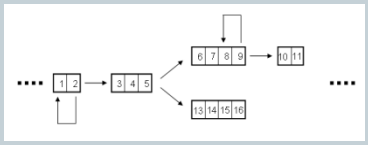
Freie Universität Berlin

Kurs: Programmieren in Java

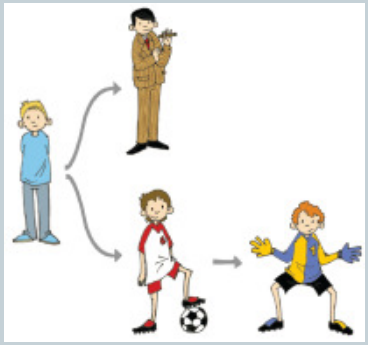
Tag 4



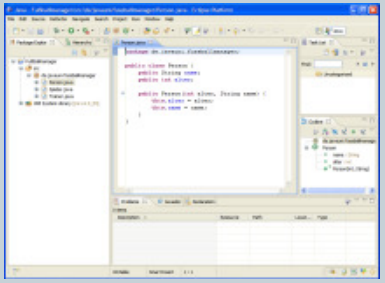
GRUNDLAGEN

GRAFIKKONZEPTE
BILDVERARBEITUNG
MUSTERERKENNUNG

KI UND SPIELE-
PROGRAMMIERUNG



OBJEKTORIENTIERTE
PROGRAMMIERUNG



ENTWICKLUNGS-
UMGEBUNGEN

Praktische Beispiele

Inhalt:

- Kryptographie
  - Verschiebung (Caesar)
  - XOR-Codierung
- Verwendung von Zufallszahlen
- Dynamischer Datentyp Vector
- Bibliotheken verwenden und erstellen

Ratz D., et.al.: „*Grundkurs Programmieren in Java*“, 4.Auflage, Hanser-Verlag 2007
Block M.: "*Java-Intensivkurs - In 14 Tagen lernen Projekte erfolgreich zu realisieren*“, Springer-Verlag 2007

Praktische Beispiele

Abstrakte Encoder-Klasse

Zwei Methoden werden bei der Verschlüsselung benötigt:

```
public abstract class Encoder {
    // verschlüsselt einen String
   public abstract String encode(String s);

   // entschlüsselt einen String
   public abstract String decode(String s);
}
```

Caesar-Codierung I

Durch Verschiebung (Schlüssel) der Buchstaben im Alphabet kann ein Text codiert und wieder decodiert werden:

```
public class CAESAR_Codierer extends Encoder{
  // geheimer Schlüssel
  private int key;

  // Definition des Alphabets
  public static final int ALPHABETSIZE = 26;
  public static final char[] alpha =
            {'A','B','C','D','E','F','G','H','I',
             'J','K','L','M', 'N','O','P','Q','R',
             'S','T','U','V','W','X','Y','Z'};

  // Übersetzungslisten für die Buchstaben
  protected char[] encrypt = new char[ALPHABETSIZE];
  protected char[] decrypt = new char[ALPHABETSIZE];

  public CAESAR_Codierer(int key) {
    this.key = key;
   computeNewAlphabet();
  }

  private void computeNewAlphabet(){
    for (int i=0; i<ALPHABETSIZE; i++)
      encrypt[i] = alpha[(i + key) % ALPHABETSIZE];

    for (int i=0; i<ALPHABETSIZE; i++)
      decrypt[encrypt[i] - 'A'] = alpha[i];
  }
```

Caesar-Codierung II

Jetzt lassen sich die Methoden encode und decode sehr einfach implementieren:

```
  // ************************************************************************
  // Encoder-Funktionen
  public String encode(String s) {
    char[] c = s.toCharArray();

    for (int i=0; i<c.length; i++)
      if (Character.isUpperCase(c[i]))
        c[i] = encrypt[c[i] - 'A'];

    return new String(c);
  }

  public String decode(String s){
    char[] c = s.toCharArray();

    for (int i=0; i<c.length; i++)
      if (Character.isUpperCase(c[i]))
        c[i] = decrypt[c[i] - 'A'];

    return new String(c);
  }
  // ************************************************************************
}
```

XOR-Verschlüsselung I

Jetzt lernen wir das XOR kennen und verwenden es gleich, um einen Text zu verschlüsseln:

B1	B2	B1 XOR B2
0	0	0
0	1	1
1	0	1
1	1	0

In Java wird das XOR durch den Operator „^“ repräsentiert.

```
boolean a, b, c;
a = true;
b = false;
c = a ^ b;
```

XOR-Verschlüsselung II

Alle Zeichen werden über die binäre Operation xor (exklusives Oder) verknüpft. Dabei werden alle Zeichen als binäre Zahlen aufgefasst:

```
public class XOR_Codierer extends Encoder{
   // hier wird der geheime Schlüssel abgelegt
   private int key;

   public XOR_Codierer(int k){
      key = k;
   }

   // verschlüsselt durch XOR-Operation mit key
   // die Zeichen der Zeichenkette s
   public String encode(String s) {
      char[] c = s.toCharArray();

      for (int i=0; i<c.length; i++)
         c[i] = (char)(c[i]^key);

      return new String(c);
   }

   // entschlüsselt mit Hilfe der Funktion
   // encode die Zeichenkette s
   public String decode(String s){
      return encode(s);
   }
}
```

