Programmering i Java, 5p

Marie Nordström (marie@cs.umu.se)
Jürgen Börstler (jubo@cs.umu.se)
Thomas Johansson (thomasj@cs.umu.se)
} Lärare

Daniel Lundmark (lundmark@cs.umu.se)
Set Norman (set@cs.umu.se)
Johan Tordsson (tordsson@cs.umu.se)
} Handledare

http://www.cs.umu.se/kurser/TDBA54/SU02

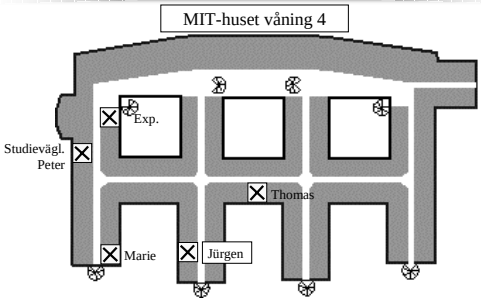
Institutionen för datavetenskap

- 2 + 2 utbildningsprogram
  - Datavetenskap
  - Teknisk datavetenskap
  - Kognitionsvetenskap
  - Interaktion och design
- C:a 80 anställda (5 professorer)
- Forskning inom många olika ämnen
  - Teknisk-vetenskapliga o parallella beräkningar
  - Medicinsk informatik
  - Kognitionsvetenskap
  - Programvaruteknik
  - Teoretisk datalogi
  - MDI
  - ...
- Flera forskningscentra
  - HPC2N
  - VR-Lab
  - Interactive Institute

Kursen kontaktpersoner

- Studentexpeditionen
  - Anne-Lie Persson (anne-lie, 6130)
  - Yvonne Löwstedt (yvonne, 5225)
- Studievägledare
  - Peter Stenberg (peter, 6985)
- Support
  - Magdalena Kågström (support, 9950)
- Kursens lärare
  - Marie Nordström (marie, 7708)
  - Claes Gahlin (claes, 7773)
  - Jürgen Börstler (jubo, 6735)
  - Thomas Johansson (thomasj, 6259)

Här sitter vi



Datorlab

- MIThuset plan 3
- MC323, MC333 och MC343
  - alltid tillgängliga via kort
- Handledning (Daniel, Set & Johan)
  - På angivna tider
  - I labben

Kursens mål

- Kunskaper om problemlösning och programmering
- Färdighet i datoranvändning och programmering i Java
- Kännedom om begrepp, metoder och hjälpmedel för programkonstruktion

Kursens uppläggning

- Föreläsningar - F
- Lektioner i lab - LL
- Metodikövningar - CRC
- Individuell handledning
- Examination
  - ”0”+3 obligatoriska uppgifter = 2 p
  - Teoriprov (3 timme)
  - Praktikprov i lab (4 timmar) = 3 p

Kursmaterial

- Kursbok :Koffman B. Elliot & Wolz Ursula
  Problemsolving with Java, 2nd ed.
  Addison Wesley 2002
  ISBN: 0-201-72214-3
  - OBS! Vi följer inte bokens upplägg till punkt och pricka

- OH bilderna från föreläsningen (finns på websidan)
- Diverse material som kommer att delas ut så småningom

Hemsidan:
`http://www.cs.umu.se/kurser/TDBA54/SU02`

Innehåll

- OOP snabbintroduktion
- Programvaruutveckling och programmering
- Datatyper
- Uttryck
- Satser
- Arv, polymorfi och dynamisk bindning
- Design (CRC)
- Metodik och klassdesign
- Att organisera Javakod
- Fält
- Undantag
- In-/utmatning och filer
- Grafik
- GUI:s
- Applets vs applikationer
- Rekursion
- Interfaces och sortering
- Noggrannheten i beräkningar

Objekt

Ett **objekt** är en individuellt identifierbar entitet som kan vara konkret eller abstrakt. Ett objekt har tillstånd, beteende och identitet.

- Reellt, gripbart, synligt ting (t ex en specifik person)
- Abstrakt ting (t ex en tid eller en anställd)

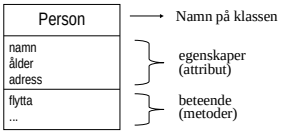
Varje **objekt** har ett tillstånd, ett beteende och en identitet.

