

# Föreläsning 7

## Flerdimensionella fält

### ArrayList

### enum

### switch-satsen

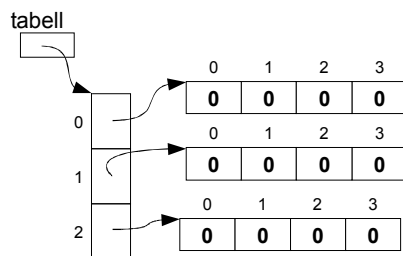
1

## Tvådimensionella fält



Tvådimensionella fält är *fält av fält*.

```
int[][] tabell = new int[3][4];
```



|   | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 |

2

## Tvådimensionella fält

Istället för att skapa ett tvådimensionellt fält med **new** kan fältet skapas genom att initiera värden till fältet vid deklarationen.

```
int[][] tabell = {{12, 34, 71, 9},  
                 {53, 43, 33, 68},  
                 {29, 10, 3, 42}};
```

|   |    |    |    |    |
|---|----|----|----|----|
|   | 0  | 1  | 2  | 3  |
| 0 | 12 | 34 | 71 | 9  |
| 1 | 53 | 43 | 33 | 68 |
| 2 | 29 | 10 | 3  | 42 |

Eftersom ett tvådimensionellt fält är ett fält med referenser till ett endimensionellt fält, kan raderna vara olika långa

```
int[][] tabell = {{12, 34, 71, 9},  
                 {53, 43, 33},  
                 {29, 10}};
```

|   |    |    |    |   |
|---|----|----|----|---|
|   | 0  | 1  | 2  | 3 |
| 0 | 12 | 34 | 71 | 9 |
| 1 | 53 | 43 | 33 |   |
| 2 | 29 | 10 |    |   |
|   |    | 3  |    |   |

## Tvådimensionella fält

```
int[][] tabell = {{12, 34, 71, 9},  
                 {53, 43, 33},  
                 {29, 10}};
```

|   |    |    |    |   |
|---|----|----|----|---|
|   | 0  | 1  | 2  | 3 |
| 0 | 12 | 34 | 71 | 9 |
| 1 | 53 | 43 | 33 |   |
| 2 | 29 | 10 |    |   |

tabell[0].length    ger 4

tabell[1].length    ger 3

tabell[2].length    ger 2

Arrays.sort(tabell[0])    sorterar rad 0 i tabell

Arrays.sort(tabell[1])    sorterar rad 1 i tabell

Arrays.sort(tabell[2])    sorterar rad 2 i tabell

## Problemexempel

Skriv ett program som läser in en NxN matris, samt avgör och skriver ut huruvida matrisen är symmetrisk eller inte. Matrisens gradtal ges som indata. För en symmetrisk matris A gäller att

$$a_{ij} = a_{ji} \text{ för alla } i \text{ och } j$$

### Analys:

Indata: Ett gradtal samt en kvadratisk matris med detta gradtal.

Utdata: Utskrift av huruvida den inlästa matrisen är symmetrisk eller inte.

### Exempel: Matrisen

|   |   |   |
|---|---|---|
| 1 | 2 | 3 |
| 2 | 3 | 4 |
| 3 | 4 | 5 |

ger utskriften MATRISEN ÄR SYMMETRISK, medan matrisen

|   |   |   |
|---|---|---|
| 1 | 2 | 3 |
| 3 | 4 | 5 |
| 5 | 6 | 7 |

ger utskriften MATRISEN ÄR INTE SYMMETRISK, medan matrisen

5

### Design:

#### Diskussion:

När vi skall kontrollera om matrisen är symmetrisk utgår vi från att så är fallet. För att handha denna kunskap sätter vi en boolsk variabel, som vi kan kalla *okey* till värdet **true**. Sedan genomlöper vi matrisen och om vi då påträffar något element  $a_{ij}$  för vilket det gäller att  $a_{ij} \neq a_{ji}$  har vi en icke-symmetrisk matris. Detta ”kommer vi ihåg” genom att sätta *okey* till värdet **false**.

#### Algoritm:

1. Läs gradtalet  $n$
2. Läs matrisen  $A$
3.  $okey = \text{true};$
4. För varje element  $a_{ij}$  i matrisen  $A$ 
  - 4.1. **if** ( $a_{ij} \neq a_{ji}$ )  
 $okey = \text{false};$
5. **if**  $okey$   
Skriv ut ”Matrisen är symmetrisk.”.  
**else**  
Skriv ut ”Matrisen är INTE symmetrisk.”.

