Objektorienterad programmering

Föreläsning 7: flerdimensionella fält och textfiler

Dr. Alex Gerdes | Dr. Carlo A. Furia

Hösttermin 2016

Chalmers University of Technology

Sammanfattning föreläsning 6

- Endimensionella fält
- Strängar

Textfiler

In- och utmatning av data

- Utan att kunna läsa och skriva data skulle de flesta program vara ganska meningslösa
- Den data som ett program är beroende av kan t.ex.
  - ges via tangentbordet
  - finnas i en fil
  - hämtas från nätet
  - vara utdata från ett annat program
- Ett program kan också behöva skriva/skicka data till dessa enheter
- Java tillhandahåller ett flertal färdiga klasser för att underlätta I/O-hanteringen

Läsa data från tangentbordet

- I Java är tangentbordet kopplat till `System.in`, vilket är ett objekt av typen `InputStream`. För att underlätta läsningen kopplas `System.in` till ett `Scanner`-objekt.

Konstruktorer	Beskrivning
`Scanner(InputStream source)`	Constructs a new Scanner that produces values scanned from the specified input stream. Bytes from the stream are converted into characters.

```java
import java.util.Scanner;

public class ReadFromKeyboard {
  public static void main(String[] args) {
    Scanner keyboard = new Scanner(System.in);
    System.out.print("Give the integer numbers: ");
    int sum = 0;
    while (keyboard.hasNextInt()) {
      sum = sum + keyboard.nextInt();
    }
    System.out.println("The sum of the numbers: " + sum);
  }
}
```

Läsa data från textfiler

- För att läsa från en textfil använder vi klasserna `File` och `Scanner`
- Konstruktorn kastar ett en `FileNotFoundException` om den angivna filen inte finns. Detta är en s.k. kontrollerande *exception* som *måste* fångas eller kastas vidare.

Konstruktor	Beskrivning
`File(String pathname)`	Creates a new File instance by converting the given pathname string into an abstract pathname.

Konstruktor	Beskrivning
`Scanner(File source) throws FileNotFoundException`	Constructs a new `Scanner` that produces values scanned from the specified file. Bytes from the file are converted into characters.

Läsa data från textfiler

```
import java.util.Scanner;
import java.io.File;
import java.io.FileNotFoundException;

public class ReadFromTextFile {
  public static void main(String[] args) throws FileNotFoundException {
    File in = new File("indata.txt");
    Scanner sc = new Scanner(in);
    int sum = 0;
    while (sc.hasNext()) {
      sum = sum + sc.nextInt();
    }
    System.out.println("The sum is: " + sum);
  }
}
```

Kasta exception vidare

Kan resultera i FileNotFoundException

Skriva till textfiler

- För att skriva till en textfil använder vi klassen `PrintWriter`

Konstruktor	Beskrivning
`PrintWriter(String fileName)` `    throws FileNotFoundException`	Creates a new `PrintWriter`, without automatic line flushing, with the specified file name.

Operationer	Beskrivning
`void close()`	Closes the stream and releases any system resources associated with it.
`void print(int i)`	Prints an integer.
`void print(double d)`	Prints a double-precision floating-point number.
…	
`void println(int i)`	Prints an integer and then terminates the line.
…	

Skriva till textfiler

```
import java.io.PrintWriter;
import java.io.FileNotFoundException;

public class WriteToTextFile {
  public static void main(String[] args) throws FileNotFoundException {
    PrintWriter out = new PrintWriter("out.txt");
    for (int i = 0; i < 10; i = i + 1) {
      out.println(i);
    }
    out.close();
  }
}
```

