Objektorienterad programmering i Java

Föreläsning 1

Kapitel 1-3 i kursboken Java Software Solutions

Läsanvisningar

- Kapitel 1 är en repetition av det ni förväntas kunna sedan tidigare. Det är dock en bra ide att läsa igenom kapitlet för att komma in i de begrepp som författarna använder
- Kapitel 2 ger några inledande exempel på Javaprogram. I kapitel 3 diskuteras en del grundläggande koncept inom programmering. I detta kapitel finns fler programexempel. Skriv gärna av exemplen i kapitel 2 och 3 och provkör dem själv. Detta ökar ofta förståelsen för vad programmen gör. Bli inte bekymrad om du inte förstår alla delar av programmen just nu. Det kommer att klarna längre fram.
- Se till att du har gått igenom laboration noll, dvs. den inledande laborationen där du lär dig att installera Javamiljön och köra ett litet Javaprogram.
- Föreläsning 2 är delvis annorlunda uppbyggd än denna föreläsning (föreläsning 1). I föreläsning 2 studerar vi ett exempel och vi följer därför inte kursbokens uppläggning riktigt så strikt som i denna föreläsning. Det finns även en viss överlappning mellan föreläsning 1 och 2.

Vad är ett datorsystem

Ett datorsystem består av:

- Hårdvara
  - De delar som datorn består av, dvs det man kan ta på
- Mjukvara
  - De program som kör på datorn

Hårdvara

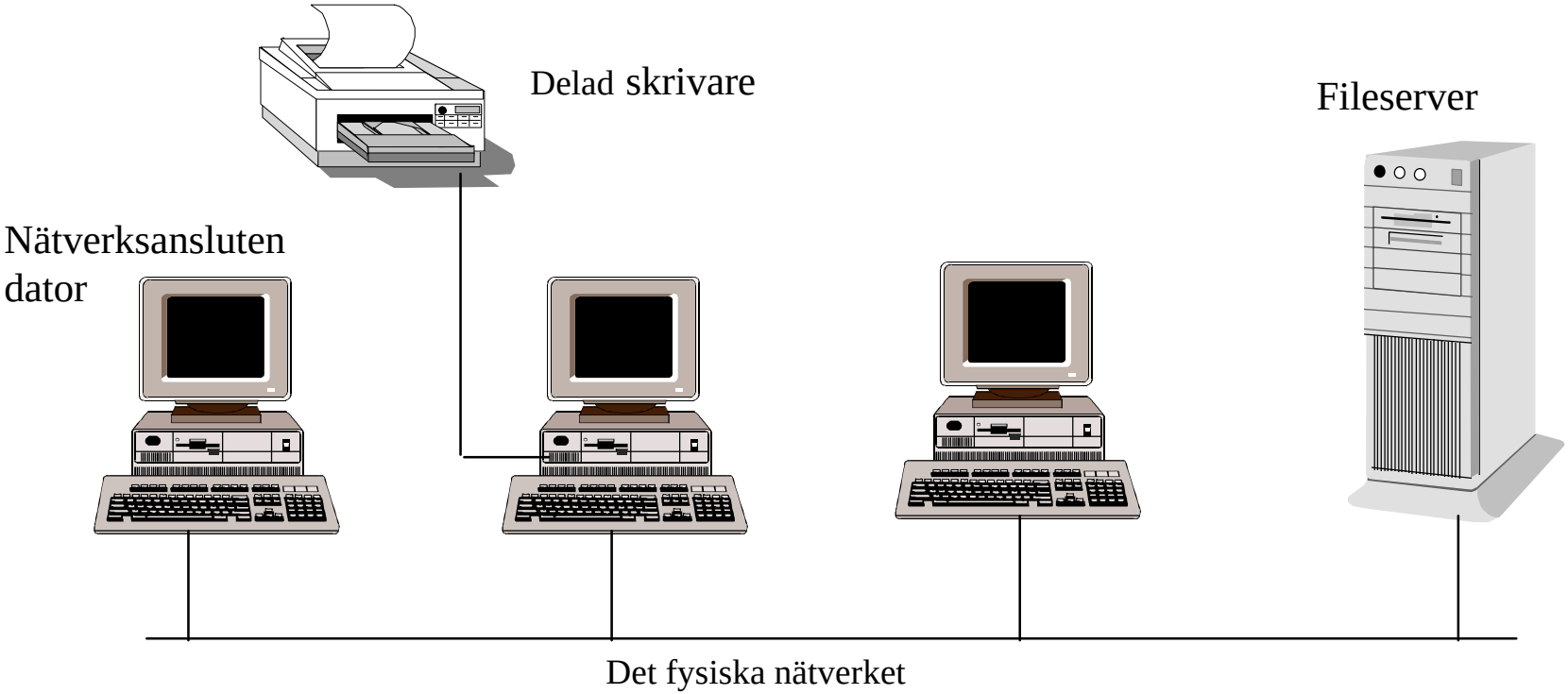
- CPU (Processorn, t.ex. Pentium Pro)
- Internminne (kan t.ex. vara 32 Mbyte)
- Hårddisk (kan t.ex. vara 2 Gbyte)
- I/O-enheter (t.ex. tangentbord och skärm)
- etc.