- Tillståndet beskriver objektets egenskaper (t ex adress och ålder hos en person)
- Beteendet beskriver vad objektet kan göra (t ex flytta). OBS! Detta kan innebära att tillståndet ändras
- Identiteten skiljer ett objekt från alla andra objekt

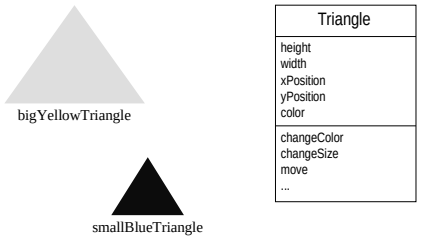
Klass

En **klass** är en “byggplan” för objekt av samma sort.

- Alla objekt av en klass (instanser) har samma sorts egenskaper och beteendet
- En klass beskriver en mängd liknande objekt



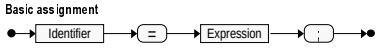
Objekt vs klass



Programspråk

- Ett programspråk specificerar
  - ord och symboler som får användas för att skriva program
  - regler som bestämmer hur ord och symboler får sättas ihop

➨ ***Syntax***en beskriver hur giltiga programstrukturer (***satser***) måste se ut (se utdelat Appendix L ur Lewis Loftus)

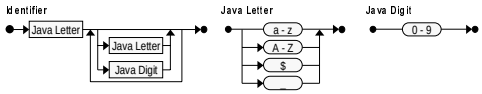


➨ ***Semantik***en beskriver vad precis olika satser betyder (dvs hur datorn ska tolka programmen)

Värdet av uttrycket i *expression* tilldelas ("sparas i") variablen *identifier*.

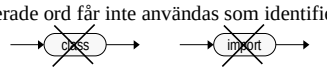
Identifierare och variabler

- Identifierare är namn på olika storheter som definieras av programmeraren
- Identifierare får innehålla bokstäver, siffror, understrykningstecknet (_), och valuta tecknen (t ex dollar tecknet)
- Identifierare får **ej** inledas med en siffra
- Identifierare måste vara entydiga
- Java är ***case sensitive***, dvs `Total` är olika `total`
- Identifierare för värden eller objekt kallas ***variabler***



Reserverade ord

- Reserverade ord får inte användas som identifierare



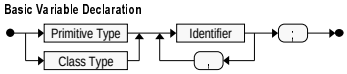
- Exempel på reserverade ord (det finns 59 sty):

```
abstract   extends    if          private     this
boolean    false      implements  public      true
class      float      import      return      void
const      for        int         static      while
```

- Reserverade ord brukar visas i en annan färg i editorn

Variabler och datatyper

- Variabler används för att spara data
- Variabler måste ***deklareras*** med datatyp och namn innan de får användas

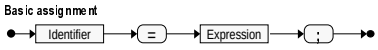


- En primitiv datatyp kan vara heltal (***byte***, ***short***, ***int***, ***long***), flyttal (***float***, ***double***), tecken (***char***) eller ***boolean***
- En klasstyp är ett namn på en klass (t ex `Triangle`)
- Primitiva datatyper kallas också inbyggda datatyper
- Klasstyper kallas också användardefinierade datatyper

Tilldelning av värden

- En variabel innehåller antingen ett ***primitivt värde*** eller en ***referens*** till ett objekt
- Värdet av en variabel efter deklarationen är odefinierad
- Referenser som inte "pekar" till ett objekt har värdet `null`
- Variabler måste ***initialiseras***

- En variabel tilldelas ett (nytt) värde genom =



Exempel

```
int value, discount;
value = 123;
discount = -15;
```

```
float net_price;
net_price = 123.0;
net_price = 1.23e2;
```

```
char aCharacter;
aCharacter = 'A';
aCharacter = '\n';
```

```
boolean continue = true;
```

Initialisering är tillåten i samband deklarationen!

```
Triangle bigYellowTriangle, smallBlueTriangle;
bigYellowTriangle = ???
```

Hur "skaffas" objekt?

