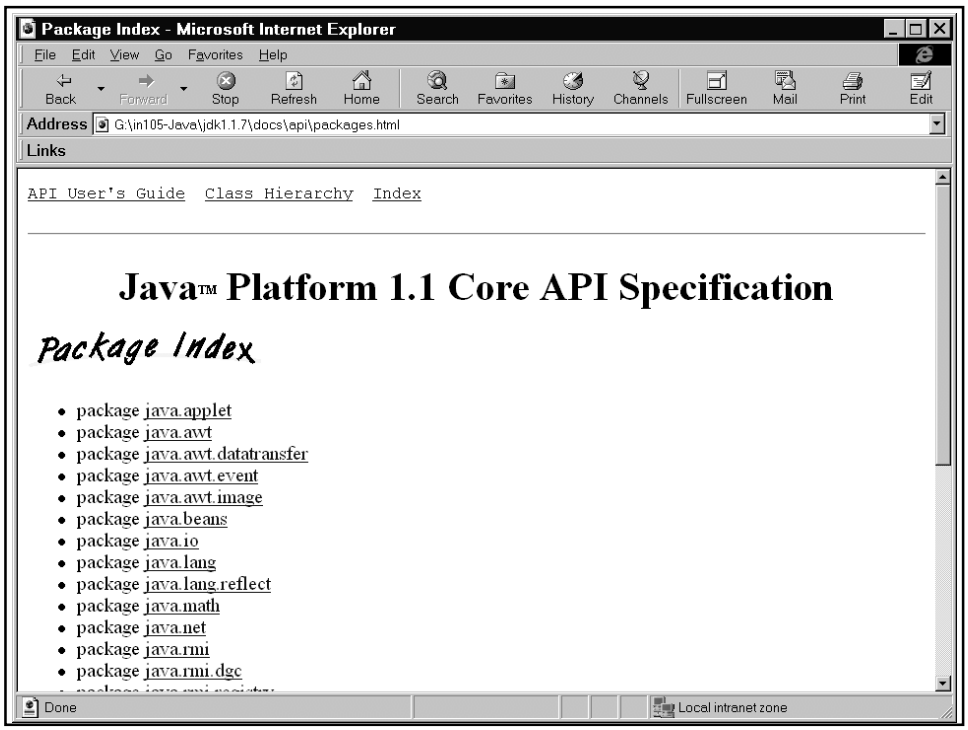
# 1. Hva er Java - ett språk mange bruksformer

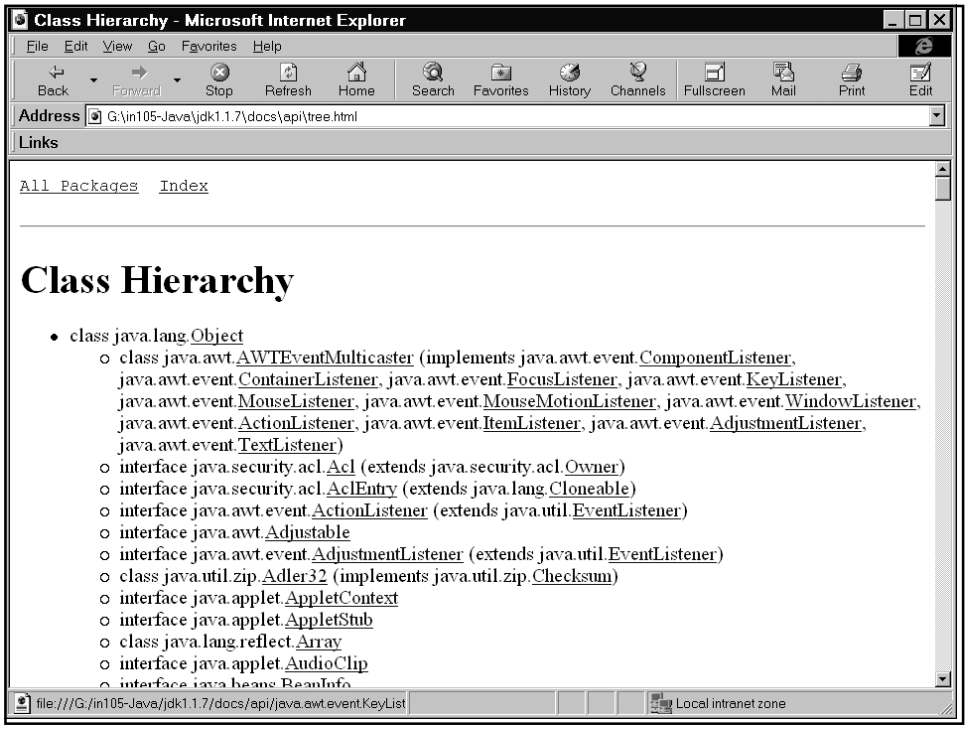
- Type språk, anvendelse
- Lage vanlige programmer (application)
- Lage Web-snutter (applet)
- Lage komponenter (bønner, beans)
- Java og biblioteker (GUI, database, ...)
- Java og nettet
- Java - CORBA



# Type språk, anvendelse

- 'Rent' OO-språk, designkriterier:
  - Enklere (enn C++) å lage programmer, fange feil ved kompilering
  - Portabelt (flytte uten omkompilering)
- Noe enklere ("mindre") enn C++, lite bakoverkompatibilitet
- Kompileres til 'Bytekode' (kode for en 'tenkt' maskin), som:
  - Tolkes av programvare (interpret), eller
  - oversettes videre (JIT-kompilering, full kompilering,..)
- Bedre og sikrere, men langsommere enn C++
- Mange typer av program lages med Java
- Stadig forbedringer (1.0, 1.1, 1.2(=2.0), nye biblioteker - gamle fjernes) språket utvikler & utvider seg -
  - ca 1600 klasser i Javabiblioteket i ver 2.0, pakket i ca. 60 pakker


