**TDA 545:** Objektorienterad programmering

# Föreläsning 3:
# Booleans, if, switch

Magnus Myréen

Chalmers, läsperiod 1, 2015-2016

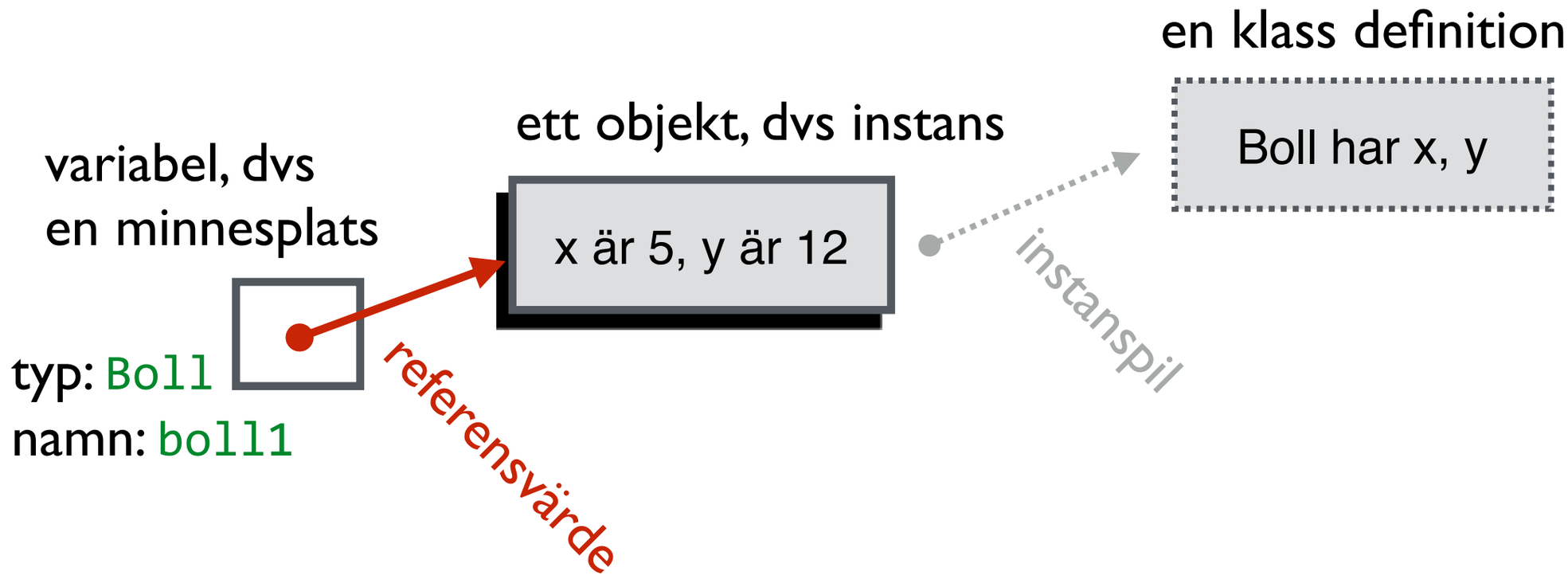
# Påminnelse om klasser och objekt

```
Boll boll1 = new Boll(5,12);
```

skapar ett nytt objekt (`boll1`) som är en instans av klassen `Boll`.

Hela raden kallas en deklaration,
och i detta fall med utgångsvärdet specificerat med `= new Boll(5,12)`.

Som bild:



# Påminnelse om terminologi och namn

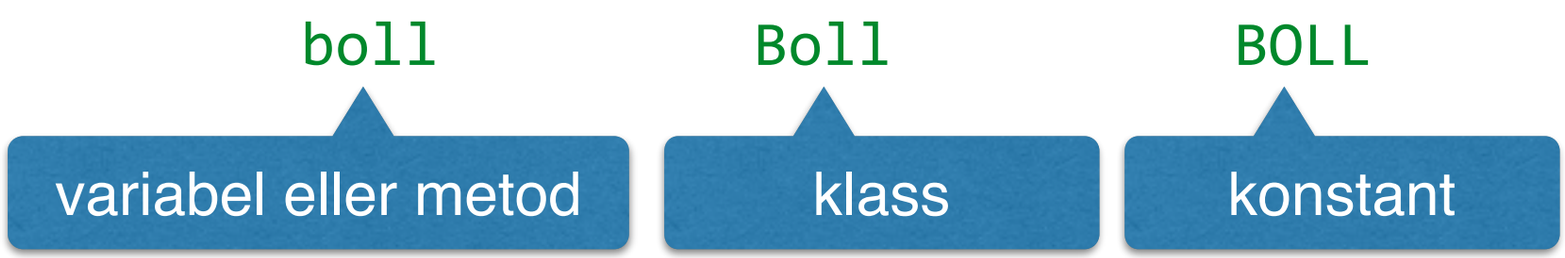
objekt = instans (av en klass)

*immutable object* — ett objekt som man inte kan ändra

primitiva typer (int, char, boolean, mm.)

klasstyper/referenstyper (allt annat, String, Rectangle, Ball, mm.)