Kasta exception vidare

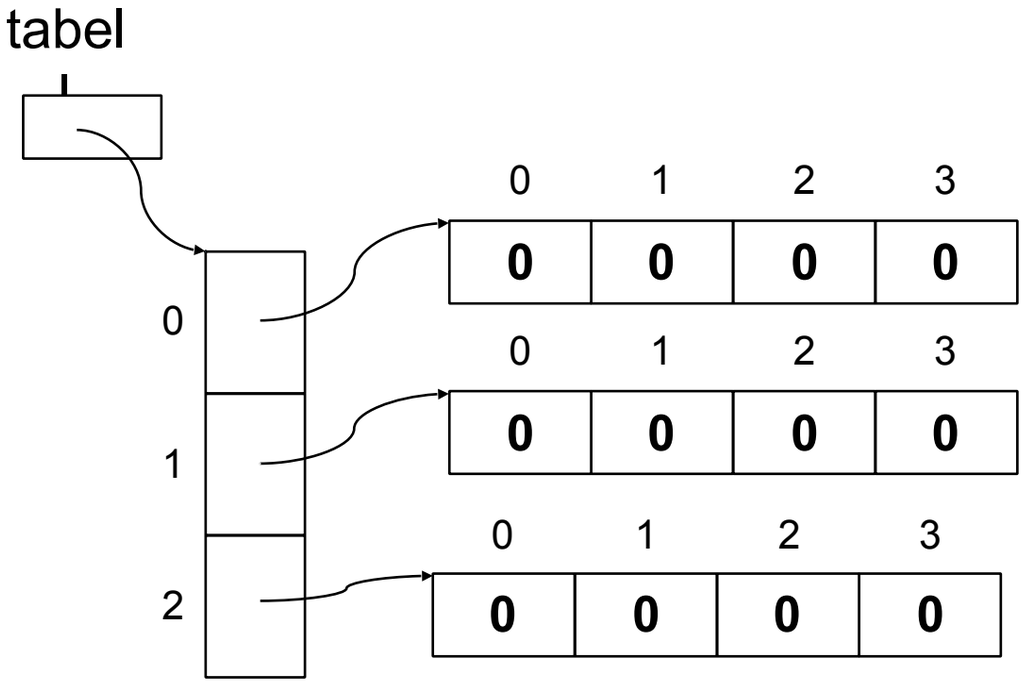
Kan resultera i `FileNotFoundException`

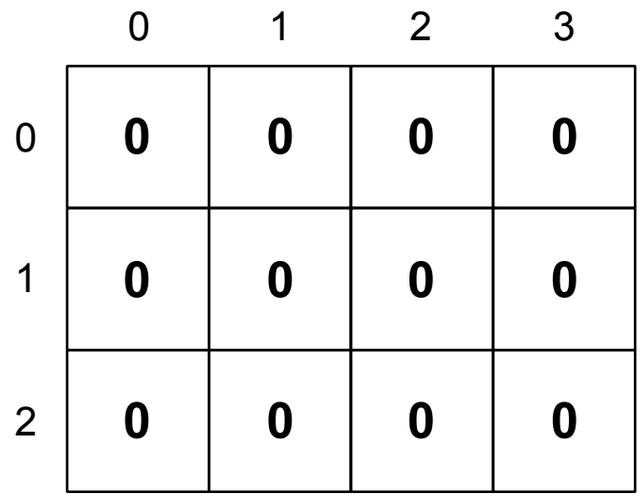
Flerdimensionella fält

Tvådimensionella fält

- Tvådimensionella fält är *fält av fält*

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





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**		

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[0])` sorterar rad 1 i `tabell`
`Arrays.sort(tabell[0])` 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

  $a_{ij} = a_{ji}$ för alla i och j

- **Analys**:
  - **Indata**: Ett gradtal samt en kvadratiskt 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

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 = true;`
  4. För varje element aij i matrisen A

     4.1. `if (`$a_{ij} \neq a_{ji}$`) okey = false;`
  5. if `okey`
     Skriv ut ”Matrisen är symmetrisk.”.
     else
     Skriv ut ”Matrisen är INTE symmetrisk.”.

- **Datarepresentation**:

  - `A` är av datatypen `double[][]`

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!");
  }
```

Implementation `readMatrix`

```
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 q = "Ge element (" + row + ", " + col + ”)";
        String input = JOptionPane.showInputDialog(q);
        theMatrix[row][col] = Double.parseDouble(input);
      }
    }
    return theMatrix;
  }
```

Implementation `isSymmetric`

CHALMERS

```
// 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;
  }
}
```

Alternativ implementation `readMatrix`

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;
}
```