Package Index - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites History Channels Fullscreen Mail Print Edit
Address G:\in105-Java\jdk1.1.7\docs\api\packages.html
Links
API User's Guide Class Hierarchy Index
Java™ Platform 1.1 Core API Specification
Package Index
package java.applet
package java.awt
package java.awt.datatransfer
package java.awt.event
package java.awt.image
package java.beans
package java.io
package java.lang
package java.lang.reflect
package java.math
package java.net
package java.rmi
package java.rmi.dgc
Done
Local intranet zone


Class Hierarchy - Microsoft Internet Explorer
File Edit View Go Favorites Help
Back Forward Stop Refresh Home Search Favorites History Channels Fullscreen Mail Print Edit
Address G:\in105-Java\jdk1.1.7\docs\api\tree.html
Links
All Packages Index
Class Hierarchy
class java.lang.Object
class java.awt.AWTEventMulticaster (implements java.awt.event.ComponentListener, java.awt.event.ContainerListener, java.awt.event.FocusListener, java.awt.event.KeyListener, java.awt.event.MouseListener, java.awt.event.MouseMotionListener, java.awt.event.WindowListener, java.awt.event.ActionListener, java.awt.event.ItemListener, java.awt.event.AdjustmentListener, java.awt.event.TextListener)
interface java.security.acl.Acl (extends java.security.acl.Owner)
interface java.security.acl.AclEntry (extends java.lang.Cloneable)
interface java.awt.event.ActionListener (extends java.util.EventListener)
interface java.awt.Adjustable
interface java.awt.event.AdjustmentListener (extends java.util.EventListener)
class java.util.zip.Adler32 (implements java.util.zip.Checksum)
interface java.applet.AppletContext
interface java.applet.AppletStub
class java.lang.reflect.Array
interface java.applet.AudioClip
file:///G:/in105-Java/jdk1.1.7/docs/api/java.awt.event.KeyList
Local intranet zone

## Java - egenskaper

- Alt er objekter (eller enkle datatyper (int, char, double, ..))
- Alle egenskaper - dvs. data og metoder (prosedyrer) er inne i klasser
- Alle utførende setninger er inne i metoder i klasser.
- Det finnes klasse-metoder og -data i tillegg til objekt-metoder og -data
- Bekyttelse (private, protected, public, 'friendly'/package))
- Enkel arv - alle klasser er sub(eller subsub,...) klasse av class Object
- Dynamisk binding - alle metoder er (med mindre man sier noe annet) 'virtuelle'
- Abstrakte klasser og grensesnitt
- Søppeltømmer
- Unntaks(feil)håndtering
- Parallellitet: Tråder
- Spesielle mekanismer for www (applet)

©Arne Maus, jan. 1999


## N.B. programfilen heter det samme som klassen

**Kompilering og kjøring:**

- Når du lager et program:
  - Filnevnet MÅ være lik Klassenavnet (til den ene klassen på filen som er **public** og som resten av systemet har direkte adgang til)
  - Filutvidelsen = .java
    dvs en fil med klassen ' Test' heter **Test.java**
- Kompilering:

  >javac Test.java
- Kompileringen genererer filen: Test.class
  - som inneholder programmet i portabel Bytekode.
  - er det flere klasser x,y,z, lages det en '.class' fil for hver klasse (x.class, y.class, z.class)
- Kjøring av programmet:

  >java Test

©Arne Maus, jan. 1999

## Lage vanlige programmer (application)

- Inneholder en metode (prosedyre) **`public static void main(..)`** inne i en brukerdefinert klasse. Intet krav til navnet på den klassen
- **main** kalles ved av kjøresystemet ved start.
- Kjøre med vanlig utskrift til linjeorientert skjerm, eller til GUI-basert skjerm. (AWT= vinduer og knapper)
- Kan lese/skrive filer, databaser o.l
- Java er som andre programmeringsspråk

©Arne Maus, jan. 1999


Kildefil: Test.java (ant. byte: 247)

```
//Fil: Test.java
import java.io.*;

public class Test
{
        public static void main (String [ ] args )
        {
                test am = new Test( );
                System.out.println("Arne: 3 * 4 er:" + am.xx(4) );
        }

        protected int xx (int i)
        { return i*3;}

}
```

Kjøring:

```
E:\java-unix>javac Test.java

E:\java-unix>java Test
Arne: 3 * 4 er:12
```

De mest interessante delene av Test.class (=Bytekode, totalt 693 byte)

```
`0` ´`-()Ljava/lang/String;´`~()
V´`¯(I)I´` (I)Ljava/lang/StringBuffer;
´` (Ljava/lang/String;)V´`
([Ljava/lang/String;)V´`
<init>´`flArne: 3 * 4 er:´`¯
Code´`
ConstantValue´`
Exceptions´`flLineNumberTable
´` Ljava/io/PrintStream;´`
```

## Oppgave 1

- Oppgave nå (5 min ):
  - Ta utgangspunkt i det første programmet og lag et enklere program, Test2, som skriver ut ditt navn + verdien av et deklarert heltall 'j'
- Hjemme-oppgave
  - Lag et program som skriver ut de tre første argumentene fra kommandolinjen

©Arne Maus, jan. 1999

## jdk - mange 'verktøy' i en kasse
### (vi bruker bare noen)

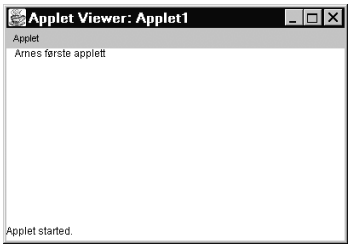
| | |
|---|---|
| >javac | - kompilerer (kildekode -> Bytekode , dvs .java-fil til en .class-fil for hver klasse på .java-filen) |
| >java | - utføre Bytekode (interpret) |
| >jre | - enklere utgave av 'java' |
| >appletviewer | - brukes for å kjøre/teste Applets |
| >jdb | - debugger |
| >javap | - dissassemblerer/tyder en .class-fil |
| >javadoc | - lager html dokumentasjon fra en .java-fil (eller helst av en pakke) |
| >jar | - lager en .jar fil av flere .class-filer (dvs. av en pakke) |
| ..... | *enda flere .....* |
| >jws | - grafisk utviklingsvertøy (a la Visual C++) |