Namn konventioner/regler:



# Deklarationer och tilldelningar

Det är viktigt att ni kan **se** hur kod körs.

( **se** = simulera på papper, eller i huvudet )

Exempel:

först evalueras deklarationen

```
int thisMonth;
int thisYear = 1987;
int nextYear = thisYear + 1;
nextYear = nextYear + 5;
```

Anta att minnet är:

variabeler, dvs
minnesplatser:

# Deklarationer och tilldelningar

Det är viktigt att ni kan **se** hur kod körs.

( **se** = simulera på papper, eller i huvudet )

Exempel:

först evalueras deklarationen

```
int thisMonth;
int thisYear = 1987;
int nextYear = thisYear + 1;
nextYear = nextYear + 5;
```

Anta att minnet är:

variabeler, dvs
minnesplatser:

minnesplatsen är tom

typ: `int`

namn: `thisMonth`

# Deklarationer och tilldelningar

Det är viktigt att ni kan **se** hur kod körs.

( **se** = simulera på papper, eller i huvudet )

Exempel:

nu har deklarationen körts
då raderar vi den

```
int thisMonth;
int thisYear = 1987;
int nextYear = thisYear + 1;
nextYear = nextYear + 5;
```

Anta att minnet är:

variabeler, dvs
minnesplatser:

typ: `int`

namn: `thisMonth`

# Deklarationer och tilldelningar

Det är viktigt att ni kan **se** hur kod körs.

( **se** = simulera på papper, eller i huvudet )

Exempel:

```
int thisYear = 1987;
int nextYear = thisYear + 1;
nextYear = nextYear + 5;
```

Anta att minnet är:

variabeler, dvs minnesplatser:

typ: `int`

namn: `thisMonth`

# Deklarationer och tilldelningar

Det är viktigt att ni kan **se** hur kod körs.

( **se** = simulera på papper, eller i huvudet )

Exempel:

```
int thisYear = 1987;
int nextYear = thisYear + 1;
nextYear = nextYear + 5;
```

Anta att minnet är:

variabeler, dvs minnesplatser:

typ: `int`

namn: `thisMonth`

# Deklarationer och tilldelningar

Det är viktigt att ni kan **se** hur kod körs.

( **se** = simulera på papper, eller i huvudet )

Exempel:

```
int thisYear = 1987;
int nextYear = thisYear + 1;
nextYear = nextYear + 5;
```

Anta att minnet är:

variabeler, dvs minnesplatser:



en ny minnesplats skapas med **1987** som värde

# Deklarationer och tilldelningar

Det är viktigt att ni kan **se** hur kod körs.

( **se** = simulera på papper, eller i huvudet )

Exempel:

```
int thisYear = 1987;
int nextYear = thisYear + 1;
nextYear = nextYear + 5;
```

Anta att minnet är:

variabeler, dvs minnesplatser:

typ: `int`

namn: `thisMonth`


typ: `int`

namn: `thisYear`

# Deklarationer och tilldelningar

Det är viktigt att ni kan **se** hur kod körs.

( **se** = simulera på papper, eller i huvudet )

Exempel:

```
int nextYear = thisYear + 1;
nextYear = nextYear + 5;
```

Anta att minnet är:

variabeler, dvs minnesplatser:

typ: `int`

namn: `thisMonth`


typ: `int`

namn: `thisYear`

# Deklarationer och tilldelningar

Det är viktigt att ni kan **se** hur kod körs.

( **se** = simulera på papper, eller i huvudet )

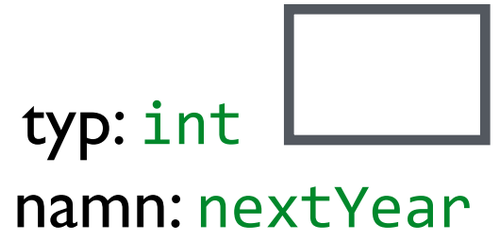
Exempel:

```
int nextYear = thisYear + 1;
nextYear = nextYear + 5;
```

Anta att minnet är:

variabeler, dvs minnesplatser:





# Deklarationer och tilldelningar

Det är viktigt att ni kan **se** hur kod körs.

( **se** = simulera på papper, eller i huvudet )

Exempel:

```
    nextYear = thisYear + 1;
nextYear = nextYear + 5;
```

Anta att minnet är:

variabeler, dvs minnesplatser:



# Deklarationer och tilldelningar

Det är viktigt att ni kan **se** hur kod körs.

( **se** = simulera på papper, eller i huvudet )

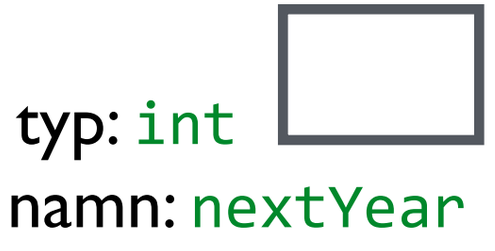
Exempel:

```
    nextYear = thisYear + 1;
nextYear = nextYear + 5;
```

Anta att minnet är:

variabeler, dvs minnesplatser:





# Deklarationer och tilldelningar

Det är viktigt att ni kan **se** hur kod körs.