Ovanstående är hopkopplat med hjälp av systembussen

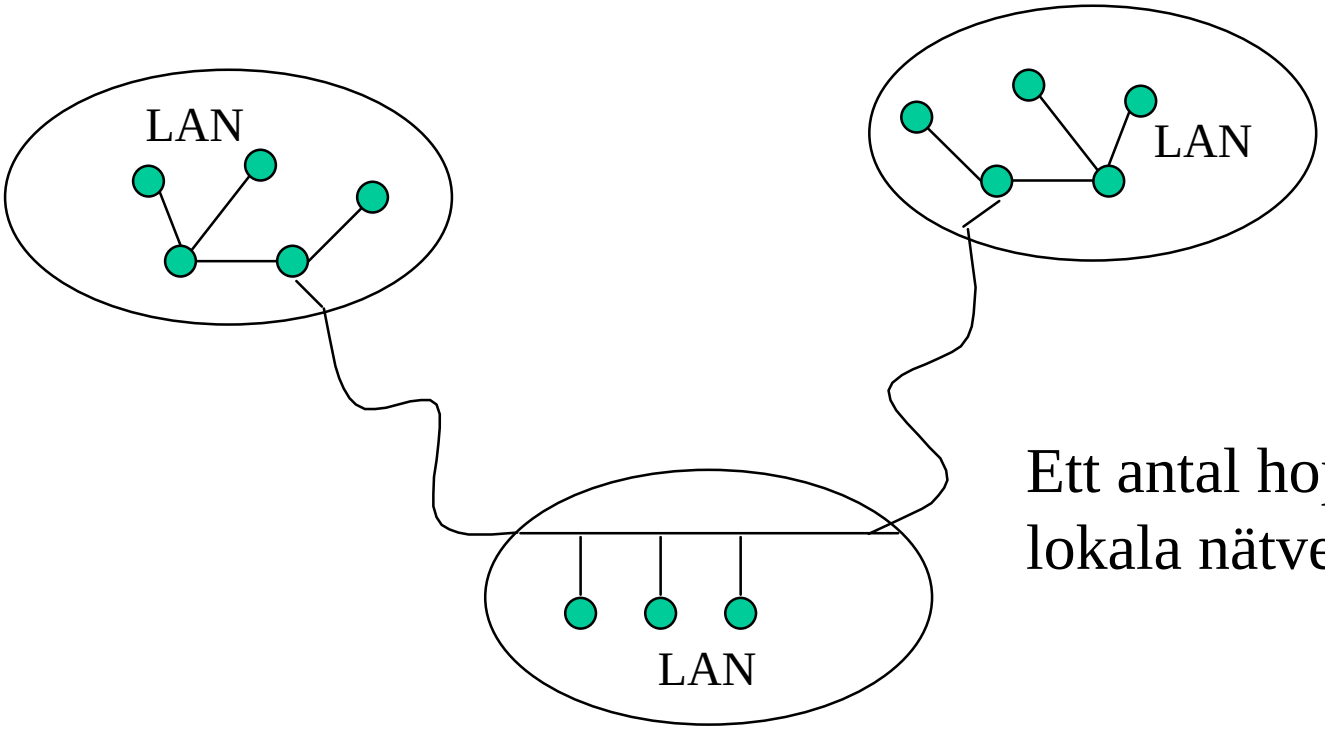
Mjukvara

- Operativsystemet (t.ex Windows-95 och Windows NT)
  - Hanterar användargränssnittet
  - Hanterar datorns hårdvaruresurser, t.ex. lagring av filer på hårddisken
- Applikationsprogram (t.ex. ordbehandlare, spel och de program användaren själv utvecklar)

Lokalt nätverk (LAN)



Wide-Area Network (WAN)



Ett antal hopkopplade lokala nätverk

Vilka steg går man igenom när man utvecklar program

- Analys: Analyserar det problem man vill lösa, dvs. ange vad programmet ska göra
- Design: Ange hur programmet ska bete sig för att resultatet ska bli det man vill
- Implementation: Skriva ner lösningen i programkod, t.ex. Javakod
- Test: Köra programmet för att undersöka om det fungerar som det ska. Om det inte fungerar på ett tillfredsställande sätt måste man felsöka i programmet, dvs. fundera ut vad som är fel, ändra i programkoden och sedan testa på nytt. Det är mycket sällan som programmen fungerar som de ska när man testar dem första gången.
- Underhåll: Även när man har har ett fungerande program kan man behöva göra förändringar i det. Man kallar detta för att man underhåller programmet. Det kan t.ex. röra sig om modifieringar och utbyggnader av programmet. Underhållsfasen sträcker sig under hela den tid som programmet används. Själva underhållsarbetet utförs ofta av någon annan än den eller de som utvecklad programmet från början.

