Algorithmen und Datenstrukturen
Beschreibung von Algorithmen

Matthias Teschner
Graphische Datenverarbeitung
Institut für Informatik
Universität Freiburg

SS 12

Lernziele der Vorlesung

- Algorithmen
  - Sortieren, Suchen, Optimieren
- Datenstrukturen
  - Repräsentation von Daten
  - Listen, Stapel, Schlangen, Bäume
- Techniken zum Entwurf von Algorithmen
  - Algorithmenmuster
  - Greedy, Backtracking, Divide-and-Conquer
- Analyse von Algorithmen
  - Korrektheit, Effizienz

Motivation

- kurze Einführung zur Beschreibung von Algorithmen
  - Beispiel
  - Pseudocode
  - Java (Programmierumgebung, Datentypen, Kontrollstrukturen)
- Konzept der Entwicklungsumgebung
- Vorstellung wichtiger Java-Konzepte
- Hinweise zu weiterführenden Veranstaltungen
- Anregung zur Eigeninitiative

Motivation

- Betriebssysteme, Programmiersprachen und Entwicklungsumgebungen sind wichtige Werkzeuge in vielen Informatik-Veranstaltungen.
- Wissen in diesen Bereich möglichst vielseitig und selbständig erwerben
- nicht zuviel Spezialwissen in einer Programmiersprache / Entwicklungsumgebung
- Alle Betriebssysteme, Programmiersprachen, Entwicklungsumgebungen und Computer- bzw. Mac-Hersteller sind gleich.

Motivation

- unterschiedliche, sich teilweise überlappende Aspekte
  - Algorithmen und Datenstrukturen
  - Softwaretechnik
  - Programmiersprachen

Überblick

- Beispielprogramm
- Pseudocode
- Java
  - Programmierumgebung
  - Datentypen
  - Operatoren, Kontrollstrukturen
- weiterführende Quellen

Maximum-Algorithmus in Pseudocode

- Eingabe: Folge von Zahlen "eingabe"
- Ausgabe: Größtes Element von "eingabe"

```
maximum(eingabe)
    max=eingabe[1]
    for index=2 to länge(eingabe) do
        if max<eingabe[index] then
            max=eingabe[index]
    return max
```

- Konzentration auf das Wesentliche des Algorithmus
- keine wohldefinierte Notation
- keine Fehlerbehandlung, Abstraktion, Modularität

in Java

Datentyp der Ausgabe | Name der Funktion | Datentyp der Eingabe

```
int maximum(int[] eingabe)
{
      int max = eingabe[0];

      for (int index = 1;
           index < eingabe.length;
           index = index+1)
      {
           if (max < eingabe[index])
                max = eingabe[index];
      }

      return max;
}
```

Schleife mit
- Initialisierung
- Abbruchbedingung
- Inkrement

Rückgabewert

in C

```
int maximum(int eingabe[], int feldGroesse)
{
     int max = eingabe[0];

     for (int index = 1;
          index < feldGroesse;
          index = index+1)
     {
          if (max < eingabe[index])
               max = eingabe[index];
     }

     return max;
}
```

in C++

```
int maximum(int eingabe[], int feldGroesse)
{
     int max = eingabe[0];

     for (int index = 1;
          index < feldGroesse;
          index = index+1)
     {
          if (max < eingabe[index])
               max = eingabe[index];
     }

     return max;
}
```

Einbettung in ein Programm
Java

```
class Max
{
        // maximum gibt die groesste Zahl eines Feldes zurueck
        static int maximum(int[] eingabe)
        {
                …
        }

        public static void main(String[] args)
        {
                // Feld erzeugen
                int zahlen[] = {1, 4, 5, 2, 0};

                // groesste Zahl im Feld suchen
                int ergebnis = maximum(zahlen);

                // Ergebnis ausgeben
                System.out.println("Maximum: " + ergebnis);
        }
}
```

Einbettung in ein Programm
C / C++

```
#include <stdio.h>

// maximum gibt die groesste Zahl eines Feldes zurueck
int maximum(int eingabe[], int feldGroesse)
{
        …
}

void main( int argc, const char* argv[] )
{
        // Feld erzeugen
        int zahlen[] = {1, 4, 5, 2, 0};
        int feldGroesse = sizeof(zahlen) / sizeof(zahlen[0]);

        // groesste Zahl im Feld suchen
        int ergebnis = maximum(zahlen, feldGroesse);

        // Ergebnis ausgeben
        printf("Maximum: %d", ergebnis);
}
```