( **se** = simulera på papper, eller i huvudet )

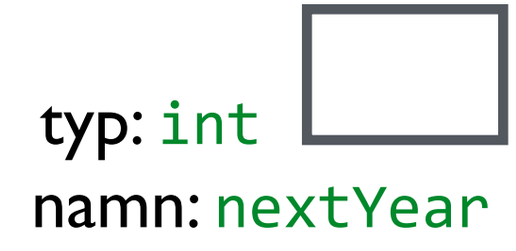
Exempel:

```
    nextYear =     1987 + 1;
nextYear = nextYear + 5;
```

Anta att minnet är:

variabeler, dvs minnesplatser:





# Deklarationer och tilldelningar

Det är viktigt att ni kan **se** hur kod körs.

( **se** = simulera på papper, eller i huvudet )

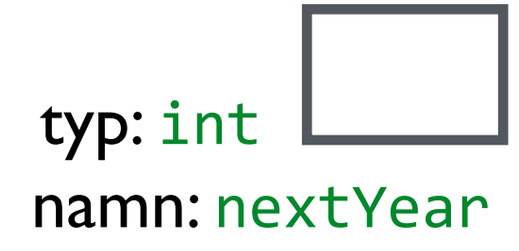
Exempel:

```
    nextYear =     1988;
nextYear = nextYear + 5;
```

Anta att minnet är:

variabeler, dvs minnesplatser:





# Deklarationer och tilldelningar

Det är viktigt att ni kan **se** hur kod körs.

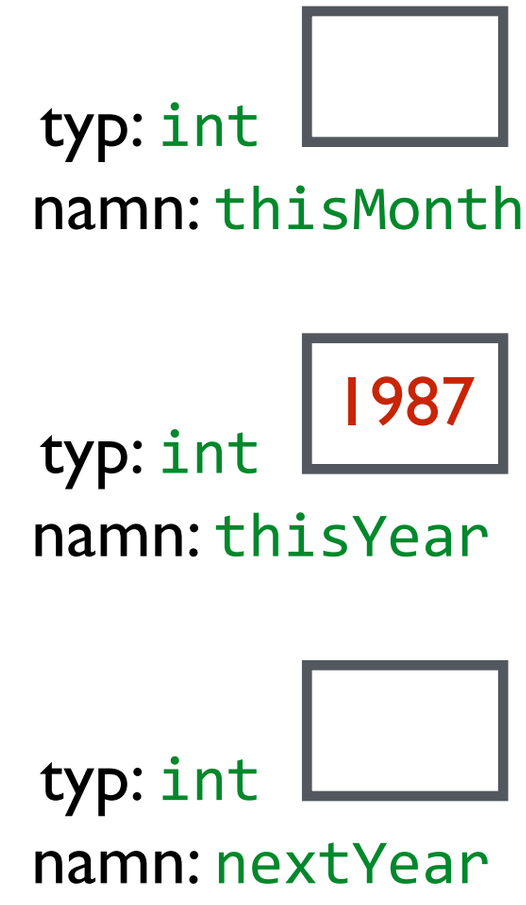
( **se** = simulera på papper, eller i huvudet )

Exempel:

```
    nextYear =     1988;
nextYear = nextYear + 5;
```

Anta att minnet är:

variabeler, dvs minnesplatser:



# Deklarationer och tilldelningar

Det är viktigt att ni kan **se** hur kod körs.

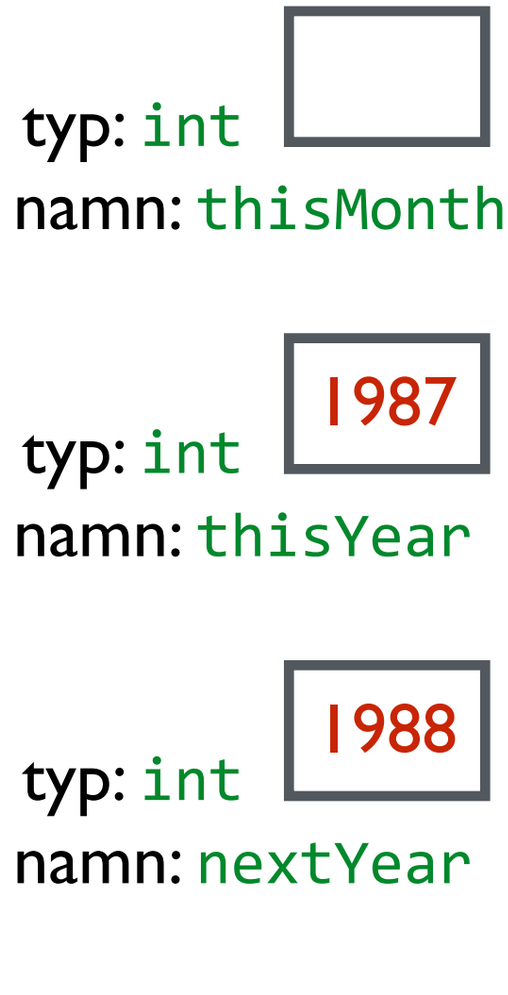
( **se** = simulera på papper, eller i huvudet )

Exempel:

```
nextYear = nextYear + 5;
```

Anta att minnet är:

variabeler, dvs minnesplatser:



# Deklarationer och tilldelningar

Det är viktigt att ni kan **se** hur kod körs.