Vad är ett programspråk

- Ett språk i vilket man berätta för datorn vad den ska göra
- Man delar ofta in programspråken i fyra grupper:
  - Maskinkod, den kombination av ettor och nollor som datorn förstår
  - Assemblerkod, varje kombination av ettor och nollor motsvaras av ett ord (assemblerinstruktioner)
  - Högnivåspråk, varje rad motsvarar ett stort antal assemblerinstruktioner, t.ex. Java, C, C++, Ada, …
  - Fjärde generationens språk (4GL), kraftfulla språk som är anpassade till ett speciellt applikationsområde oftast olika typer av databasapplikationer

Hur gör man när man skriver själva programmet (programmeringscykeln)

- Man skriver in programkoden (källkoden) med hjälp av en texteditor
- Man sparar koden i en fil. När man arbetar med Java ska filen alltid ha suffixet java, dvs. filnamn.java
- Man kompilerar (översätter) filen till ett format som datorn kan förstå. När man arbetar med Java översätts källkoden till ett format som kallas bytekod. Bytekoden interpreteras (tolkas) sedan av datorn (alternativt kompileras till maskinkod som körs på datorn).
- Man testar programmet för att se om det fungerar som man hade tänkt sig. Om det inte fungerar som man hade tänkt sig letar man upp felet, ändrar i källkoden och testar igen.