## Lage en Web-snutt (applet)

- Hensikt: Lage lite program som kan startes fra en html-side.
- En av klassene er subklasse av klassen Applet (som hentes fra et bibliotek : 'applet')
- Lastes så opp av en WWW-side (kodet i 'html') og kjører der
- En applet kan bla. ikke gjøre IO mot filer pga:
  - sikring mot virus
  - Kjører i en egen beskyttet 'sikkerhets'-tolker (sandkasse) i nettleser
  - Dette kalles 'Sandkassen' - er endret/utvidet i ver 2.0

Applet -eksempel




```
//App1et1.java

import java.awt.*;
import java.applet.*;

public class Applet1 extends Applet {
public void paint(Graphics g) {
g.drawString("Arnes første applett", 10,
10);
}
}
```

Tilhørende lagt inn på en 'html-side'(fil: xyz.html):

```
<applet
 code=Applet1
 width=500
 height=200>
 </applet>
```

## Lage komponenter (beans)

- Lage komponenter med veldefinerte grensesnitt som passer inn som del av et (annet) større system
- En javabønne har
  - for hver egenskap XXX (f.eks data av typen int, float, String..) en set- og en get-metode (getXXX() )
  - generelle add- og remove- metoder for ulik bruker-IO (mus o.l)
  - egen-definerte public metoder (problem-spesifikke)
  - gir info om sitt eget grensesnitt dynamisk
- enhver klasse kan være en 'bean', men
  - Standardisert grensesnitt gjøre det mulig/lettere for andre komponenter/beans å snakke med /bruke denne 'bønna'

Bønner lite/ikke dekket av in105

©Arne Maus, jan. 1999

## Java og nettet

Mange måter å bruke nettet på i Java:

- **Applet via HTML side**
- CGI
- Sockets
- **RMI - Remote Method Invocation**
- (Servlet, JDBC)
- DCOM
- **CORBA**



## Java - CORBA

- Klient /tjener i 2 eller 3 lag (flere)
- CORBA er en generell teknologi
- Oppretter kontakt som om det var lokale kall mellom de ulike lagene, uansett:
  - Hvilket språk de ulike lagene er skrevet i
  - hvilke op.sys som nyttes
  - hvilken maskiner og nett og ORB som brukes
  - hvordan data er representert
- CORBA er del av Java 2 (1.2) også IDL -kompilator
- (RMI er raskere på store pakker enn CORBA)

## 2. Java og objekter

- Alt er objekter
  - Unntatt enkle data som int, float,..
  - Enkle datatyper har gitt lengde (int er 32 bit, osv) ikke som C++
- Alltid adgang via 'handles' (håndtak) til objekter
- Et håndtak kan bare peke på objekter
  - av riktig type, eller subklasse av denne
- Vi må lage alle objekter med **new**
- Arrayer er egne objekter (og må også lages med new)
- Array.length og String.length( )
- Døde objekter samles inn (garbage collector)
- static = klassevariable
- Javadoc

©Arne Maus, jan. 1999


## 'Alt' er objekter

- Unntatt enkle data som int, float,..
  - effektivitet
  - Finnes klasse-alternativer for basaltypene (med stor forbokstav og hele navnet - basaltypene har små forbokstav)- f.eks

```
int i = 1;
Integer  j;
....
 j = new Integer(i);                    // kan IKKE si:  j = i;
 j = new Integer( j.intValue() + 1); // kan IKKE si:  j = j + 1;
```

- Enkle datatyper har gitt lengde (int er 32 bit, osv) ikke som C++

©Arne Maus, jan. 1999

# Alltid adgang via 'handles' (håndtak) til objekter

- Alle objekter ligger på 'heap'en - dvs:
  - ikke lokalt 'inline' som C++ kan
- Vi kan selvsagt peke på objekter vi er deklarert som pekere til
- Vi kan prøve å peke på objekter som er subklasse av vår type, men det kan gi kjøretidsfeil.
- 'null' er som Simula 'none'
- Peker vi på supertype - objekter, må vi gjøre typekonvertering :

```
Integer j;
Object k;

j = new Integer( i);
k = j;                    // OK alltid, j er et 'større objekt' enn k
j = (Integer) k;          //OK kompilering, men IKKE  j = k;
j = new Integer ( ((Integer)k).intValue() + 1); // OK kompilering, husk parentesen ( )
                                  // MEN gir kjørefeil hvis k IKKE var en Integer
```



# Vi må lage alle objekter med new

```
// fil: biltest.java
import java.io.*;

class Bil{
  Bil neste ;
  int reg_nr = 0;
}

public class biltest{
    Bil first = new Bil( );
    Bil sist = first;
    Bil refA;

  public static void  main
     ( String[] args)          {
       int i = 0;
       biltest b = new biltest( );
       b.make_queue( );
  }
```

```
void make_queue ( )
   {   int i;
       System.out.println("Start test");

       for (i= 1; i <= 1000; i++){
         refA = new Bil( );
         sist.neste = refA;
         sist = sist.neste;
       }
       System.out.println
        ("Slutt test, antall biler er : " +  i);
   }
} //
```

```
Kjøring:
E:\Sun-NT-Java\PC-Java>java biltest
Start test
Slutt test, antall biler er : 1001
```

## Tekster i Java

- Character array -
  - char buf [] = new char[128];
- klassen String - tekst-konstanter kan ikke endres
  - String s1, S2 = "Hei, hå nå er det jul igjen";
    s1 = S2 + " - det er vel noe overdrevet";
    - brukes mye
- klassen StringBuffer - tekster som kan endres

  StringBuffer sb = new StringBuffer("Hei hå,nå er det snart pøske") ;
  sb. setCharAt(sb.length( ) - 4, 'å');
  - ikke så mye brukt

( class Text i 'JavaGently'-boka er for et annet formål:
Provides simple input/output from the keyboard and files.)