XOR-Verschlüsselung III

Jetzt wollen wir den Encoder testen:

```
public class Demo{
   public static void demo(Encoder enc, String text) {
      String encoded = enc.encode(text);
      System.out.println("codiert  : " + encoded);
      String decoded = enc.decode(encoded);
      System.out.println("decodiert: " + decoded);

      if (text.equals(decoded))
         System.out.println("Verschluesselung erfolgreich!");
      else
         System.out.println("PROGRAMMFEHLER!");
   }

   public static void main(String[] args){
      int key     = 1;
      String text = "";
      try{
         key   = Integer.parseInt(args[0]);
         text  = args[1];
      } catch(Exception e){
         System.out.println("Fehler ist aufgetreten!”);
         System.out.println(“Bitte nochmal Demo <int> <String> aufrufen.");
      }

      Encoder enc = new XOR_Codierer(key);
      demo(enc, text);
   }
}
```

Praktische Beispiele

Zufallszahlen

Es gibt verschiedene Möglichkeiten Zufallszahlen zu verwenden. Oft benötigt man sie als Wahrscheinlichkeitsmaß im Intervall [0,1]. In anderen Fällen ist es wünschenswert aus einer Menge A mit n Elementen eines auszuwählen {1, 2, ..., n}.

Wir unterscheiden zunächst einmal den Datentyp der Zufallszahl. In jedem Fall verwenden wir die Klasse aus dem Package java.util.

```
import java.util.Random;
...
```

Um eine der Klassenmethoden verwenden zu können, erzeugen wir eine Instanz der Klasse **Random**:

```
...
Random randomGenerator = new Random();
...
```

Praktische Beispiele

Ganzzahlige Zufallszahlen vom Typ int und long

Das kleine Lottoprogramm (6 aus 49) dient als Beispiel für die Erzeugung der Funktion *nextInt(n)*. Es werden Zufallszahlen aus dem Bereich [0, 1, ..., n-1] gewählt. Wenn Zahlen aus dem Bereich long benötigt werden, so kann die Funktion *nextLong(n)* analog verwendet werden.

```
import java.util.*;

public class Lotto {
   public static void main(String[] args)
   {
      Random rg = new Random();
      int[] zuf = new int[6];

      System.out.print("Lottotipp (6 aus 49): ");
      int wert, i=0;

      aussen:
      while(i<6){
         wert = rg.nextInt(49) + 1; // +1 da nicht 0,...,48 sondern 1,...,49

         // schon vorhanden?
         for (int j=0; j < i; j++)
            if (zuf[j]==wert)
               continue aussen;

         zuf[i] = wert;
         i++;
         System.out.print(wert + " ");
      }
   }
}
```

Praktische Beispiele

Ganzzahlige Zufallszahlen vom Typ float und double

Für die Erzeugung einer Zufallszahl aus dem Intervall [0,1] gibt es eine kürzere Schreibweise. In der Klasse **Math** im Package java.lang gibt es eine statische Funktion *random*, die eine Instanz der Klasse **Random** erzeugt, die Funktion *nextDouble* aufruft und den erzeugten Wert zurückliefert.

Wir schreiben lediglich die folgende Zeile:

```
double zuffi = Math.random();
```

Bei der Initialisierung der Klasse **Random** gibt es zwei Varianten. Die erste mit dem parameterlosen Konstruktor initialisiert sich in Abhängigkeit zur Systemzeit und erzeugt bei jedem Start neue Zufallszahlen. Für Programme, bei denen beispielsweise zeitkritische Abschnitte getestet werden, die aber abhängig von der jeweiligen Zufallszahl sind oder Experimente, bei denen die gleichen Stichproben verwendet werden sollen, ist der Konstruktor mit einem long als Parameter gedacht.

Wir können beispielsweise einen long mit dem Wert 0 immer als Startwert nehmen und erhalten anschließend immer dieselben Zufallszahlen:

```
long initwert          = 0;
Random randomGenerator = new Random(initwert);
```

Praktische Beispiele

Der dynamische Datentyp Vector I

Im Gegensatz zu einem Array, bei dem die Anzahl der Elemente bei der Initialisierung festgelegt wird, verhält sich der von Java angebotene Datentyp Vector dynamisch. Wenn wir also vor der Verwendung einer Liste die Anzahl der Elemente nicht kennen, können wir diesen Datentyp nehmen.