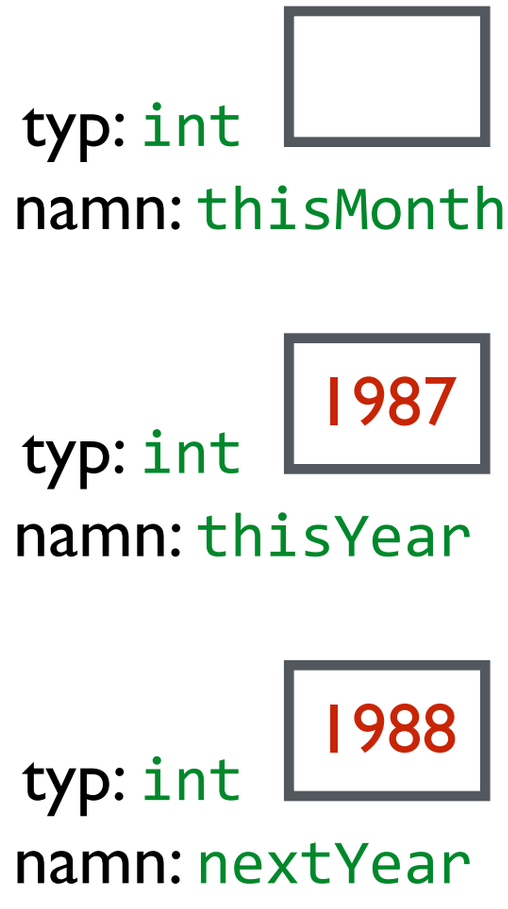
( **se** = simulera på papper, eller i huvudet )

Exempel:

```
nextYear = nextYear + 5;
```

Anta att minnet är:

variabeler, dvs minnesplatser:



# Deklarationer och tilldelningar

Det är viktigt att ni kan **se** hur kod körs.

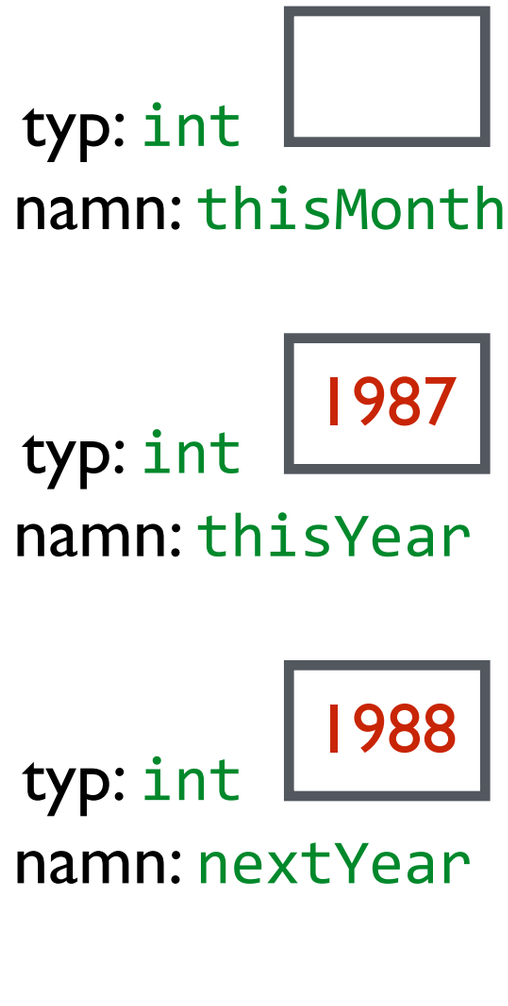
( **se** = simulera på papper, eller i huvudet )

Exempel:

```
nextYear =      1988 + 5;
```

Anta att minnet är:

variabeler, dvs minnesplatser:



# Deklarationer och tilldelningar

Det är viktigt att ni kan **se** hur kod körs.

( **se** = simulera på papper, eller i huvudet )

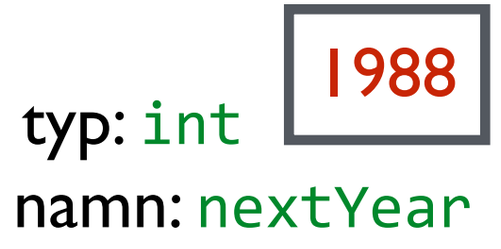
Exempel:

```
nextYear =     1993;
```

Anta att minnet är:

variabeler, dvs minnesplatser:





# Deklarationer och tilldelningar

Det är viktigt att ni kan **se** hur kod körs.

( **se** = simulera på papper, eller i huvudet )

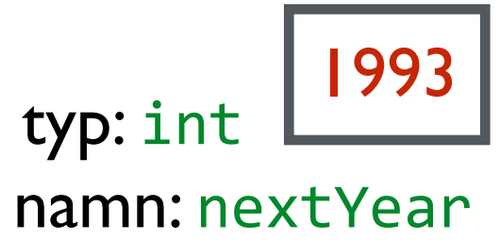
Exempel:

*koden har kört klart*

Anta att minnet är:

variabeler, dvs minnesplatser:





# Beräkningar

Det är viktigt att ni kan **se** hur kod körs.