©Arne Maus, jan. 1999


## Arrayer og String er egne objekter, Array.length og String.length()

```
// fil: astest.java
import java.io.*;

class Bil{
  Bil neste ;
  int reg_nr = 0;
}

public class astest{
          int [ ]  a   = new int [100];
          Bil [ ]  b  = new  Bil [14];
          String s1 ="Hei, hei";

         // array av string-pekere
          String [ ] sa;

          public static void  main
            ( String[ ] args){
              new astest( ).test( );}
```

```
void test( ) {
    for (int i = 0; i < b.length ; i++)
     b[i] = new Bil();

    System.out.println("a.length = " + a.length );
    System.out.println("b.length = " + b.length );
    System.out.println("s1=" + s1+ ",
                        s1.length ( ) = " + s1.length( ));
    sa = new String [17];
    System.out.println("sa.length = " + sa.length );
    System.out.println("sa[0]=" + sa[0] );

    if ( sa[0] != null)    System.out.
          println("sa[0].length( ) = " + sa[0].length( ) );
    else   System.out.println("Null-peker");
  }
} //
```

©Arne Maus, jan. 1999

## Kjøring

```
E:\Sun-NT-Java\PC-Java>java astest
a.length = 100
b.length = 14
s1=Hei, hei, s1.length() = 8
sa.length = 17
sa[0]=null
Null-peker

E:\Sun-NT-Java\PC-Java>
```



## Døde objekter samles inn (garbage collector)

- Alle objekter som ikke har et håndtak (handle/peker) som peker på seg, tas før eller siden av et søppeltømmingsprogram som i Simula, men ikke i C++
- Plassen gis tilbake til systemet
- Stor fordel
  - Et kjørende program 'vokser ikke' (p.g.a. uaktuelle, 'gamle' objekter)
- Liten ulempe
  - Tar tid (ca. 10 % tillegg på kjøretida) og kan gi 'rykkete' oppførsel ved interaktiv bruk

## static = klassevariable

- Statiske variable er **felles** for alle objekter av en klasse
- Det er bare ***ett*** stk. av hver av disse
- Et 'klasse-objekt' som inneholder disse statiske opprettes før programmet startes
- Derfor eksisterer 'main( )' ved start av programmet

```
public static void main ( String [ ] args) {.....;}
```



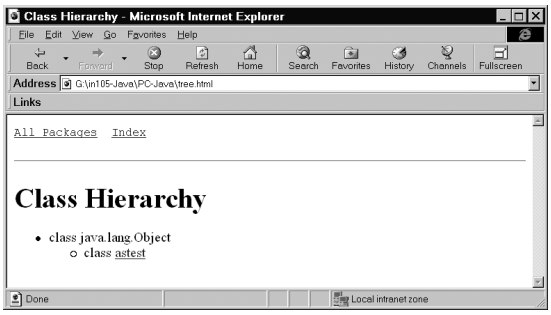
## Javadoc

```
>javadoc  xxx.java
```

- Genererer en standard (god) dokumentsjon av programmet
- Kommentarer i programmet (kodet på en angitt måte) gir enda bedre dok med 'javadoc'

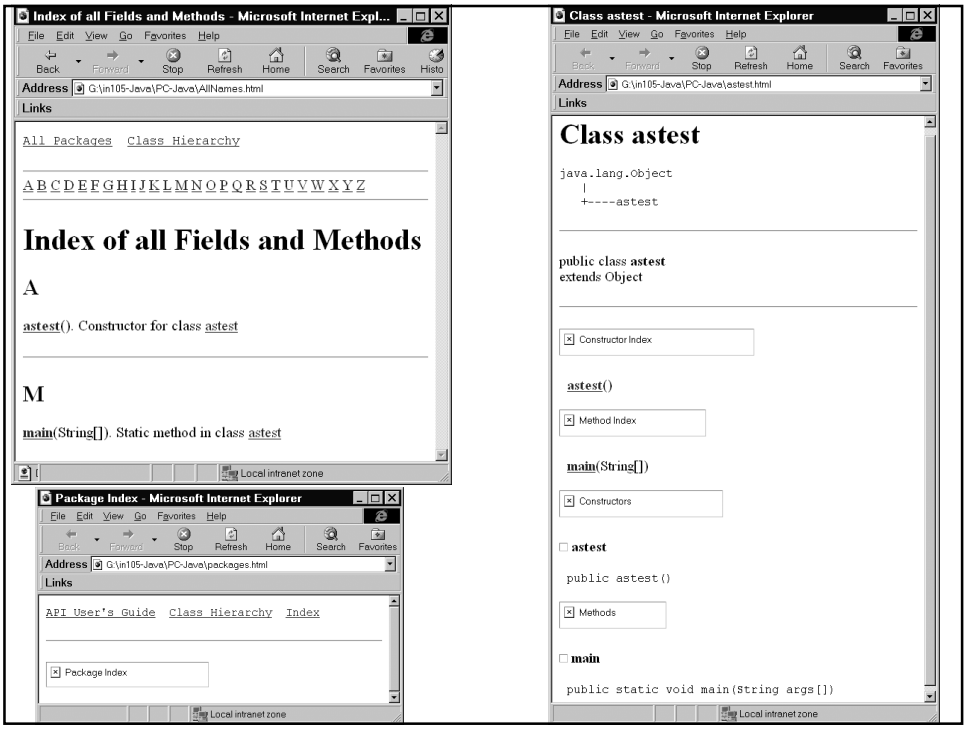
## >javadoc astest.java
## (kjøre javadoc på biltest-eksempelet)

```
-rw-------  1 arnem    arnem      2019 Dec 17 13:30 AllNames.html
-rw-------  1 arnem    arnem      1438 Dec 17 13:30 astest.html
-rw-------  1 arnem    arnem       463 Dec 17 13:30 packages.html
-rw-------  1 arnem    arnem       464 Dec 17 13:30 tree.html
```