Olika typer av fel

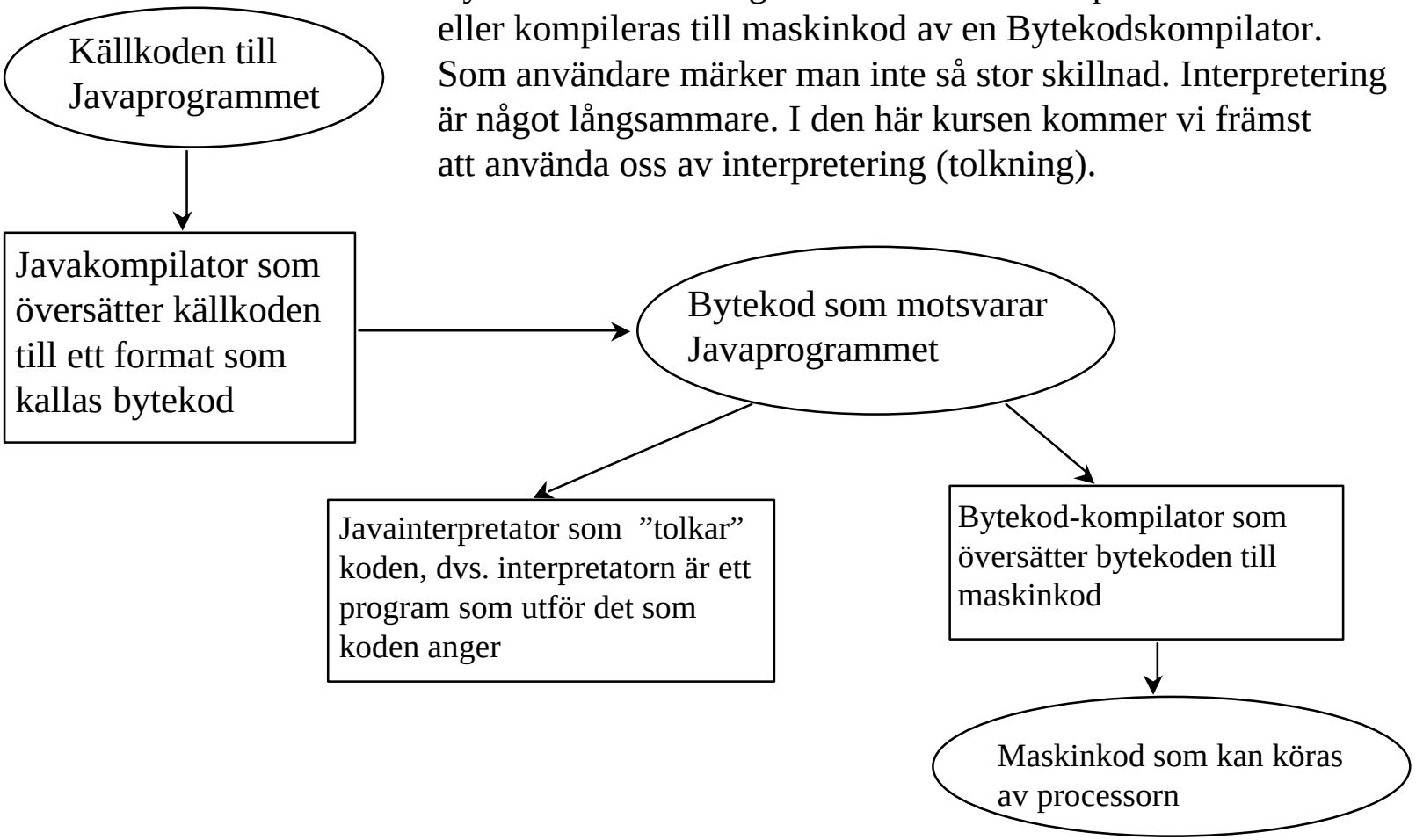
- Kompileringsfel
  - Dessa fel uppstår redan när man försöker att kompilera programmet, dvs. Javakompilatorn skriver ut ett felmeddelande. Ett vanligt exempel är att man han stavat fel på något ställe i källkoden eller glömt ett semikolon.
  - Exekveringsfel
  - Dessa fel uppstår när man exekverar (kör) programmet, dvs man får ett felmeddelande från Javainterpretatorn. Ett vanligt exempel är att man försöker dividera ett tal med noll (någonting delat med noll ger ett odefinierat resultat)
  - Logiska fel
  - Dessa fel visar sig genom att programmet inte gör det man hade tänkt, dvs. programmet kör utan felmeddelanden men resultatet blir inte det avsedda. Den här typen av fel är de som är svårast att hitta.