( **se** = simulera på papper, eller i huvudet )

**Öva på egenhand! … på papper t.ex.**

Man kan testa sina egna simuleringar mot ett verktyg:

http://www.pythontutor.com/java.html

… men OBS ni måste kunna göra detta utan verktyg!

```
int thisMonth;
int thisYear = 1987;
int nextYear = thisYear + 1;
nextYear = nextYear + 5;
```

# Läsanvisning

Jag pekar nedan på var i boken de olika sakerna tas upp. Boken tar dock upp dessa saker ganska sent i sammanhang med andra saker som ni inte ännu behärskar. Det kan därför vara rätt svårt att läsa dessa hänvisningar nu.

- datatypen Boolean 4.2, 4.6
- relationsuttryck och logiska uttryck (.)
  - De Morgan's lag (s 217)
  - operatorer (utspritt)
  - if (4.1), switch(4.8)

Nästa föreläsning:

- for (12.7,12.6),
- while (7.3.1, 12.6), do-while (7.5)

# Datatypen Boolean och logiska uttryck

Utsagan: "det snöar" eller "a<8" är antingen sann eller falsk

Att ta reda på vilket är en form av beräkning som resulterar i något av värdena sant eller falskt

Typen boolean har litteralerna {true, false}
och de logiska (booleska) operatorerna:
and (&&), or (||) och not (!)

**Relationsuttryck och logiska uttryck**

Relationsuttryck: `a<10, a!=10, a<=10-b`

Logiska uttryck byggs upp av variabler av typ boolean, relationsuttryck och de logiska operatorerna and (&&), or (||) och not (!)

**Exempel:**

```
x>10 && x<20-y || b
10<x<20
10<x && x<20
```

går ej!

# Sanningstabell för de logiska operatorerna

| a | b | ! a | a && b | a \|\| b |
|---|---|---|---|---|
| false | false | true | false | false |
| false | true | true | false | true |
| true | false | false | false | true |
| true | true | false | true | true |

```
boolean sunny, overcast, pleasant;
double temp;
...
sunny = true;
overcast = !sunny;
pleasant = sunny && !(temp < 18.0 || 30.0 < temp);
```

testa koden med visualiseringsverktyget!

```
public class Weather {
  public static void main(String[] args) {
    boolean sunny, overcast, pleasant;
    double temp;
    temp = 19.0;
    sunny = true;
    overcast = !sunny;
    pleasant = sunny && !(temp < 18.0 || 30.0 < temp);
    System.out.println("Pleasant: " + pleasant);
  }
}
```

# Förenkling av logiska uttryck

**De Morgan's lag**

!(A && B) är samma sak som !A || !B

!(A || B) är samma sak som !A && !B

**Enkelt ex:**

Betrakta uttrycket på förra sidan

```
!(temp < 18.0 || 30.0 < temp)
```

är då detsamma som

```
!(temp < 18.0) && !(30.0 < temp)
```

vilket i sin tur är

```
18.0 <= temp && temp <= 30.0
```

Logiska uttryck evalueras med "lat" evaluering. (lazy evaluation)

# Förenkling av logiska uttryck

```
public class Weather {
  public static boolean p(String s, boolean v) {
    System.out.println(s);
    return v;
  }
  public static void main(String[] args) {
    boolean sunny, overcast, pleasant;
    double temp;
    temp = 17.0;
    sunny = true;
    overcast = !sunny;
    pleasant = sunny && !(p("mindre",temp < 18.0) || p("större",30.0 < temp));
    System.out.println("Pleasant: " + pleasant);
  }
}
```

*Vad skrivs ut?*

Logiska uttryck evalueras med "lat" evaluering. (lazy evaluation)

# De *flesta* operatorer i Java i precedensordning

| | |
|---|---|
| högsta | arg++, arg--, [], (), metod anrop: name.fieldname |
| unära | ++arg, --arg, +arg, -arg, !arg |
| | new, (type)expr |
| multiplikationsop. | *, /, % |
| additionsop. | +, - |
| relationsop. | <, <=, >, >=, instance of |
| likhetsop. | ==, != |
| tilldelning | =, +=, -=, ..... |
| logiskt och | && |
| logiskt eller | \|\| |

Prioriteten kan ändras med parenteser

dvs i Java binder **&&** starkare än **||**

a && b || c

är samma som

(a && b) || c

OBS:

(definieras olika i olika språk jmf **Pascal**:

[not] [*, /, div, mod, and] [+, -, or] [=, <, >, <= .....])

**Ada**:

[abs, not, **] [*, /, mod] [monadiska +,-]

[+, -, &] [=, /=, <, <=, >, >= ] [and, or, xor]

# if satsen: selektion/val

**Syntax**

```
if (<boolean expression1>) {
    <statements>
} else if (<boolean expression2>){
    <statements>
} else if ( ...
} else {
    <statements>
}
```

**Semantik**

1. Villkoret(n) evalueras (i tur och ordning).
2. Satserna efter det första sanna villkoret exekveras en gång.
3. Resten av villkoren evalueras inte.
4. Om inget av villkoren är sant exekveras satserna efter else en gång.