©Arne Maus, jan. 1999





## 3. Vanlige setninger I

- Operatorer (+, *,..) har vanlig presedens, bruk parenteser
- shift og bit-operatorer
- Håndtak og alias-problemet
- String og +
- I boolske uttrykk har vi bare betinget 'and' og betinget 'or':

```
if (p != null && p.j == 12)...;  // test p.j == 12 utføres bare hvis p!= null
if ( i == 3 || k > 13) ...;      // test k >13 utføres bare hvis   i != 3
```

- kommentarer:  //  og  /*  */



## Vanlige setninger II

Eks:

- i = j + 4;
- **int** i, j, k = 17, m;
- **Bil** b1,b2;  // deklarerer pekere (håndtak) til bil-objekter

- Setninger og deklarasjoner kan blandes (gjør det helst ikke med unntak av :

```
for (int i = 1; i < n; i++)
{.....}
```

- semikolon **;** er del av en setning - og 'ligger i' **}**

siste setning før } trenger ';' fordi } avslutter {...} (og bruker sitt ; til det)

```
if (...) {int i = 4; j = 2 + i;}
 class A { int a,b,c; }
```

## Vanlige setninger III

- while( ), do...while()
- for( i = 0 ; i <n ; i++ ) s;  og : for( int i = 0 ; i <n ; i++ )
- if (B) S1; else s2;
- switch (int eller char utrykk)
  case konst1:  setninger; break ;
  case konst2:  setninger; break ;
  ……
  default: setninger;
  erstatter en lang: if().. else if() .. else if ()…….
- break
  - hopp ut av omsluttende if, while, for eller switch
- continue
  - Hopp til neste runde (til toppen) i omsluttende while, for -setning

©Arne Maus, jan. 1999


## 4. Initialisering og opprydding

- Alt intitialiseres (av deg)
- Konstruktor
  - default
  - flere
  - subklasser
- Metode overlasting
- this
- super
- finalize( ) - bare ved søppeltømming

©Arne Maus, jan. 1999

## Konstruktor

- Konstruktoren er en metode som kalles når vi oppretter et objekt ved new
- Vi kan skrive en (eller flere) slike til en klasse. De har samme navn som klassen.
- Skriver vi flere, må de ha ulike sett parametre før eller siden (antall og/eller typer) . Det aktuelle kallet ved **new** avgjør da (ved antall/type parametre som er brukt) hvilken som kalles
- Har vi ikke skrevet noen, får vi en 'default' konstruktor med null parametre som ikke gjør noe annet enn evt. kalle super-klassens 'default' konstruktor, osv
- Er klassen en subklasse, kalles super-klassens konstruktor først (må gjøres eksplisitt fra 'første linje' i konstruktoren i subklassen hvis man har egne konstruktorer):

```
super(...param..);
```

©Arne Maus, jan. 1999


## Metode overlasting

- Som ved konstruktorer, kan samme metoder med samme navn deklareres ved siden av hverandre
- De må ha ulike parametersett (antallet og/eller typene må før eller siden være ulike) - som konstruktorene
- Aktuelt kall med sine parametre kan da sammenlignes med de ulike parameterlistene, og riktig metode velges (ved kompilering)

**N.B. Dette er ikke det samme som 'virtuelle' metoder .**

©Arne Maus, jan. 1999

## this

- 'this' er et reservert ord som alltid gir et håndtak til det objektet vi er inne i.



## super

To typer bruk:

1) Kall av superklassens konstruktor med parentes og parametre
   (N.B må være første setning i egen konstruktor)
2) Generell peker til superklassen - brukes f.eks for å kalle den 'gamle metoden' når denne redfineres virtuelt i en subklasse:

```
class A { int aa(){return 1; } }
class B extends A {int aa(){ return ( super.aa()+2 ); }}
```

# finalize( ) - bare ved søppeltømming

- Metode som kalles automatisk når dette objektet søppeltømmes.
- Lite nyttig

©Arne Maus, jan. 1999


```
//Fil: garb.java
import java.io.*;

public class garb
{     Tell t ;      int gen = 0;

                public static void main (String [] args )
                { garb g = new garb();
                  g.objgen(10);
                  g.objgen(100);
                  g.objgen(1000);
                  g.objgen(10000);
                 g.objgen(100000);
                }

                void objgen (int i)
                { System.out.println("Genererte obj= “+ gen + “Ant garb:” + Tell.tell);
                   for (int j = i ; j > 0 ; j--) t = new Tell();
                   gen += i;
                  System.out.println("Genererte obj= “+ gen + “Ant garb:” + Tell.tell);
                 }
}

class Tell
{ static int tell;
  protected  void finalize(){tell++;}
  int [] A = new int [100];
}
```

**Kjøring:**

```
E:\java-unix>java garb
Genererte obj= 0Ant garb:0
Genererte obj= 10Ant garb:0
Genererte obj= 10Ant garb:0
Genererte obj= 110Ant garb:0
Genererte obj= 110Ant garb:0
Genererte obj= 1110Ant garb:0
Genererte obj= 1110Ant garb:0
Genererte obj= 11110Ant garb:10288
Genererte obj= 11110Ant garb:10288
Genererte obj= 111110Ant garb:109988
```