Ein kleines Beispiel dazu:

```
import java.util.Vector;
public class VectorTest{
   public static void main(String[] args){
      Vector v = new Vector();
      for (int i=0; i<4; i++) // füge nacheinander Elemente in den Vector ein
         v.addElement(new Integer(i));
      System.out.println("Vector size = "+v.size()); // Anzahl der Elemente im Vector v

      // Auslesen der aktuellen Inhalts
      for (int i=0; i<v.size(); i++){
         Integer intObjekt = (Integer)v.elementAt(i);
         int wert = intObjekt.intValue();
         System.out.println("Element "+i+" = "+wert);
      }
      System.out.println();
      v.insertElementAt(new Integer(9), 2); // wir geben ein neues Element hinzu
      v.removeElementAt(4); // und löschen ein Element

      for (int i=0; i<v.size(); i++) // Auslesen der aktuellen Inhalts
         System.out.println("Element "+i+" = "
                     +((Integer)v.elementAt(i)).intValue());
   }
}
```

Der dynamische Datentyp Vector II

Unser Beispiel liefert folgende Ausgabe:

```
C:\JavaCode>java VectorTest
Vector size = 4
Element 0 = 0
Element 1 = 1
Element 2 = 2
Element 3 = 3

Element 0 = 0
Element 1 = 1
Element 2 = 9
Element 3 = 2
```

Lineare Algebra I

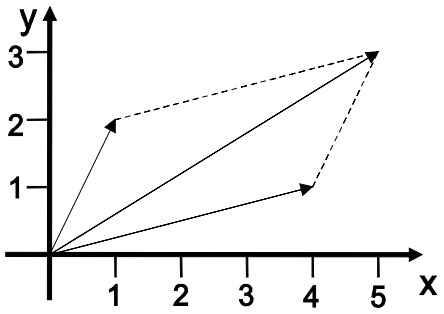
Es gibt viele nützliche Bibliotheken, die wir verwenden können. JAMA ist beispielsweise die meist verwendete Bibliothek für Methoden der Linearen Algebra. Hier ein Beispiel zur Vektoraddition:

```
import Jama.*;
public class JAMATest{
   public static void main(String[] args){
      double[][] vector1 = {{1},{2}};
      double[][] vector2 = {{4},{1}};

      Matrix v1 = new Matrix(vector1);
      Matrix v2 = new Matrix(vector2);

      Matrix x  = v1.plus(v2);

      System.out.println("Matrix x: ");
      x.print(1, 2);
   }
}
```



Praktische Beispiele

Lineare Algebra II

Nun wollen wir die Determinante einer Matrix berechnen.

```
import Jama.*;
public class JAMATest{
   public static void main(String[] args){
      double[][] array = {{-2,1},{0,4}};
      Matrix a = new Matrix(array);
      double d = a.det();

      System.out.println("Matrix a: ");
      a.print(1, 2);

      System.out.println(“Determinante: ” + d);
   }
}
```

det(a) = (-2)*4 – 1*0 = -8

Installation der JAMA-Bibliothek

Um JAMA zu installieren, gehen wir zunächst zu dem aufgeführten Link:
http://math.nist.gov/javanumerics/jama/

Wir speichern nun das Zip-Archiv vom Source (zip archive, 105Kb) z.B. in den Ordner "c:\Java\". Jetzt ist das Problem, dass die bereits erstellten .class-Dateien nicht zu unserem System passen. Desshalb müssen die sechs ".class" Dateien im Ordner *Jama* löschen und zusätzlich das Class-File im Ordner *util*. Jetzt ist es quasi clean.

Nun gehe in den Ordner "c:\Java" und gebe folgendes ein:

```
C:\Java>javac Jama/Matrix.java
  Note: Jama\Matrix.java uses unchecked or unsafe operations.
  Note: Recompile with -Xlint:unchecked for details.J
```

Jetzt wurde das Package erfolgreich compilert.

Eine eigene Bibliothek bauen I

Die Erzeugung eines eigenen Packages unter Java ist sehr einfach. Wichtig ist das Zusammenspiel aus Klassennamen und Verzeichnisstruktur. Angenommen wir wollen eine Klasse **MeinMax** in einem Package meinMathe anbieten.

Dann legen wir ein Verzeichnis meinMathe an und speichern dort z.B. die folgende Klasse:

```
package meinMathe;

public class MeinMax{
   public static int maxi(int a, int b){
      if (a<b) return b;
      return a;
   }
}
```

Durch das Schlüsselwort package haben wir signalisiert, dass es sich um eine Klasse des Package meinMathe handelt. Unsere kleine Matheklasse bietet eine bescheidene *maxi*-Funktion.

Nachdem wir diese Klasse mit javac compiliert haben, können wir ausserhalb des Ordners eine neue Klasse schreiben, die dieses Package jetzt verwendet.

Eine eigene Bibliothek bauen II

Dabei ist darauf zu achten, dass der Ordner des neuen Packages entweder im gleichen Ordner wie die Klasse liegt, die das Package verwendet, oder dieser Ordner im PATH aufgelistet ist.

Hier unsere Testklasse:

```
import MeinMathe.MeinMax;

public class MatheTester{
   public static void main(String[] args){
      System.out.println("Ergebnis = "+MeinMax.maxi(3,9));
   }
}
```

Wir erhalten nach der Ausführung folgende Ausgabe:

```
C:\JavaCode>java MatheTester
Ergebnis = 9
```

Praktische Beispiele

Vielen Dank fürs Zuhören!

Das genügt erstmal für heute ...

Block M.: "*Java-Intensivkurs - In 14 Tagen lernen Projekte erfolgreich zu realisieren*", Springer-Verlag 2007