- Noll eller ett **else**,
- Noll eller flera **else if** tillåtna.

( evalueras = beräknas)

# if satsen

```
public class Test {
  public static void main(String[] args) {
    int a = 5; int b = 6;
    if (a < b) {
      System.out.println("a minst");
    } else if (a == b) {
      System.out.println("a = b");
    } else {
      System.out.println("b minst");
    } // end if
  }
} // end Test
```

**OBS Använd alltid block ( dvs { } )**

Annars luras du med indenteringen

```
if (false)
   System.out.println(“one”);
   System.out.println(“two”);
System.out.println(“three”);
```

# Exempel: Skottår

```
if (year%4 == 0) {
  if (year%100 == 0) {
    if (year%400 == 0) {
      return true;
    } else {
      return false;
    }
  } else {
    return true;
  }
} else {
  return false;
}
```

om året inte är delbart med 4 => ej skottår

om året är delbart med 4 -

    om det inte är delbart med 100 => skottår

    om det är delbart med 100 -

        om det är delbart med 400 => skottår

        annars inte

# Evaluering

Det är viktigt att ni kan **se** hur kod körs.

( **se** = simulera på papper, eller i huvudet )

Exempel:

först beräknas villkoret

```
if (year%4 == 0) {
  if (year%100 == 0) {
    if (year%400 == 0) {
      return true;
    } else {
      return false;
    }
  } else {
    return true;
  }
} else {
  return false;
}
```

Anta att minnet är:

variabel, dvs

en minnesplats


typ: `int`

namn: `year`

# Evaluering

Det är viktigt att ni kan **se** hur kod körs.

( **se** = simulera på papper, eller i huvudet )

Exempel:

först beräknas villkoret

```
if (year%4 == 0) {
  if (year%100 == 0) {
    if (year%400 == 0) {
      return true;
    } else {
      return false;
    }
  } else {
    return true;
  }
} else {
  return false;
}
```

Anta att minnet är:

variabel, dvs

en minnesplats

# Evaluering

Det är viktigt att ni kan **se** hur kod körs.

( **se** = simulera på papper, eller i huvudet )

Exempel:

```
if (300 % 4 == 0) {
  if (year%100 == 0) {
    if (year%400 == 0) {
      return true;
    } else {
      return false;
    }
  } else {
    return true;
  }
} else {
  return false;
}
```

först beräknas villkoret

Anta att minnet är:

variabel, dvs

en minnesplats

# Evaluering

Det är viktigt att ni kan **se** hur kod körs.

( **se** = simulera på papper, eller i huvudet )

Exempel:

```
if (       0 == 0) {
  if (year%100 == 0) {
    if (year%400 == 0) {
      return true;
    } else {
      return false;
    }
  } else {
    return true;
  }
} else {
  return false;
}
```

först beräknas villkoret

Anta att minnet är:

variabel, dvs

en minnesplats

typ: `int`


namn: `year`

# Evaluering

Det är viktigt att ni kan **se** hur kod körs.

( **se** = simulera på papper, eller i huvudet )

Exempel:

```
if (true) {
  if (year%100 == 0) {
    if (year%400 == 0) {
      return true;
    } else {
      return false;
    }
  } else {
    return true;
  }
} else {
  return false;
}
```

först beräknas villkoret

Anta att minnet är:

variabel, dvs

en minnesplats

# Evaluering

Det är viktigt att ni kan **se** hur kod körs.

( **se** = simulera på papper, eller i huvudet )

Exempel:

nu vet vi att första { … }
körs och **else raderas**

```
if (true) {
  if (year%100 == 0) {
    if (year%400 == 0) {
      return true;
    } else {
      return false;
    }
  } else {
    return true;
  }
} else {
  return false;
}
```

t minnet är:

variabel, dvs

en minnesplats


typ: `int`

namn: `year`

# Evaluering

Det är viktigt att ni kan **se** hur kod körs.

( **se** = simulera på papper, eller i huvudet )

Exempel:

```
if (year%100 == 0) {
  if (year%400 == 0) {
    return true;
  } else {
    return false;
  }
} else {
  return true;
}
```

Anta att minnet är:

variabel, dvs

en minnesplats


typ: `int`

namn: `year`

# Evaluering

Det är viktigt att ni kan **se** hur kod körs.

( **se** = simulera på papper, eller i huvudet )

Exempel:

```
if (year%100 == 0) {
  if (year%400 == 0) {
    return true;
  } else {
    return false;
  }
} else {
  return true;
}
```

Anta att minnet är:

variabel, dvs

en minnesplats

typ: `int`


namn: `year`

# Evaluering

Det är viktigt att ni kan se hur kod körs.

( se = simulera på papper, eller i huvudet )

Exempel:

```
if ( 300 % 100 == 0) {
  if (year%400 == 0) {
    return true;
  } else {
    return false;
  }
} else {
  return true;
}
```

Anta att minnet är:

variabel, dvs

en minnesplats

# Evaluering

Det är viktigt att ni kan se hur kod körs.