## Oppgave nå

- Lag en klasse T med en default konstruktør (dvs. uten parametre) som skriver ut en melding.
  Lag et objekt av denne klassen.
- Legg på en ny konstruktør i T med en integer parameter
- Legg på en subklasse T2 av tell med konstruktører og lag et objekt av klassen T2

©Arne Maus, jan. 1999


```
//  file:T.java
import java.io.*;

public class T {
      int i ;
         public static void  main ( String[] args)
         { T2 t =new T2( ),  tt = new T2(5);   }

      T ( )      { i = 17;}
      T (int j) { i = j;}
}

class T2 extends T {
      T2 ( )      { super( );}
      T2 (int j) { super(j);}
}
```

## 5. Java og pakker (= biblioteker som GUI, database, ...)

- Utstrakt bruk av 'package' i Java
- En pakke består av en rekke filer på et filområde
- Filområdets navn er pakkens navn
- Hver fil (av typen .class) inneholder:
  - Bare en klasse som er public - det er denne som andre programmer og filer kan referere
  - + evt. andre (indre) klasser
- kan lages med setningen 'package'
- hentes inn med 'import '



## Pakker og CLASSPATH

- Navnet som import bruker er er av typen 'x.y.z' eller x.y.*
  - x er filområde (evt. x1.x2....xn)
  - y er filområde og pakkenavnet
  - z er navnet på klassen man vil importere
  - * betyr at man importerer alle klassene i pakken y
  - x.y. oversettes til 'x\y\' - dvs. en relativ adresse i et filsystem
- CLASSPATH er en operativsystem-varabel som sier hvor kjøresystemet skal lete etter 'import' klassene, eks:
  - eks: >set CLASSPATH=.;C:\jdk1.1.6\lib
  - Dvs i eksempelet letes først fra underfil-katalog '.' (det inneværende), dvs: .\x\y\ etter fil (og da klasse) z og deretter i C:\jdk1.1.6\lib\x\y etter z.

## hyppig brukte pakker.*

import

- java.awt.* - Grafisk brukergrensesnitt
- java.awt.event.* - Event-håndtering
- java.util.* - Div, hyppig brukte klasser
- java.applet.* - Lage applet
- java.io.* - linjeorientert og fil- I/O
- java.lang - er automatisk 'importert'
  - (inneholder System, Math, Thread, ... osv)



## 6. Polymorfisme (kap. 7)

- Skille:
  - overloading : flere metoder på samme nivå, ulike parametre
  - virtuelle/polymorfisme: sen binding ved eksekvering, samme parametre
- 'final' metode = ikke mulig å redefinere virtuelt i subklasser
- upcast og downcast
- Abstrakte metoder og klasser
- grensesnitt (interface)
- indre (lokale) klasser i en klasse

## upcast og downcast

- Casting : Utgi seg for å være objekt av en annen klasse, f.eks ved metodekall eller i vanlige setninger
- Upcast: La et håndtak brukes der man forventer et håndtak til super-klassen (eller super.super,.. klassen) - Dvs. 'opp' i klassetreet (er sikkert, man inneholder minst super-klassedelen av objekter)
- Downcast: Bruke et håndtak der man forventer et håndtak til en sub- (subsub,..) klasse (er usikkert, kan gi feil under kjøring)

```
class A{int i;...}
class B extends A {int j;...}
B b = new B();  A a = new A(); ..
                k = (A) b.i;  // O.K.
                k = (B) a.j;  // Feil i dette eks.!!
```



## Abstrakte metoder og klasser

- Abstrakte metoder
  - abstract før deklarasjonen betyr at subklasser må gi kode (redefinere) denne metoden.
  - Abstrakte metoder kan ikke kalles
  - Man kan ikke lage objekter av slike klasser, men av subklasser som gir kode for de abstrakte metodene
- Abstrakte klasser
  - klasse med 'abstract' før 'class'
  - kan ikke lage objekter (men subklasser) av denne
  - kan (men må ikke) inneholde abstrakte metoder

## grensesnitt (interface)

- Har bare
  - konstanter (static final - eks static final int i = 4;)
  - metodenavn med parametre, men ikke kode
- Bruker 'interface' i steden for 'class' før navnet
- Gir en 'type' som andre må implementere
  - class ABC implements ButikkSystem
    {....her gies kode for alle metodene i 'ButikkSystem'...}
- Meget nyttig, brukes mye ved distribuerte systemer



## indre (lokale) klasser i en klasse

- Er lovlig (og nyttig)
- Brukes mye ved vindus-programmering

- Gjør det mulig å skjule en delmodell inne i en klasse

## 7. Gjenbruk/subklasser (kap. 6)

- Subklasser, enkel arv, men kan implementere flere grensesnitt
  - class A extends B {....;}
  - class A extends B  implements C,D,...,E {....;}
  - class A implements C,D {....;}
  - osv...
- this og super
- Konstruktøren
  - Default - eller egen
  - Kallrekkefølge ved subklasser - super først.
  - Super-konstruktøren  må kalles fra første linje i sub-klassens konstruktør hvis vi ikke her egen konstruktør - bruker da default kunstruktør (som er uten parameter)



## 8. Arrayer og 'mengder'

- Array - raskt og fast lengde
- Vektor - fleksibel men langsommere, tar imot alt man putter i den ('ingen' øvre grense på lengden)
  - Vector platesamling = new Vector( );
  - addElement(), elementAt(i), size(),.. operasjoner
- Hashtable
  - Brukes i in105 - mengde med søkenøkkel
- Ulike typer ansamlinger av objekter (collections)
  - lister, ordnete lister
  - Etter en nøkkel
  - eks: stack
- Alt kan gjøres til en string
  - Alle objekter inneholder metoden toString( ), nyttig ved testing

```
System.out.println("her er xx:" + xx.toString( ))
```