Skapa och manipulera objekt

- Varje klass har (minst) en ***konstruktor*** för skapa (nya) objekt (s k ***instanser***) av denna klass
- Variabeln tilldelas en referens till objektet
- Konstruktorn används genom ***new*** operatorn

```
Triangle bigYellowTriangle = new Triangle();
```

- Objektet manipuleras med hjälp av objektets metoder
- Kom ihåg: Metoderna definieras i klassen

Klassnamn
Attribut (egenskaper)
Metoder (beteendet)

Referenser vs värden

- Tilldelning har lite olika effekt för primitiva datatyper och klasstyper!
  - Primitiva datatyper: Variabel ≈ själva värdet
  - Klasstyper: Variabel = referens till objektet

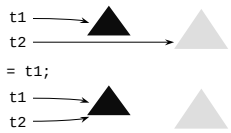
```
int i1 = 5, i2 = 10;
```

i1: 5 i2: 10

```
i2 = i1;
```

i1: 5 i2: 5

```
Triangle t1 = new Triangle();
Triangle t2 = new Triangle();
```




```
t2 = t1;
```


Alias och garbage collection

```
Triangle t1 = new Triangle();
Triangle t2 = new Triangle();
```

Ett objekt med flera namn; namnen kallas ***alias***

```
t2 = t1;
```

Objekt som inte längre kan nås; rensas bort automatiskt (***garbage collection***)

```
t2.changeColor (”red”);
```

”Båda” trianglar (`t1` och `t2`) har blivit röda
➨ Var försiktig med aliasing

Instantiering och manipulation

Triangle
height width xPosition yPosition color
changeColor changeSize move ...

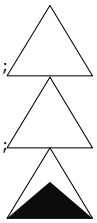
```
Triangle aTriangle = new Triangle();

aTriangle.move();

aTriangle.changeColor(”blue”);

aTriangle.move();

aTriangle.changeSize(50, 100);
```



En vanlig metodanrop ser ut så här

```
<objektnamn>.<metodnamn> (<parameterlista>);
```


`Triangle` i detalj 1

```
public class Triangle
{
   private int height;
   private int width;
   private int xPosition;
   private int yPosition;
   private String color;
```

Att göra attribut oåtkomliga är bra programmeringsstil

```
   /**
    * Create a new triangle at default position with default color.
    */
   public Triangle()
   {
      height = 30;
      width = 40;
      xPosition = 50;
      yPosition = 15;
      color = "green";
      draw();
   }
   ...
```

Kommentar

Konstruktorn

`Triangle` i detalj 2

```
   /**
    * Move the square a few pixels down.
    */
   public void move()
   {
      erase();
      yPosition = yPosition + 100;
      draw();
   }
```

Kommentar

```
// clear old position
// determine new position
// draw on new position
```

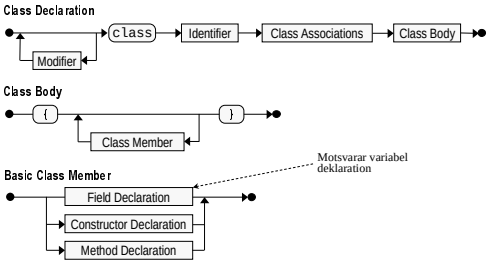
En metod

```
   /**
    * Change the size to the new size (in pixels).
    * PRE: Size must be >= 0.
    */
   public void changeSize (int newHeight, int newWidth)
   {
      erase();
      height = newHeight;
      width = newWidth;
      draw();
   }
   ...
```

Parameter (formell)

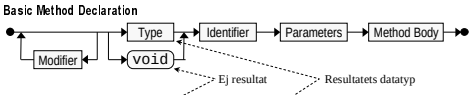
Klassdeklaration

- Syntax:



Metoddeklaration

- Syntax:



- Exempel:

```
public void changeSize (int newHeight, int newWidth)
{
   …
}

public float getArea ( ) {…}
```

Ej resultat
Resultatets datatyp
Parameter

Parameter

- En metod kan ta noll eller flera parameter
- Varje parameter specificeras med datatyp och namn, de s k ***formella parametrar***
- Antal, datatyperna och ordningen av de formella parametrar, metodnamnet och resultattyp kallas en metods ***signatur***
- De s k ***aktuella parametrar*** i metodanropet måste vara kompatibelt med metodens signatur
- Exempel:

```
aTriangle.changeSize (100, 50);  // OK
aTriangle.changeSize (100);      // fel antal
aTriangle.changeSize ('1', 50);  // fel datatyp
char c = aTriangle.getArea ();   // fel datatyp
```


Parameteröverföring

- I Java överförs alla parameter ***by value***
  - Värdet på den aktuella parametern kopieras till den formella parametern (motsvarar tilldelningssats)
  - När en referens överförs blir den formella parametern ett alias för aktuell parameter
- `Return`-satsen anger vilket värde som returneras (dvs överförs tillbaka)
  - Överförs också by value