Ett enkelt Javaprogram (källkod)

```
// Ett litet program som skriver ut texten: Java är skoj!
public class JavaFun {
    public static void main(String[] args) {
        System.out.println("Java är skoj!");
    } // här slutar metoden main
} // här slutar klassen JavaFun
```

Översättning och exekvering i Java

Bytekoden kan antingen tolkas av Javainterpretatorn eller kompileras till maskinkod av en Bytekodskompilator. Som användare märker man inte så stor skillnad. Interpretering är något långsammare. I den här kursen kommer vi främst att använda oss av interpretering (tolkning).



Hur gör man för att köra programmet JavaFun

- Skriv källkoden i en texteditor
- Spara textfilen under namnet JavaFun.java
- Kompilera genom att skriva: **javac JavaFun.java**
- Exekvera (kör) programmet genom att skriva: **java JavaFun**
- Resultat: utskrift av **Java är skoj!** på skärmen

Objekt och klasser (mer om detta i föreläsning 2)

- Ett Javaprogram består av ett antal programobjekt (software objects) som motsvarar specifika objekt i verkligheten (real-world objects).
- Exempel på objekt är: Min bil, pappas bil, Kalles bil. Slagskeppet Bismark, Titanic.
- Varje objekt tillhör en klass.
- Klassen anger vilken typ av objekt det rör sig om.
- Objekten Min bil, pappas bil och Kalles bil kan exempelvis vara av klassen Bil, medan objekten Slagskeppet Bismark och Titanic kan vara av klassen Fartyg.

Klassbibliotek

- Man kan samla ett antal relaterade klasser i ett så kallat klassbibliotek. Dessa kallas för packages (paket) i Java.
- Det finns ett antal klassbibliotek som följer med Javamiljön. Dessa klassbibliotek kallas med ett gemensamt namn för Java API (Java Application Programmer Interface).
- Namnen på alla klassbiblioteken (paketen) i Java API börjar med java., t.ex. java.io, java.net och java.math.
- Klassbiblioteken i Java API innehåller en rad användbara metoder (rutiner). T.ex. metoder för att:
  - hantera in- och utmatning till och från programmet
  - kommunicera via olika nätverk
  - hantera olika matematiska operationer, t.ex. Sinus och Cosinus.

(API =Application Programmers Interface)

Java Applets och Java Applications

- Det finns två typer av Javaprogram:
  - Java Applets
  - Java Applications
- De Javaprogram som man hittar på olika hemsidor på Internet är nästan alltid Applets. Bytekoden till Applets-program byggs in i HTML-dokument och körs med hjälp av Webbläddraren (Netscape eller MS Explorer), dvs. Webbläddraren sköter interpreteringen (eller kompileringen) av bytekoden.
- Java Applications fungerar som mer traditionella datorprogram, och de körs med hjälp av den vanliga interpretatorn (som heter java).
- Skillnaden mellan att skriva en Application och en Applet är relativt liten och för att inte röra till det i onödan kommer vi att koncentrera oss på Applications (i alla fall i början av kursen).
- Laboration 1-3 kommer att handla om Java Applications och laboration 4 handlar om Applets.

Datatyper och variabler sid 76- (60-)

De data som ett program arbetar med kan vara av olika typ, t.ex.