Alternativ implementation `readMatrix`

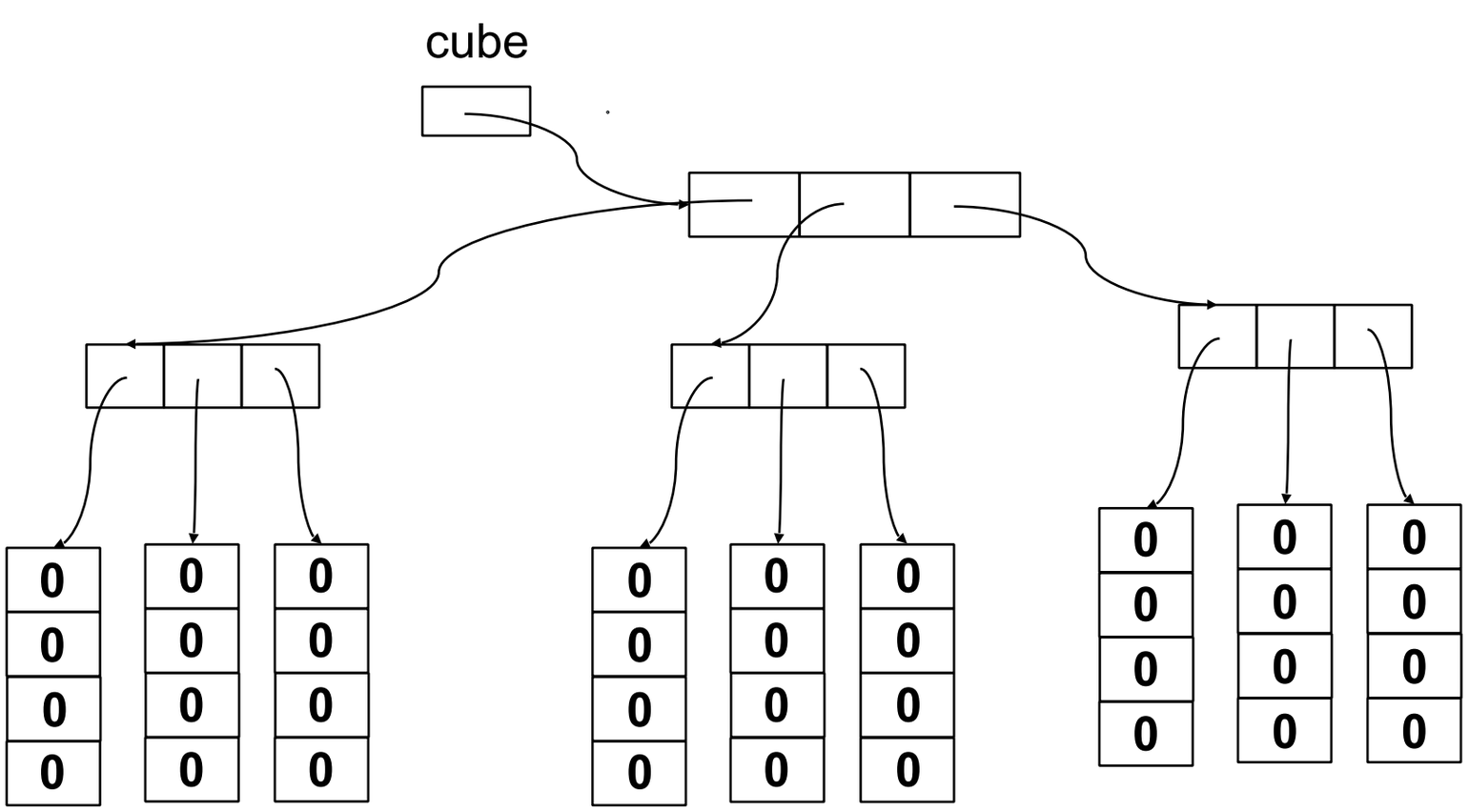
Användning av ett `Scanner`-objekt

```
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;
}
```

Flerdimensionella fält

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

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



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];
```

Paus

ArrayList

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 all 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`
`void add(E elem)`	lägger in `elem` sist i listan (d.v.s. efter de element som redan finns i listan).
`void add(int pos, E elem)`	lägger in `elem` på plats `pos`. Efterföljande element flyttas ett position framåt i listan.
…	…
`E get(int pos)`	returnerar elementet på plats `pos`
`E set(int pos, E elem)`	ersätter elementet på plats `pos` med `elem`, returnerar elementet som fanns på platsen `pos`
`E remove(int pos)`	tar bort elementet på plats `pos`, returnerar det borttagna elementet. Efterföljande element i listan flyttas en position bakåt i listan.

Klassen `ArrayList<E>`

Metod	Beskrivning
`int size()`	returnerar antalet element i listan
`boolean isEmpty()`	returnerar `true` om listan är tom, annars returneras `false`
`int indexOf(E elem)`	returnerar index för elementet `elem` om detta finns i listan, annars returneras -1
`boolean contains(Object elem)`	returnerar `true` om `elem` finns i listan, annars returneras `false`
`void clear()`	tar bort alla elementen i listan
`String toString()`	returnerar en textrepresentation på formen $[e_1, e_2, \ldots, e_n]$

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`.

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 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*

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
```

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

```
double[] values = new double[100];
ArrayList<String> listan = new ArrayList<String>();

// Genomlöpning av hela samlingarna med den vanliga for-satsen
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));
}

// Genomlöpning av hela samlingarna med den förenklade for-satsen
for (double v : values)
  System.out.println(v);

for (String str : listan)
  System.out.println(str);
```

Problemexempel

- Skriv en metod

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

  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, 3

Analys och implementation

- **Algoritm**:
  1. while (fler tal att läsa)
     1.1. läs talet
     1.2. if (talet inte finns i listan)
          lagra talet i listan;
  2. returnera listan

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

Klassen `PhoneBook` implementerad med fält

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

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

Maximal storlek måste anges

Antalet element måste bokföras

Exekveringsfel om count >= size

Klassen `PhoneBook` implementerad med `ArrayList`

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

```java
import java.util.ArrayList;

public class PhoneBook {
  private ArrayList<Entry> book = new ArrayList<Entry>();

  public void put(String name, String nr) {
    book.add(new Entry(name, nr));
  }

  public String get(String name) {
    String res = null;
    for (Entry e : book)
      if (name.equals(e.getName()))
        res = e.getNumber();
    return res;
  }
}
```

Shorthand operatorer

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*

Shorthand	***Motsvarand uttryck***
`++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`

Shorthand operatorer

- Efter som operatorerna ++ 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 `fistNumber` och `secondNumber` värdet 11

- När däremot följande satser exekveras

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

  har variabeln `fistNumber` värdet 11 och variabeln `secondNumber` värdet 10

- Prefixoperatorn (`++i`) utförs *före* tilldelningsoperatorn, medan postfixoperatorn (`i++`) utförs *efter* tilldelningsoperatorn

- Använd shorthand operatorerna med med försiktighet!

Live coding!