Metodkropp

- Innehåller lokala deklarationer & exekverbara satser
- Lokala deklarationerna gäller bara internt
- De formella parametrar betraktas som lokala deklarationer
- Om resultattyp är inte `void` måste minst en `return`-sats finnas
- Exempel:

```
public float getArea ()
{
   float temp = 0.0;
   temp = height * width / 2;
   return temp;
}
```

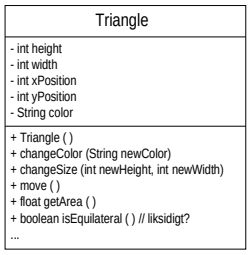
Lokal variabel

Konstruktordeklaration

- En konstruktor är en speciell metod
  - Samma namn som klassen
  - Ingen explicit resultattyp
  - Används för att initialisera objektets attribut
  - Får ha parameter, t ex för att påverka initialiseringen
- Exempel:

```
public Triangle () {…}
```

Grafisk notation



Abstraktion, information hiding och inkapsling

- En ***abstraktion*** är en förenklad men trogen modell av nånting som kan vara väldigt komplicerat
- En objekt/klass är abstrakt i den meningen att vi inte behöver känner till alla detaljer för att kunna använda det
  - Vi behöver t ex inte känna till koden till `move`
- Med ***information hiding*** menas att viss information görs åtkomlig för användaren
  - T ex för att säkerställa att `xPosition` och `yPosition` bara förändras på ett kontrollerad sätt (genom `move`)
- Med ***inkapsling*** menas att abstraktion och information hiding används för att definierar meningsfulla klasser
  - Abstraktionen görs offentlig (`public`)
  - Känslig och oväsentlig data görs oåtkomlig (`private`)

Programmeringsfel

En dator gör inte det programmeraren menade, den gör precis det ett programs semantik föreskriver.

- Man kan skiljer tre typer av fel
  - ***Kompileringsfel***
    När kompilatorn upptäcker att syntaxreglerna ej har följts eller att vissa typer ej är kompatibla (t ex `int i = 'c';`)
  - ***Run-time fel***
    När programmet "kraschar" när det körs, t ex pga division med noll eller att typer ej är kompatibla
  - ***Logiska fel***
    När programmet verkar fungera bra, men levererar fel resultat

Innehåll

- OOP snabbintroduktion
- Programvaruutveckling och programmering
- Datatyper
- Uttryck
- Satser
- Arv, polymorfi och dynamisk bindning
- Design (CRC)
- Metodik och klassdesign
- Att organisera Javakod
- Fält
- Undantag
- In-/utmatning och filer
- Grafik
- GUI:s
- Applets vs applikationer
- Rekursion
- Interfaces och sortering
- Noggrannheten i beräkningar

Programvaruutveckling

- Vitsen med att skriva ett program är att lösa ett problem

- Typiska steg för att lösa ett problem:
  - Första problemet
  - Dela in problemet i hanterliga delar
  - Skissa och konstruera en lösning
  - Fundera på alternativa lösningar
  - Förfina lösningen
  - Realisera lösningen
  - Testa lösningen och åtgärda alla fel

Programmering

Programmering är att formulera en lösning till ett problem i ett programspråk.

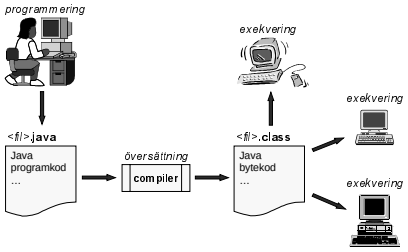
- Känt problem
- Känd lösning
- Känt programspråk

Programmering är inte ett sätt att komma fram till en lösning.

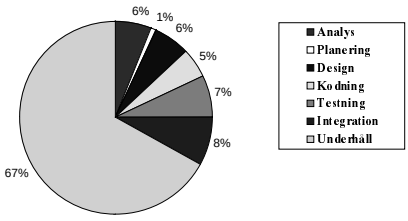
Problemlösning m h a datorer

0. Lös problemet
1. Skriv ett program som löser problemet
2. Mata in programmet i datorn
3. Exekvera programmet
4. Läs av resultatet

Programmering i Java



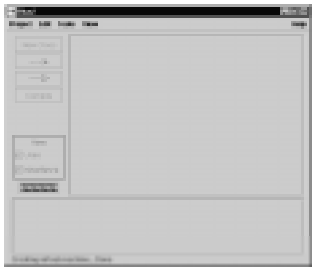
Kodning är bara en liten del i programvaruutvecklingen