#### Datarepresentation:

$A$  är av datatypen **double[][]**.

6

### Implementation:

```
import javax.swing.*;
public class Symmetric {
    public static void main(String[] args) {
        String input = JOptionPane.showInputDialog("Ange matrisens gradtal: ");
        int size = Integer.parseInt(input);
        double[][] matrix = readMatrix(size);
        if (isSymmetric(matrix))
            JOptionPane.showMessageDialog(null, "Matrisen är symetrisk!");
        else
            JOptionPane.showMessageDialog(null, "Matrisen är INTE symetrisk!");
    } // main
}
```

### Implementation: fortsättning

```
public static double[][] readMatrix(int size) {
    double[][] theMatrix = new double[size][size];
    for (int row = 0; row < size; row = row + 1) {
        for (int col = 0; col < size; col = col + 1) {
            String input = JOptionPane.showInputDialog("Ge element (" + row + ", " + col + ")");
            theMatrix[row][col] = Double.parseDouble(input);
        }
    }
    return theMatrix;
} // readMatrix

//before: matrix != null
public static boolean isSymmetric(double[][] matrix) {
    boolean okay = true;
    for (int row = 0; row < matrix.length; row = row + 1)
        for (int col = 0; col < matrix[row].length; col = col + 1)
            if (matrix[row][col] != matrix[col][row])
                okay = false;
    return okay;
} //isSymmetric
} //Symmetric
```

Alternativ implementation av metoden readMatrix, med användning av ett Scanner-objekt:

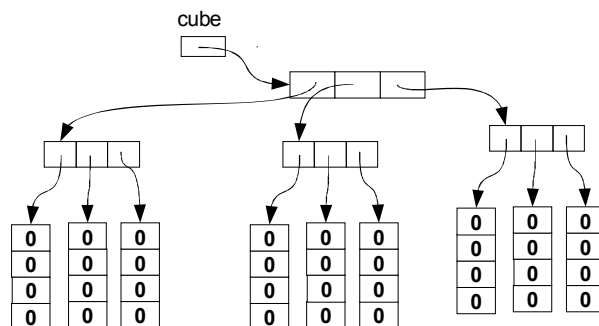
```
import java.util.*;
...
public static double[][] readMatrix(int size) {
    double[][] theMatrix = new double[size][size];
    String input = JOptionPane.showInputDialog( "Ge element: ");
    Scanner sc = new Scanner(input);
    for (int row = 0; row < size; row = row + 1) {
        int col = 0;
        while (col < size) {
            if (sc.hasNextDouble()) {
                theMatrix[row][col] = sc.nextDouble();
                col = col + 1;
            }
            else {
                input = JOptionPane.showInputDialog( "Ge fler element: ");
                sc = new Scanner(input);
            }
        }
    }
    return theMatrix;
} // readMatrix
```

9

## Flerdimensionella fält

Man kan ha ett godtyckligt antal dimensioner i ett fält, dvs man kan bilda fält av fält av fält av ....

```
int[][][] cube = new int[3][3][4];
```



10

## Flerdimensionella fält



En bild kan lagras som ett tvådimensionellt fält av bildpunkter (eller pixels).

I en gråskalebild är varje bildpunkt ett heltal i intervallet  $[0, 255]$ , där 0 betecknar svart och 255 betecknar vitt.

I en färgbild utgörs varje bildpunkt av tre heltal i intervallet  $[0, 255]$ , som representerar intensiteten av färgerna rött, grönt respektive blått.

En gråskalebild respektive en färgbild med höjden 800 pixels och bredden 600 pixels avbildas således enligt:

```
int[][] grayImage = new int[800][600];  
int[][][] colorImage = new int[800][600][3];
```

## Klassen ArrayList

Ett fält är en statisk datastruktur, vilket innebär att storleken på fältet måste anges när fältet skapas. Detta innebär att fält inte är särskilt väl anpassade för att handha dynamiska datasamlingar, dvs datasamlingar som under sin livstid kan variera i storlek.

För att handha dynamiska datasamlingar i ett fält måste man själv utveckla programkod för att t.ex:

- ta bort ett element ur fältet
- lägga in ett nytt element på en given position i fältet
- öka storleken på fältet om ett nytt element inte ryms.