Überblick

- Beispielprogramm
- Pseudocode
- Java
  - Programmierumgebung
  - Datentypen
  - Operatoren, Kontrollstrukturen
- weiterführende Quellen

Imperative Programmierung

- Programmierparadigma
- Beschreibung einer Berechnung durch Folge von Anweisungen, die den Status des Programms (der Eingabemenge) verändern
- Beispiele: Fortran, Pascal, C / C++, Java
- teilweise prozedural (C) oder objektorientiert (C++, Java)
  - Datenkapselung (information hiding) durch Zusammenfassung von Prozeduren in Paketen oder Modulen
  - Verbergen von konkreten Realisierungen (Übersichtlichkeit)
  - Zugriff auf Datenstrukturen nur über definierte Schnittstellen (Fehlervermeidung)

Bausteine für Algorithmen

- elementare Operationen
  - werden nicht weiter aufgeschlüsselt
  - "bringe Wasser zum Kochen;"
- Sequenzen
  - Hintereinanderausführen elementarer Operationen
  - "bringe Wasser zum Kochen; dann gib Nudeln dazu;"
- bedingte Ausführungen
  - elementare Operation wird nur ausgeführt, wenn Bedingung erfüllt ist
  - "wenn die Soße zu dünn ist, dann füge Mehl dazu;"
- Schleifen
  - Wiederholung einer elementaren Operation, bis Bedingung erfüllt ist
  - "rühre, bis die Soße braun ist;"

Bausteine für Algorithmen

- Unterprogramm
  - Zusammenfassung von Operationen zu einer Einheit mit aussagekräftiger Bezeichnung
  - Ausführung innerhalb eines Algorithmus durch Verwendung der Bezeichnung
  - "bereite Soße nach Rezept auf Seite 42;"
- Rekursion
  - Anwendung derselben Berechnungsvorschrift auf kleinere Teilprobleme, (bis das Teilproblem direkt gelöst werden kann)
  - "schneide Fleischstück:
    wenn Fleischstück größer 2cm,
    dann vierteile Fleischstück; schneide jedes neue Fleischstück;
    wenn Fleischstück nicht größer 2cm,
    dann fertig;"

Pseudocode

- festgelegte Notation für Bausteine (Schlüsselworte)

- `if Bedingung then A; else B;`
- `for (Menge von Elementen) do A;`
- `while Bedingung do {A; B; C;}`
-
```
repeat
    begin
    A; B; C; D;
    end
until Bedingung;
```

Unterprogramm

- ```
  procedure koche Kaffee
  begin
      koche Wasser;
      gib Kaffeepulver in Tasse;
      fülle Wasser in Tasse;
  end
  ```

- ```
  procedure gib Kaffeepulver in Tasse
  begin
      öffne Kaffeeglas;
      entnehme Löffel voll Kaffee;
      kippe Löffel in Tasse;
      schließe Kaffeeglas;
  end
  ```

Rekursion

```
procedure fakultät (n)
{
    if n<=1
        then return 1;
        else return n*fakultät(n-1);
}
```

Überblick

- Beispielprogramm
- Pseudocode
- Java
  - Programmierumgebung
  - Datentypen
  - Operatoren, Kontrollstrukturen
- weiterführende Quellen

Geschichte

- seit 1991 im Auftrag von Sun Microsystems unter dem Namen Oak entwickelt
- 1993 Umbenennung in Java
- seit 1995 Entwicklung von Java durch Sun Microsystems
- 2010 Übernahme von Sun Microsystems durch Oracle
- Entwurfsziele
  - Plattformunabhängigkeit durch Verwendung von Zwischencode
  - interpretative Ausführung des Zwischencode,
    Programme in kompilierter Form portabel
  - Objektorientierung
  - Anlehnung an C / C++

Werkzeuge

- Texteditor zur Erstellung von Java-Quellcode
  - notepad, vi, emacs
- Compiler zur Erstellung von Java-Bytecode (oder Nativecode)
  - Teil des plattformabhängigen Software Development Kit SDK
  - wird nur zur Entwicklung benötigt, nicht zur Ausführung
- Laufzeitumgebung zur Ausführung von Java-Bytecode
  - plattformabhängiges Java Runtime Environment JRE
  - wird zur Ausführung benötigt, nicht ausreichend für Entwicklung
  - in der Regel in SDK enthalten
- Java ist eingetragenes Warenzeichen der Oracle Corporation.
  - Oracle stellt SDK und JRE für verschiedene Plattformen bereit (Windows, Linux, Solaris).
  - Apple stellt Werkzeuge für Mac OS bereit.