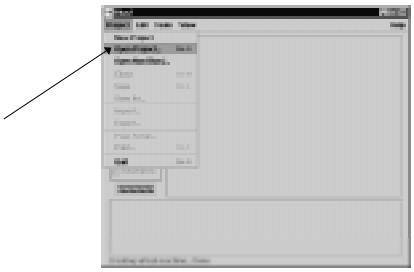
Problemlösning igen

- Många projekt misslyckas, eftersom utvecklarna inte till fullo förstår problemet som ska lösas
  - ➨ Alla antaganden måste klargöras
  - ➨ Möjliga feltolkningar måste undanröjas
- När problem blir större, måste lösningen delas in i hanterliga delar
- Denna teknik är fundamental för programvaruutveckling
- I ***objektorienterad utveckling*** delas lösningen in i så kallade ***objekt*** och ***klasser***
  - ➨ Java är ett objektorienterad språk

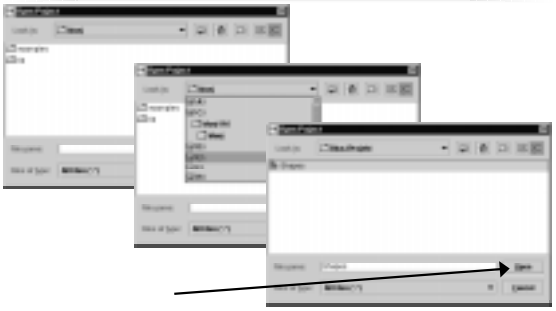
Vårt verktyg BlueJ



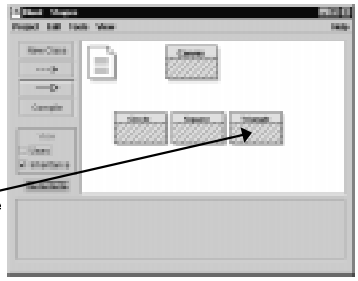
Öppna ett existerande projekt



”Browsa” till rätt projekt…

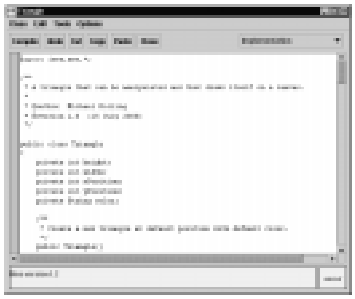


Shapes

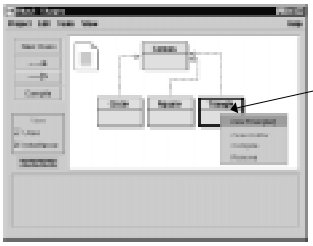


Dubbelklicka för att se källkoden till `Triangle`

Källkod i separat fönster



Skapa objekt i BlueJ

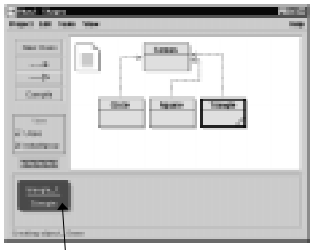


Högerklicka för att skapa en `Triangle`

Välj konstruktorn

Välj namn på objektet - ok

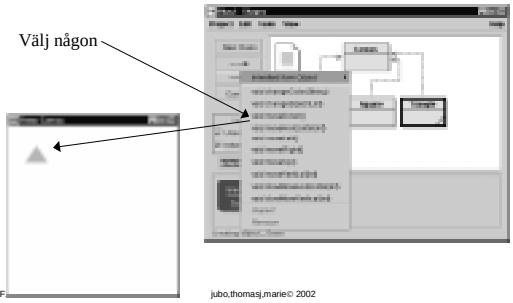
Manipulera triangeln



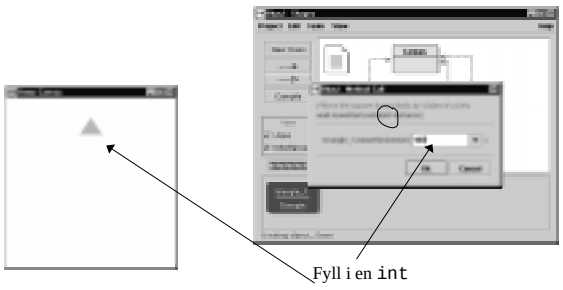
Högerklicka för att se metoderna

Skapa objekt i BlueJ

Välj någon



Parametrar i BlueJ



Flera objekt av samma klass