Klassen `ArrayList` är en standardklass (av flera) för att handha samlingar av objekt. Särskilt när vi handhar dynamiska datasamlingar, är det lämpligt att använda klassen `ArrayList` istället för ett endimensionellt fält.

`ArrayList` finns i paketet `java.util`.

## Klassen ArrayList<E>

| Metod                                             | Beskrivning                                                                                                                                     |
|---------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| ArrayList<E>()                                    | skapar en tom ArrayList för element av typen E.                                                                                                 |
| <b>void</b> add(E elem)                           | lägger in <b>elem</b> sist i listan (d.v.s. efter de element som redan finns i listan).                                                         |
| <b>void</b> add(int pos, E elem)                  | lägger in <b>elem</b> på plats <b>pos</b> . Efterföljande element flyttas ett position framåt i listan.                                         |
| <b>void</b> add(Collection<? <b>extends</b> E> c) | lägger in alla element i samlingen c sist i listan.                                                                                             |
| E get(int pos)                                    | returnerar elementet på plats <b>pos</b> .                                                                                                      |
| E set(int pos, E elem)                            | ersätter elementet på plats <b>pos</b> med elem, returnerar elementet som fanns på platsen <b>pos</b> .                                         |
| E remove(int pos)                                 | tar bort elementet på plats <b>pos</b> , returnerar det borttagna elementet. Efterföljande element i listan flyttas en position bakåt i listan. |

13

## Klassen ArrayList<E>

| Metod                         | Beskrivning                                                                                            |
|-------------------------------|--------------------------------------------------------------------------------------------------------|
| int size()                    | returnerar antalet element i listan                                                                    |
| boolean isEmpty()             | returnerar <b>true</b> om listan är tom, annars returneras <b>false</b>                                |
| int indexOf(E elem)           | returnerar index för elementet <b>elem</b> om detta finns i listan, annars returneras -1               |
| boolean contains(Object elem) | returnerar <b>true</b> om <b>elem</b> finns i listan, annars returneras <b>false</b>                   |
| void clear()                  | tar bort alla elementen i listan                                                                       |
| String toString()             | returnerar en textrepresentation på formen [e <sub>1</sub> , e <sub>2</sub> , . . . , e <sub>n</sub> ] |

Anm: Metoderna `indexOf` och `contains` förutsätter att objekten i listan kan jämföras, d.v.s. klassen som objekten tillhör måste definiera metoden

**public boolean equals(Object obj)**

Alla standardklasser, såsom `String`, `Integer` och `Double`, definierar metoden `equals`.

14

## Klassen ArrayList

Klassen `ArrayList` är en *generisk klass*. Detta innebär att när man skapar en lista av klassen `ArrayList` måste man ange en typparameter som specificerar vilken typ av objekt som skall lagras i listan.

### Exempel:

```
ArrayList<String> words = new ArrayList<String>();  
ArrayList<Integer> values = new ArrayList<Integer>();  
ArrayList<BigInteger> bigValues = new ArrayList<BigInteger>();  
ArrayList<Person> members = new ArrayList<Person>();
```

I en `ArrayList` kan man *endast spara objekt*, dvs. en `ArrayList` kan inte innehålla de primitiva datatyperna (t.ex. **int**, **double**, **boolean** och **char**).

Vill man handha primitiva datatyper med hjälp av en `ArrayList` måste man lagra objekt av motsvarande omslagsklass.

15

## Autoboxing och auto-unboxing

Typomvandling sker automatiskt mellan primära datatyper och motsvarande omslagsklass. Detta kallas för *autoboxing* respektive *auto-unboxing*.

Istället för att skriva

```
Integer talObjekt = new Integer(10);  
...  
int tal = talObjekt.toValue();
```

kan man skriva

```
Integer talObjekt = 10;    //autoboxing  
...  
int tal = talObjekt;      //auto-unboxing
```

16



## Förenklad for-sats

När man vill löpa igenom alla objekt i en samlingar (t.ex. ett objekt av `ArrayList` eller ett en-dimensionellt fält) finns den förenklade **for**-satsen.