Java Virtual Machine JVM

- Teil der Java-Laufzeitumgebung JRE
- Abstraktionsschicht zwischen Java-Programm (Bytecode) und Betriebssystem (Maschinencocde)
- Compiler des SDK (`javac`) erzeugt keinen ausführbaren Maschinencode, sondern Bytecode, der von der JVM des JRE (`java`) interpretiert werden kann.

plattform-spezifisch

plattform-spezifisch

Quelltext → `javac` → Bytecode → `java` → ausführbarer Maschinencode

plattform-unabhängig

.java

.class

Integrierte Entwicklungsumgebungen

- Integrated Development Environment IDE
- z. B. Eclipse
  - quelloffen
  - www.eclipse.org
  - Windows, Linux, Mac OS, …
  - Java, C, C++, …
  - baut auf SDK und JRE auf

Überblick

- Beispielprogramm
- Pseudocode
- Java
  - Programmierumgebung
  - Datentypen
  - Operatoren, Kontrollstrukturen
- weiterführende Quellen

Datentypen

- byte, short, int, long, float, double, boolean, char

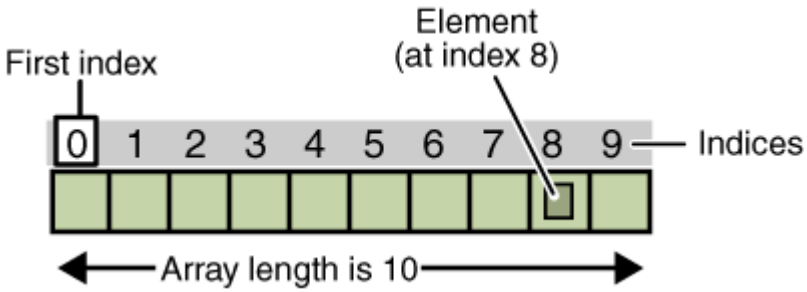
```
boolean result = true;
char capitalC = 'C';
byte b = 100;
short s = 10000;
int i = 100000;
double d1 = 123.4;
double d2 = 1.234e2; // Kommentar
int hexVal = 0x1a;   // hexadezimal
```

Datentypen - Feld

- Datenstruktur fester Größe, die aus Elementen gleichen Typs besteht

```
int[] einFeld;
einFeld = new int[10];

einFeld[0]=100;
einFeld[9]=900;
```

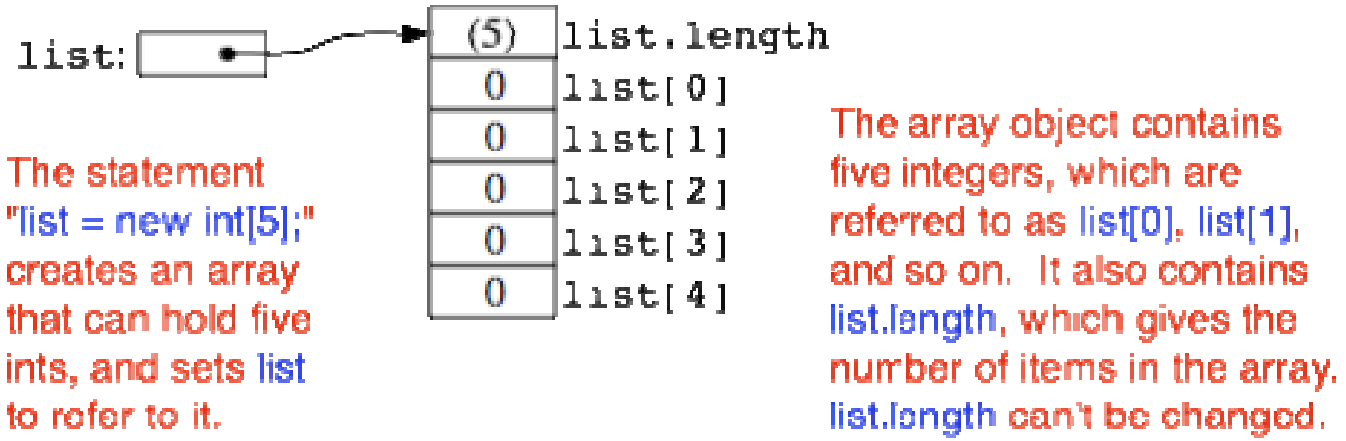


-
```
boolean[] anArrayOfBooleans;
double[] anArrayOfDoubles;
```