- Heltal
- Flyttal (decimaltal)
- Textsträngar

I programmet lagrar man data i så kallade variabler. I ett program som ska räkna ut medelvärdet av fem heltal, kan man exempelvis ha en variabel för vart och ett av de fem talen och en variabel för själva medelvärdet. Varje variabel är av en viss datatyp. Datatypen för de fem talen är heltal (integer på engelska) och datatypen för medelvärdet (som ju kan bli ett decimaltal) är flyttal (float på engelska). Om man ska läsa in en textsträng till programmet så kan man lagra denna i en variabel av datatypen textsträng (String).

Vi återkommer till variabler och datatyper i föreläsning 2.

Primitiva datatyper (64-)

- Vissa datatyper kan man själv definiera i sitt program. Detta gör man genom att definiera klasser. Variabler av av en klass är samma sak som objekt av klassen.
- Det finns emellertid ett antal datatyper som är inbyggda i språket. Dessa datatyper kallas för primitiva datatyper.
- Några exempel på primitiva datatyper är:
  - Heltal (int)
  - Flyttal (float)
  - Tecken (char)
  - Logiska variabler (booleans) som kan ha värdet Sant eller Falskt

Konstanter sid 82 (63-)

Ibland vill man kunna hantera värden som inte ska kunna variera under programmets gång utan hela tiden ha ett konstant värde. I sådana fall är det lämpligt att använda konstanter istället för variabler.

Ordet final före namnet anger att det är en konstant.

Ex: final int summa;

Konstanter används på samma sätt som variabler med den viktiga skillnaden att konstanter inte kan ändra värde under programmets exekvering.

In- och utmatning sid 83- (52- print, println)

- I Java hanteras all in- och utmatning med hjälp av så kallade *streams*
- Man läser data från en *input stream* och skriver ut till en *output stream*
- En stream kan vara kopplad till tangentbord (input), skärm (output) eller disken (input eller output)
- I Java finns tre fördefinierade streams, dvs. tre streams som alltid finns utan att man behöver definiera något själv:
  - **System.in**, för att läsa in data till programmet. Om man inte anger något annat är denna kopplad till tangentbordet
  - **System.out**, för att skriva ut resultat och information från programmet. Om man inte anger något annat är denna kopplad till skärmen.
  - **System.err**, för att skriva ut felmeddelanden från programmet. Om man inte anger något annat är denna kopplad till skärmen (i vissa fall vill man kunna separera felmeddelanden från vanliga utskrifter och då ser man till att System.out och System.err kopplas till olika filer eller motsvarande)

Evalueringsordning i aritmetriska uttryck

sid 89-92 (67-70)

- Exempel på aritmetriska uttryck
  - 2+3*5
  - (2+3)*5
- Aritmetriska uttryck innehåller ett antal operatorer, t.ex. +, -, *, /.
- Vissa operatorer har ”högre prioritet” än andra och de evalueras (räknas ut) före de med lägre prioritet, multiplikation har t.ex. högre prioritet än addition. Detta innebär att uttrycket 2+3*5 räknas ut på följande sätt: 2+3*5 = 2+15 = 17, dvs. multiplikationen räknas ut före additionen.
- Man kan styra evauleringsordningen (dvs. den ordning som uttryckets delar räknas ut) genom att använda parenteser. Exempelvis evalueras uttrycket (2+3)*5 på följande vis: (2+3)*5 = 5*5 = 25, dvs. uttrycket inom parentes evaluerades först.

if-sats sid 92-102 (111-119)

Ibland vill man att ett program ska göra olika saker beroende på vilka värden man matar in till det.

Detta kan man uppnå med hjälp av if-satsen (det engelska ordet ”if” betyder ”om”).
I Java skriver man:

```
if (logiskt uttryck som kan vara sant eller falskt) {
        Dessa satser utförs endast om uttrycket är sant
}
```

T.ex.

```
if (summa > 3) {
   summa = 4;
}
```

I detta exempel jämför vi variabeln summa med 3. Om summa är större än 3 får variabeln summa ett nytt värde, nämligen 4. Om summa är mindre än eller lika med 3 händer ingenting.