- Iteratorer

## class Hashtable

**This example creates a hashtable of numbers. It uses the names of the numbers as keys:**

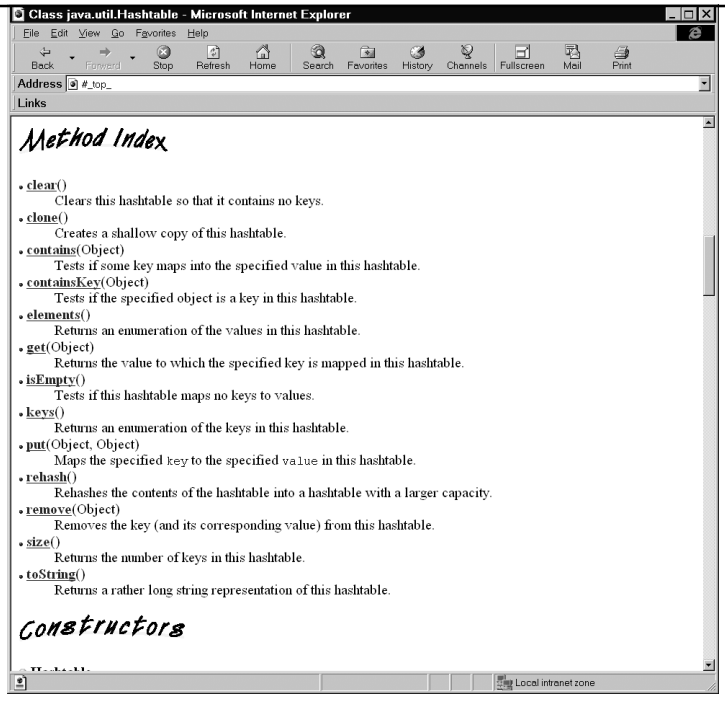
```
Hashtable numbers = new Hashtable();
numbers.put("one", new Integer(1));
numbers.put("two", new Integer(2));
numbers.put("three", new Integer(3));
```

**To retrieve a number, use the following code:**

```
Integer n = (Integer)numbers.get("two");
    if (n != null) {
        System.out.println("two = " + n);
    }
```

©Arne Maus, jan. 1999





# 9. Feil-håndtering

Programmet skal selv bli kalt ved feil-situasjoner:

- Exception
  - et objekt med feilmeldingen
- throw
  - verb for å gi feilmelding - metoden man er inne i, avsluttes
- throws
  - deklarasjon av hvilke feilmeldinger en metode kan kaste
- catch
  - Fang opp feilmelding
- try
  - utfør kode som kan gi feilmelding
- RunTimeException (divisjon med null, null-peker ,...)
  - så vanlig at man ikke behøver 'fange' dem (programmet terminerer hvis vi ikke fanger dem)

©Arne Maus, jan. 1999


# Generelt

- To strategier:
  - a) Fang opp feilobjektet i koden selv (try)
  - b) kast feil-objektet videre (throws) som Text i JavaGently
- Kan enten fange hver type av feilmelding (tester da på typen i catch og har flere catch-setninger) , eller alle uansett type (Exception) - se også eksempel under IO

```
try {
      // kode som kan gi feil
      // f.eks kalles en metode som sier at
      // den kaster visse feilmeldinger
    }
   catch (Exception e){
     // kode som behandler feilsituasjonen, f.eks:
     e.printStackTrace( );
    }
```

©Arne Maus, jan. 1999

## 10. I/O (filer)

- Java 1.0 (gammelt) - vanskelig
- Java 1.1 - IO Streams - (filer,hukommelsen, pipes) bedre:
  - Alt betraktes som strømmer. Klasser:
    - Reader, Writer
    - FileReader, FileWriter
    - StringReader, StringWriter
    - CharArrayReader, CharArrayWriter
    - PipedReader, PipedWriter
    - \+ Modifiserende klasser som LineNumberReader, BufferedReader,..for å gi tilleggsfunksjonalitet..
  - Disse kombineres (legger rundt basisklassene) for å gi kjente IO-modeller som
    - Linjevis lesing og skriving
    - Bufret I/O, .. osv
  - for seg selv: RandomAccessFile



## standard input ('vanskelig')

```
BufferedReader stdin =
  new BufferedReader(new FileReader(System.in));

  int i;
  String s = new String();

  System.out.print("Skriv et tall:")
  s = stdin.readLine();
  i = Integer.parseInt(s);

  System.out.println(s+", tolket som tall" + i);
```

## Eks lese inn fil linjevis buffret

```
import java.io.*;

..public static void main(String[] args) {

try{
   BufferedReader in = new BufferedReader(
                    new FileReader(args[0]));
   String s, s2 = new String();

   while ((s = in.readLine()) != null)
     s2 += s + "\n";
   in.close();
} catch (FileNotFoundException e) {
     System.out.println("File not found: " + args[0])
}  catch(IOException e) {
    System.out.println("IO Exception");
}}
```



## class Text i JavaGently: enkel fil/skjerm/tastatur inn/ut

```
public class Text {
      /*public static void    prompt (String s)
      * public static int      readInt (BufferedReader in)
      * public static double readDouble (BufferedReader in)
      * public static String  readString (BufferedReader in)
      * public static char     readChar (BufferedReader in)
      * public static String  writeInt (int number, int align)
      * public static String  writeDouble  (double number, int align, int frac)
      * public static BufferedReader   open (InputStream in)
      * public static BufferedReader   open (String filename)
      * public static PrintWriter          create (String filename) */
   }
```

Les fra tastatur:

```
import java.io.*;
import javagently.*;

class HighestValue {
 public static void main(String[] args) throws IOException {

  BufferedReader in = Text.open(System.in);

 Text.prompt("How many numbers (1 or more)?");
  int n = Text.readInt(in);

  System.out.println("Type them in");
  Text.prompt("1>");
  int highest = Text.readInt(in);

 // Read and check the rest of the numbers
  int number;
  for (int i = 2; i <= n; i++) {
   Text.prompt(i+">");
   number = Text.readInt(in);
   if (number > highest)   highest = number;
  }
  System.out.println("That's enough, thanks");
  System.out.println("The highest number was "+ highest);
 }
}
```