( se = simulera på papper, eller i huvudet )

Exempel:

```
if (          0 == 0) {
  if (year%400 == 0) {
    return true;
  } else {
    return false;
  }
} else {
  return true;
}
```

Anta att minnet är:

variabel, dvs

en minnesplats

typ: `int`


namn: `year`

# Evaluering

Det är viktigt att ni kan se hur kod körs.

( se = simulera på papper, eller i huvudet )

Exempel:

```
if (true) {
  if (year%400 == 0) {
    return true;
  } else {
    return false;
  }
} else {
  return true;
}
```

Anta att minnet är:

variabel, dvs

en minnesplats

typ: `int`

300

namn: `year`

# Evaluering

Det är viktigt att ni kan **se** hur kod körs.

( **se** = simulera på papper, eller i huvudet )

Exempel:

```
if (true) {
  if (year%400 == 0) {
    return true;
  } else {
    return false;
  }
} else {
  return true;
}
```

Anta att minnet är:

variabel, dvs

en minnesplats

typ: `int`

300

namn: `year`

# Evaluering

Det är viktigt att ni kan **se** hur kod körs.

( **se** = simulera på papper, eller i huvudet )

Exempel:

```
if (year%400 == 0) {
  return true;
} else {
  return false;
}
```

Anta att minnet är:

variabel, dvs

en minnesplats

typ: `int`


namn: `year`

# Evaluering

Det är viktigt att ni kan **se** hur kod körs.

( **se** = simulera på papper, eller i huvudet )

Exempel:

```
if (year%400 == 0) {
  return true;
} else {
  return false;
}
```

Anta att minnet är:

variabel, dvs
en minnesplats

typ: `int`


namn: `year`

# Evaluering

Det är viktigt att ni kan **se** hur kod körs.

( **se** = simulera på papper, eller i huvudet )

Exempel:

```
if (300 % 400 == 0) {
  return true;
} else {
  return false;
}
```

Anta att minnet är:

variabel, dvs
en minnesplats

# Evaluering

Det är viktigt att ni kan **se** hur kod körs.

( **se** = simulera på papper, eller i huvudet )

Exempel:

```
if (       300 == 0) {
  return true;
} else {
  return false;
}
```

Anta att minnet är:

variabel, dvs
en minnesplats

typ: `int`


namn: `year`

# Evaluering

Det är viktigt att ni kan **se** hur kod körs.

( **se** = simulera på papper, eller i huvudet )

Exempel:

```
if (false) {
  return true;
} else {
  return false;
}
```

Anta att minnet är:

variabel, dvs

en minnesplats


typ: `int`

namn: `year`

# Evaluering

Det är viktigt att ni kan **se** hur kod körs.

( **se** = simulera på papper, eller i huvudet )

Exempel:

```
if (false) {
  return true;
} else {
  return false;
}
```

Anta att minnet är:

variabel, dvs

en minnesplats


typ: `int`

namn: `year`

# Evaluering

Det är viktigt att ni kan **se** hur kod körs.

( **se** = simulera på papper, eller i huvudet )

Exempel:

```
return false;
```

Anta att minnet är:

variabel, dvs
en minnesplats

typ: `int`

namn: `year`


resultatet är att metoden/funktionen returnerar **false**, mera om metoder senare…

# Testa koden själv!

```
public class Leap {
  public static boolean isLeapYear(int year) {
    if (year%4 == 0) {
      if (year%100 == 0) {
        if (year%400 == 0) {
          return true;
        } else {
          return false;
        }
      } else {
        return true;
      }
    } else {
      return false;
    }
  }
  public static void main(String[] args) {
    System.out.println(isLeapYear(300));
  }
}
```

**Verktyg:** http://www.pythontutor.com/java.html

# Förenkla...

```
if (year%4 == 0) {
  if (year%100 == 0) {
    if (year%400 == 0) {
      return true;
    } else {
      return false;
    }
  } else {
    return true;
  }
} else {
  return false;
}
```

```
if (year%4 == 0) {
  if (year%100 == 0) {


    return (year%400 == 0)


  } else {
    return true;
  }
} else {
  return false;
}
```

# Förenkla...

```
if (year%4 == 0) {
  if (year%100 == 0) {
    return (year%400 == 0)
  } else {
    return true;
  }
} else {
  return false;
}
```

```
if (year%4 == 0) {
  return (year%100 != 0) ||
         (year%400 == 0)
} else {
  return false;
}
```

# Förenkla...

```
if (year%4 == 0) {
  return (year%100 != 0) ||
         (year%400 == 0)
} else {
  return false;
}
```

```
return (year%4 == 0) &&
       ((year%100 != 0) ||
        (year%400 == 0))
```

**OBS:**
Förenkla försiktigt!
Gör inte koden svårläst eller fel.

**OBS:**
Logiska uttryck evalueras med "lat" evaluering. (lazy evaluation)

# switch/case satsen: selection/val

**Syntax**

switch ( <**integer** expression> ) {(Java7 även String)

```
   case <value> :
      <statements>
      break; // exit the switch
   case <value1> :
      <statements>
   case <value2> :
      <statements>
      break; // exit the switch
   default :
      <statement>
      break; // exit the switch
}
```