### Genomlöpnig av hela samlingarna med den vanliga for-satsen

```
double[] values = new double[100];
ArrayList<String> listan = new ArrayList<String>();
...
for (int index = 0; index < values.length; index = index + 1)
    System.out.println(values[index]);
for (int pos = 0; pos < listan.size(); pos = pos + 1)
    System.out.println(listan.get(pos));
```

### Genomlöpnig av hela samlingarna med den förenklade for-satsen

```
double[] values = new double[100];
ArrayList<String> listan = new ArrayList<String>();
...
for (double v : values)
    System.out.println(v);
for (String str : listan)
    System.out.println(str);
```

17

## Problemexempel

Skriv en metod

```
private static ArrayList<Integer> readLista()
```

som läser in en indatasekvens består av osorterade heltal från standard input och returnerar dessa i en `ArrayList`. I indatasekvensen kan samma tal förekomma flera gånger, men i listan skall endast den första förekomsten av varje unikt tal skall lagras.

### Exempel:

Antag att indatasekvensen består av talen 1 4 1 2 4 5 12 3 2 4 1,  
ett anrop av metoden `readList` skall då returnera en lista som innehåller talen 1, 4, 2, 5, 12 och 3.

18

### Algoritm:

1. **while** (fler tal att läsa)
  - 2.1. läs *talet*
  - 2.2. **if** (*talet* inte finns i *listan*)
    - 2.2.1. lagra *talet* i *listan*;
3. returnera *listan*

### Implementation

```
public static ArrayList<Integer> readList() {  
    ArrayList<Integer> list = new ArrayList<Integer>();  
    Scanner in = new Scanner(System.in);  
    while (in.hasNextInt()) {  
        int value = in.nextInt();  
        if (!list.contains(value)) {  
            list.add(value);  
        }  
    }  
    return list;  
}  
//readList
```

19

## Klassen PhoneBook implementerad med fält

```
public class PhoneBook {  
    private Entry[] book;  
    private int count;  
    public PhoneBook(int size) {  
        book = new Entry[size];  
        count = 0;  
    }  
    //constructor  
    public void put(String name, String nr) {  
        book[count] = new Entry(name, nr);  
        count = count + 1;  
    }  
    //put  
    public String get(String name) {  
        for (int i = 0; i < count; i = i + 1)  
            if (name.equals(book[i].getName()))  
                return book[i].getNumber();  
        return null;  
    }  
    //get  
}  
//PhoneBook
```

Maximal storlek måste anges

Antalet element måste bokföras

```
public class Entry {  
    private String name;  
    private String number;  
    public Entry(String name, String number) {  
        this.name = name;  
        this.number = number;  
    }  
    //constructor  
    public String getName() {  
        return name;  
    }  
    //getName  
    public String getNumber() {  
        return number;  
    }  
    //getNumber  
}  
//Entry
```

20

## Klassen PhoneBook implementation med ArrayList

```
import java.util.ArrayList;
public class PhoneBook {
    private ArrayList<Entry> book = new ArrayList<Entry>();
    public PhoneBook(int size) {
        book = new ArrayList<Entry>();
    } //constructor
    public void put(String name, String nr) {
        book.add(new Entry(name, nr));
    } //put
    public String get(String name) {
        for (Entry e : book)
            if (name.equals(e.getName()))
                return e.getNumber();
        return null;
    } //get
} //PhoneBook
```

```
public class Entry {
    private String name;
    private String number;
    public Entry(String name, String number) {
        this.name = name;
        this.number = number;
    } //constructor
    public String getName() {
        return name;
    } //getName
    public String getNumber() {
        return number;
    } //getNumber
} //Entry
```

21

## Uppräkningstyper - enum

I bland vill man kunna använda variabler som endast skall kunna anta vissa givna värden:

gender: MALE, FEMALE

state: READY, RUNNING, BLOCKED, DEAD

suit: HEARTS, SPADES, DIAMONDS, CLUBS

season: WINTER, SPRING, SUMMER, FALL

bearing: NORTH, WEST, SOUTH, EAST

I Java kan detta göras genom att deklarerera särskilda *uppräkningsklasser*:

```
public enum suit {
    HEARTS, SPADES, DIAMONDS, CLUBS;
} //suit
```

enum inte class

Räkna upp alla värden (instanser) typen skall ha

22

## Uppräkningstyper - enum

Varje deklarerat (uppräknat) värde i en **enum** är en instans av klassen.