Logiska uttryck (Boolean expressions)

sid 95-96 (123-126)

- Ett logiskt uttryck kan ha något av värdena Sant eller Falskt, dvs. uttrycket är antingen sant eller falskt.
- Ett logiskt uttryck består ofta av en eller flera jämförelser.
- I Java finns följande jämförelseoperatorer:

  x == y (två likhetstecken), sant om x är *lika med* y

  x != y, sant om x är *skilt från* y, dvs. sant om x inte är lika med y

  x < y, sant om x är *mindre än* y

  x <= y, sant om x är *mindre än eller lika med* y

  x > y, sant om x är *större än* y

  x >= y, sant om x är *större än eller lika med* y
- Exempel:

  4 >= 5, är falskt

  4 != 5, är sant

If-else-satsen sid 97- (114-)

Detta är en användbar variant av if-satsen (det engelska ordet ”else” betyder ”i annat fall”).
I Java skriver man:

```
if (logiskt uttryck) {
        satser som utförs om uttrycket är sant
}
else {
        satser som utförs om uttrycket är falskt
}
```

T.ex.

```
if (summa > 3) {
        summa = 4;
}
else {
        summa = 2;
}
```

I detta exempel jämför vi variabeln summa med 3. Om summa är större än 3 får variabeln summa ett nytt värde, nämligen 4. Om summa är mindre än eller lika med 3 får variabeln summa värdet 2.

Nästlade if-satser sid 101- (115-)

Man kan ha en if-sats inuti en annan if-sats. Detta kallas för nästlade if-satser, t.ex:

```
if (summa > 3) {
        if (summa > 5) {
                summa = 6;
        }
        else {
                summa = 4;
        }
}
else {
        summa = 2;
}
```

I detta exempel jämför vi först variabeln summa med 3. Om summa är större än 3 jämför vi summa med 5. Om summa är större än 5 får summa ett nytt värde, nämligen 6. Om summa är större än 3 men mindre än eller lika med 5 får variabeln summa värdet 4, annars (dvs om summa <=3 får den värdet 2.

forts


```
// Eftersom vi i föregående bilds if-sats, bara utförde 1 sats,
// så behöver satsen ej omslutas av { }
// föregående bilds if-sats kan alltså skrivas så här:
if (summa > 3)
        if (summa > 5)
                summa = 6;
        else
                summa = 4;
    else
        summa = 2;
// här nedan visas ytterligare ett sätt att skriva samma if-sats
if(summa>5) summa = 6;
    else if (summa>3) summa =4;
        else summa = 2;
```

forts if-sats

Man kan förstås ange sammansatta villkor,

&& betyder och , dvs båda uttrycken ska vara sanna för att det sammansatta uttrycket ska bli sant

|| betyder eller , det räcker att ett uttryck är sant

// vi kan därmed skriva if-satsen så här:

```
if (summa>5) summa=6;
   else if (summa>3 && summa <=5) summa=4;
else summa = 2;
```

Repetitionssatser sid 102- (131-)

- Ibland kan man vilja repetera en eller flera programsatser ett antal gånger. Man använder sig då av så kallade repetitionssatser.
- I Java finns ett antal repetitionssatser, t.ex. while-loop (det engelska ordet ”while” betyder ”så länge”).
- Exempel:

```
summa = 1;
while (summa <= 5) {
    summa = summa + 1;
    ... Satser som utförs 5 gånger ...
}
```

Så länge uttrycket (summa < =5) är sant kommer satserna i while-loopen att utföras

(145-) Repetitionssatser forts

for-sats:

for (startvärde; villkor; sats0)
{sats1;sats2;…..satsn;}

startvärde är en variabel som ges ett startvärde

sats1..satsn utförs så länge villkor är sant

sats0 utförs sist, innan nästa repetition

Om vi skriver förra iterationen med for-sats blir det så:

```
for (summa =1;summa<=5;summa++) {
      ... Satser som utförs 5 gånger ...
  }
```