**Exempel**

```
char ch = nåt lämpligt tecken
switch (ch) {
   case 'x':
      System.out.println("Option x");
      break;
   case 'c':
      System.out.println("Option c");
      break;
   default:
      ...println("okänd flagga");
      break;
} // end case
```

# switch exempel

```
// Precon: 1 <= month <= 12
// Postcon: returns the number of days in the month
public static int daysInMonth(int year, int month) {
  switch (month) {
    case 1: case 3: case 5: case 7: case 8:
    case 10: case 12: return 31; // return => no break needed
    case 4: case 6: case 9: case 11: return 30;
  case 2:
    if (isLeapYear(year)) {
      return 29;
    } else {
      return 28;
    }
  default:
    return -1;
  } // end switch
} // end daysInMonth
```

# Evaluering/beräkning av switch

Vi börjar med att räkna uttrycket som bestämmer vad vi kommer att radera.

```
switch (month) {
  case 1: case 3: case 5: case 7: case 8:
  case 10: case 12: return 31;
  case 4: case 6: case 9: case 11: return 30;
  case 2:
    if (isLeapYear(year)) {
      return 29;
    } else { return 28; }
default:
  return -1;
}
```

Anta att minnet är:

variabel, dvs
en minnesplats

typ: `int`


namn: `month`

# Evaluering/beräkning av switch

Vi börjar med att räkna uttrycket som bestämmer vad vi kommer att radera.

```
switch (month) {
  case 1: case 3: case 5: case 7: case 8:
  case 10: case 12: return 31;
  case 4: case 6: case 9: case 11: return 30;
  case 2:
    if (isLeapYear(year)) {
      return 29;
    } else { return 28; }
default:
  return -1;
}
```

Anta att minnet är:

variabel, dvs
en minnesplats

typ: `int`


namn: `month`

# Evaluering/beräkning av switch

Vi börjar med att räkna uttrycket som bestämmer vad vi kommer att radera.

```
switch (6) {
  case 1: case 3: case 5: case 7: case 8:
  case 10: case 12: return 31;
  case 4: case 6: case 9: case 11: return 30;
  case 2:
    if (isLeapYear(year)) {
      return 29;
    } else { return 28; }
default:
  return -1;
}
```

Anta att minnet är:

variabel, dvs
en minnesplats

typ: `int`


namn: `month`

# Evaluering/beräkning av switch

```
switch (6) {
  case 1: case 3: case 5: case 7: case 8:
  case 10: case 12: return 31;
  case 4: case 6: case 9: case 11: return 30;
  case 2:
    if (isLeapYear(year)) {
      return 29;
    } else { return 28; }
default:
  return -1;
}
```

Anta att minnet är:

variabel, dvs
en minnesplats


typ: `int`
namn: `month`

# Evaluering/beräkning av switch

```
switch (6) {

          case 6: case 9: case 11: return 30;
  case 2:
    if (isLeapYear(year)) {
      return 29;
    } else { return 28; }
default:
    return -1;
}
```

Anta att minnet är:

variabel, dvs
en minnesplats

typ: `int`


namn: `month`

# Evaluering/beräkning av switch

```
switch (6) {

          case 6: case 9: case 11: return 30;
  case 2:
    if (isLeapYear(year)) {
      return 29;
    } else { return 28; }
default:
    return -1;
}
```

Anta att minnet är:

variabel, dvs
en minnesplats

typ: `int`


namn: `month`

# Evaluering/beräkning av switch

```
switch (6) {

          case 6: case 9: case 11: return 30;
  case 2:
    if (isLeapYear(year)) {
      return 29;
    } else { return 28; }
default:
    return -1;
}
```

Anta att minnet är:

variabel, dvs
en minnesplats

typ: `int`


namn: `month`

# Evaluering/beräkning av switch

```
switch (6) {

          case 6: case 9: case 11: return 30;
  case 2:
    if (isLeapYear(year)) {
      return 29;
    } else { return 28; }
default:
    return -1;
}
```

case 9 och 11 är bara namn, de skippar vi

Anta att minnet är:

variabel, dvs
en minnesplats

typ: `int`


namn: `month`

# Evaluering/beräkning av switch

```
switch (6) {

          case 6:                          return 30;
  case 2:
    if (isLeapYear(year)) {
      return 29;
    } else { return 28; }
default:
    return -1;
}
```

Anta att minnet är:

variabel, dvs
en minnesplats

typ: `int` 6

namn: `month`

# Evaluering/beräkning av switch

```
switch (6) {

          case 6:                          return 30;
  case 2:
    if (isLeapYear(year)) {
      return 29;
    } else { return 28; }
default:
    return -1;
}
```

Anta att minnet är:

variabel, dvs
en minnesplats

typ: `int` 6

namn: `month`

# Evaluering/beräkning av switch

resultatet är att metoden returnerar 30

```
return 30;
```

Anta att minnet är:

variabel, dvs
en minnesplats

typ: `int`


namn: `month`