Varje värde är implicit **public**, **static** och **final**, alltså en *klasskonstant*.

| Metod                               | Beskrivning                                                                                                                                                                                                                                                  |
|-------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>int</b> compareTo(E o)           | returnerar ett negativt heltal om aktuellt objekt är mindre än argumentet o, 0 om aktuellt objekt och argumentet o är lika och ett positivt heltal om aktuellt objekt är större än argumentet o. Jämförelsen görs efter den ordning objekten har deklarerats |
| <b>boolean</b> equals(E o)          | returnerar <b>true</b> om o är lika med aktuellt objekt, annars returneras <b>false</b>                                                                                                                                                                      |
| String name()                       | returnerar namnet på aktuellt objekt (enligt deklarationen)                                                                                                                                                                                                  |
| <b>int</b> ordinal()                | returnerar ordningstalet för aktuellt objekt (enligt deklarationen)                                                                                                                                                                                          |
| String toString()                   | returnerar namnet på aktuellt objekt (enligt deklarationen)                                                                                                                                                                                                  |
| <b>static</b> E valueOf(String str) | returnerar objektet med det angivna namnet                                                                                                                                                                                                                   |
| <b>static</b> E[] values()          | returnerar ett fält innehållande objekten i klassen                                                                                                                                                                                                          |

23

## Lägga till tillstånd på enum-konstanter

Eftersom **enum** definierar en klass kan man ge tillstånd och beteenden till objekten som tillhör en uppräkningsstyp,

Tillståndet hos en instans beskrivs med hjälp av instansvariabler.

Tillståndet för en instans ges som argument till instansen.

En konstruktor för att initiera tillståndet behöver definieras. Konstruktorn får/kan inte vara **public**.

```
public enum Customer {  
    CHILD(50), ADULT(100), SENIOR(60);  
    private int price;  
    private Customer(int price) {  
        this.price = price;  
    } //constructor  
    public int getPrice() {  
        return price;  
    } //getPrice  
} //Customer
```

Argument som anger  
tillståndet

Konstruktor som  
sätter tillståndet

24

## Lägga till beteenden på enum-konstanter

```
public enum DayOfWeek {  
    MONDAY, TUESDAY, WEDNESDAY, THURSDAY,  
    FRIDAY, SATURDAY, SUNDAY;  
    public int nrInWeek() {  
        return this.ordinal() + 1;  
    } //nrInWeek  
    public DayOfWeek tomorrow() {  
        DayOfWeek[] days = DayOfWeek.values();  
        return days[(this.ordinal() + 1) % days.length];  
    } //tomorrow  
    public DayOfWeek yesterday() {  
        DayOfWeek[] days = DayOfWeek.values();  
        return days[(this.ordinal() - 1) % days.length];  
    } //yesterday  
    public static DayOfWeek getDayWithNr(int dayNr) {  
        DayOfWeek[] days = DayOfWeek.values();  
        return days[dayNr - 1];  
    } //getDayWithNr  
} //DayOfWeek
```

Returerar vilket  
ordningsnummer i veckan  
Den aktuella dagen har

Returerar dagen  
efter aktuell dag

Returerar dagen  
före aktuell dag

Returerar dagen med  
ordningsnummer  
dayNr i veckan

25

## switch-satsen

En **switch**-sats är en selekteringssats med flervalsoalternativ.

Varje alternativ beskrivs av en **case**-sats; som består dels av ett **case**-uttryck, dels av de satser som anger vad som skall göras om alternativet inträffar.

I **switch**-satsen testas värdet av ett uttryck (**expression**) mot ett antal givna **case**-uttryck (*value1, value2, ...*).

Uttrycket (*expression*) måste vara en heltalstyp, typen **char** eller en uppräkningsstyp.

**case**-uttrycken måste ha samma typ som *expression*.

**case**-uttrycken måste ha olika värden.

**switch**-satsen kan innehålla ett **default**-alternativ, som avser de möjliga alternativ som inte anges i ett eget **case**-uttryck

### Syntax:

```
switch (expression) {  
    case value1:  
        statements;  
        break;  
    case value2:  
    case value3:  
        statements;  
        break;  
    ... (more cases) ...  
    default:  
        statements;  
}
```

26

## switch-satsen

Först evalueras uttrycket *expression*.

Det erhållna värdet jämförs med värdena av **case**-uttrycken.

Om det finns ett **case**-uttryck som har samma värde som *expression*, sker ett hopp till den sats som följer efter detta **case**-uttryck. Annars sker ett hopp till satsen efter **default**-alternativet (om ett sådant finns).