Les fra fil og tastatur:

```
import java.io.*;
import javagently.*;

class Summation2 {
  /* Reads in numbers from a file and display their sum.
  * Illustrates the declaration of input from both
  * the keyboard and a file.
  */

 public static void main (String [] args) throws IOException {

  int count;
  double total = 0;
  double number;

  BufferedReader in = Text.open(System.in);
  BufferedReader fin = Text.open("numbers.dat");

  System.out.println("****** Summing numbers from a file ******");

  Text.prompt ("How many numbers?");
  count = Text.readInt(in);

  for (int i = 1; i <= count;  i++) {
    number = Text.readDouble(fin);
    total += number;
  }
  System.out.println("That's enough, thanks.");
  System.out.println("The total is "+total);
 }
}
```

# 13. AWT - vinduer og knapper

- Godt bibliotek med klasser for vinduer, dialoger, trykknapper og alt annet i et GUI
- Det som skjer i programmet er at det ligger 'stille' etter initiering og venter på brukerens handling (musebevegelse, klikk, tastetrykk,..osv)
- Hver slik handling generer en hendelse (event) som pakkes inn som en melding (objekt) fra op.-systemet
- Disse meldingen oversendes programmet i generert rekkefølge

©Arne Maus, jan. 1999


# Event-programmering:

- Event-modellen er meget enkel:
  - Klassene som vil motta slike meldinger sier hvilke typer meldinger de vil ha
    - eks: `implements ActionListener`, og en av metodene klassen da implementerer de(n) metoden(e) som per def mottar meldingen: `actionPerformed(ActionEvent e) {..'her behandles en slik hendelse e.}`
  - De typer av meldinger som ingen klasse vil ha, kastes
  - Hvis flere klasser vil ha en meldingstype får de hver sine.
  - Den metoden som per def. skal ha slik melding kalles i alle de klassene som har sagt fra.

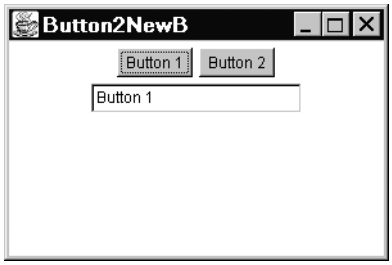
©Arne Maus, jan. 1999

```
//: Button2NewB.java - s.516 Bruce Eckel
// An application and an applet
import java.awt.*;
import java.awt.event.*; // Must add this
import java.applet.*;

public class Button2NewB extends Applet {
  Button
    b1 = new Button("Button 1"),
    b2 = new Button("Button 2");
    TextField t = new TextField(20);

  public void init() {
    b1.addActionListener(new B1());
    b2.addActionListener(new B2());
    add(b1);
    add(b2);
    add(t);
  }
  class B1 implements ActionListener {
    public void actionPerformed(ActionEvent e) {
      t.setText("Button 1");
    }
  }
```

```
class B2 implements ActionListener {
  public void actionPerformed(ActionEvent e) {
    t.setText("Button 2");
  }
}
// To close the application:
static class WL extends WindowAdapter {
  public void windowClosing(WindowEvent e) {
    System.exit(0);
  }
}
// A main() for the application:
public static void main(String[] args) {
  Button2NewB applet = new Button2NewB();
  Frame aFrame = new Frame("Button2NewB");
  aFrame.addWindowListener(new WL());
  aFrame.add(applet, BorderLayout.CENTER);
  aFrame.setSize(300,200);
  applet.init();
  applet.start();
  aFrame.setVisible(true);
}
///:~
```



# 14. Tråder og parallellitet

- Tråder (threads) gir separate programeksekveringer inne i ett program, men felles adresseområde
- Hver tråd sprer seg evt. på hver sin CPU (hvis flerCPU)
- Stopper den ene, forsetter den andre problemfritt
- Vilkårlig hastighet/framdrift i forhold hverandre
- Tråder kan synkroniseres
  - Når tråder trenger å lese/skrive felles data - kan det skje ordnet (ikke samtidig, men køes opp)
  - synchronized metoder

©Arne Maus, jan. 1999


```
//: SimpleThread.java. litt endret Bruce Eckel, 1998

public class SimpleThread extends Thread{
  private int countDown = 5;
  private int threadNumber;
  private static int threadCount = 0;
  public SimpleThread() {
    threadNumber = ++threadCount;
    System.out.println("Making " + threadNumber);
  }
  public void run() {
    while(true) {
      try {System.out.println("Thread " +
            threadNumber + "(" + countDown + ")");
             sleep(50); // sov 50 millisek.
            if(--countDown == 0) return;
       } catch (InterruptedException e){
           System.out.println("Exception" + e);
       }
    }
  }
  public static void main(String[] args) {
    for(int i = 0; i < 5; i++)
      new SimpleThread().start();
    System.out.println("All Threads Started");
  }
} ///:~
```

**Kjøring:**

```
Making 1
Making 2
Making 3
Making 4
Making 5
All Threads Started
Thread 1(5)
Thread 2(5)
Thread 3(5)
Thread 4(5)
Thread 5(5)
Thread 1(4)
Thread 2(4)
Thread 3(4)
Thread 4(4)
Thread 5(4)
Thread 1(3)
Thread 2(3)
Thread 3(3)
Thread 5(3)
Thread 4(3)
Thread 2(2)
Thread 1(2)
Thread 3(2)
Thread 5(2)
Thread 4(2)
Thread 2(1)
Thread 3(1)
Thread 5(1)
Thread 1(1)
Thread 4(1)
```