Exekveringen av en **case**-sats, försätter förbi andra **case**-uttryck tills **switch**-blocket är slut eller tills ett **break**-uttryck påträffas.

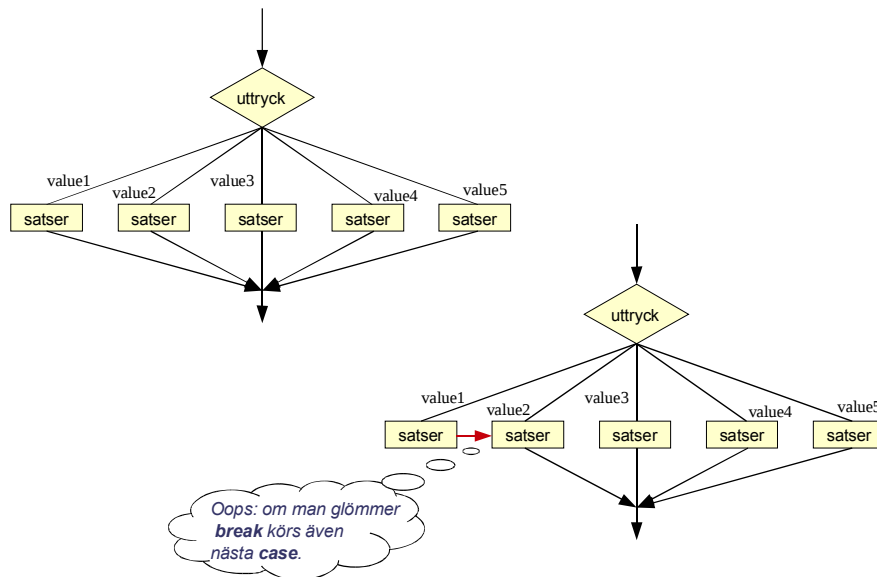
Flera **case**-uttryck kan således finnas framför ett visst "alternativ".

### Syntax:

```
switch (expression) {  
  case value1:  
    statements;  
  break;  
  case value2:  
  case value3:  
    statements;  
  break;  
  ... (more cases) ...  
  default:  
    statements;  
}
```

27

## switch-satsen: Flödesscheman



28

## switch-satsen: Exempel

```
public enum DayOfWeek {  
    MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY;  
    ...  
} // DayOfWeek
```

```
public static void tellItLikeItIs(DayOfWeek day) {  
    switch (day) {  
        case MONDAY:  
            System.out.println("Mondays are bad.");  
            break;  
        case FRIDAY:  
            System.out.println("Fridays are better.");  
            break;  
        case SATURDAY:  
        case SUNDAY:  
            System.out.println("Weekends are best.");  
            break;  
        default:  
            System.out.println("Midweek days are so-so.");  
    }  
} // tellItLikeItIs
```

Utförs för SATURDAY  
och SUNDAY

Utförs för TUESDAY,  
WEDNESDAY och  
THURSDAY

29

## Shorthand operatorer

I Java finns ett antal *shorthand* operatorer.

Dels finns operatorer för *increment* och *decrement*, både i en prefix och i en postfix version, dels finns sammansatta tilldelningsoperatorer.

| <u>Shorthand</u> | <u>Motsvarand uttryck</u> |
|------------------|---------------------------|
| ++x              | x + 1                     |
| --x              | x - 1                     |
| x++              | x + 1                     |
| x--              | x - 1                     |
| x += y           | x = x + y                 |
| x -= y           | x = x - y                 |
| x *= y           | x = x * y                 |
| x /= y           | x = x / y                 |

30

## Shorthand operatorer

Efter som operatorena ++ och -- ändrar värdet på en variabel måste man vara observant om man använder dessa operatorer i kombination med en tilldelningsoperator.

Betrakta nedanstående satser:

```
firstNumber = 10;  
secondNumber = ++firstNumber;
```

Efter att satserna har utförts har både variabeln `firstNumber` och `secondNumber` värdet 11.

När däremot följande satser exekveras

```
firstNumber = 10;  
secondNumber = firstNumber++;
```

Har variabeln `firstNumber` värdet 11 och variabeln `secondNumber` värdet 10.

Prefixoperatörn (++i) utförs *före* tilldelningsoperatörn, medan postfixoperatörn (i++) utförs *efter* tilldelningsoperatörn.

Använd shorthand operatorena med försiktighet!