Datentypen - Feld

- Referenzdatentyp

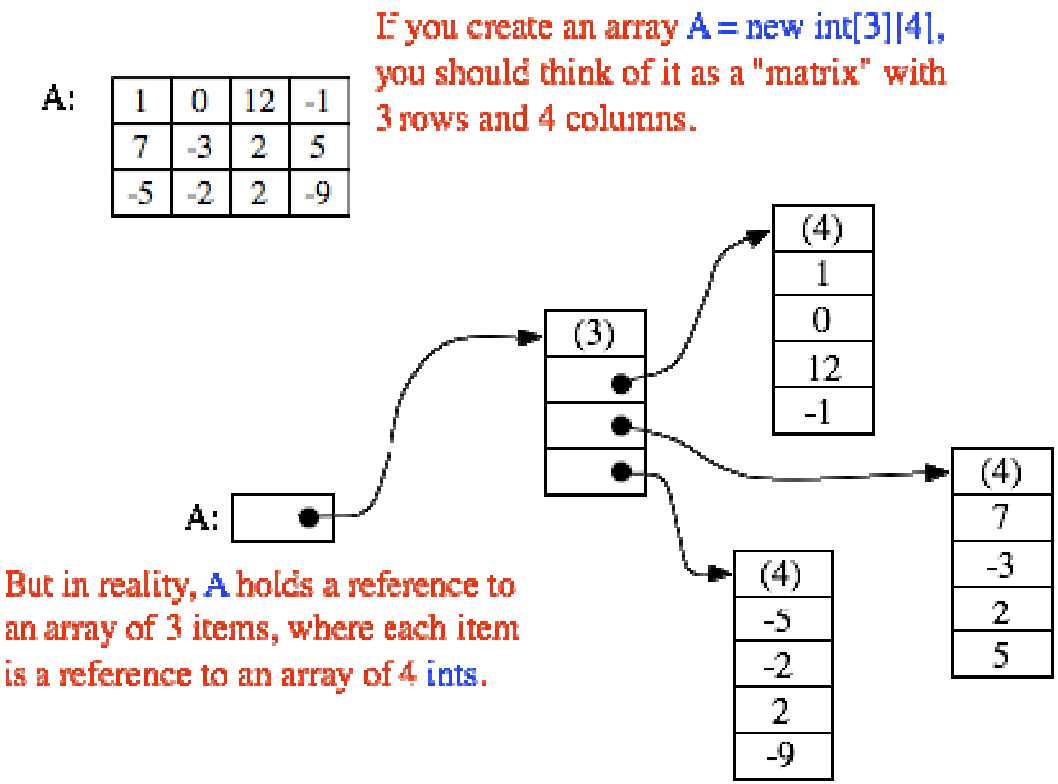
```
int[] list;          // list=null
list = new int[5]; // list=Adresse des Feldes
```



http://math.hws.edu/javanotes/c7/s1.html

Datentypen – mehrdimensionales Feld

- `int[][] A;`
  `A = new int[3][4];`



http://math.hws.edu/javanotes/c7/s1.html

Klassen

- Datentyp, der aus Menge von Attributen und Methoden besteht

```
class Bicycle {

    int cadence = 0;
    int speed = 0;
    int gear = 1;

    void changeCadence (int newValue);
    void changeGear (int newValue);
    void speedUp (int increment);
    void applyBrakes (int decrement);
    void printStates ();
}
```

Status
- beschrieben durch Attribute

Verhalten
- beschrieben durch Methoden
- Zugriff auf Attribute typischer-weise durch Methoden

Klassen

- Verwendung

```
Bicycle bike1;
```

Referenz: bike1 = null. Es wurde kein Objekt / Instanz der Klasse Bicycle generiert!

```
bike1 = new Bicycle();
```

Referenz: bike1 = Adresse einer Instanz der Klasse Bicycle. Objekt / Instanz der Klasse Bicycle wird generiert.

```
// wende Methoden auf das Objekt an

bike1.changeCadence(50);
bike1.speedUp(10);
bike1.changeGear(2);
bike1.printStates();
```

Zugriff auf Attribute der durch die Referenz bike1 definierten Instanz durch Methoden

Klassen

```
class Student {
    String name;                        // Name
    double test1, test2, test3;         // Noten
    double getAverage();                // berechne Durchschnitt
 }

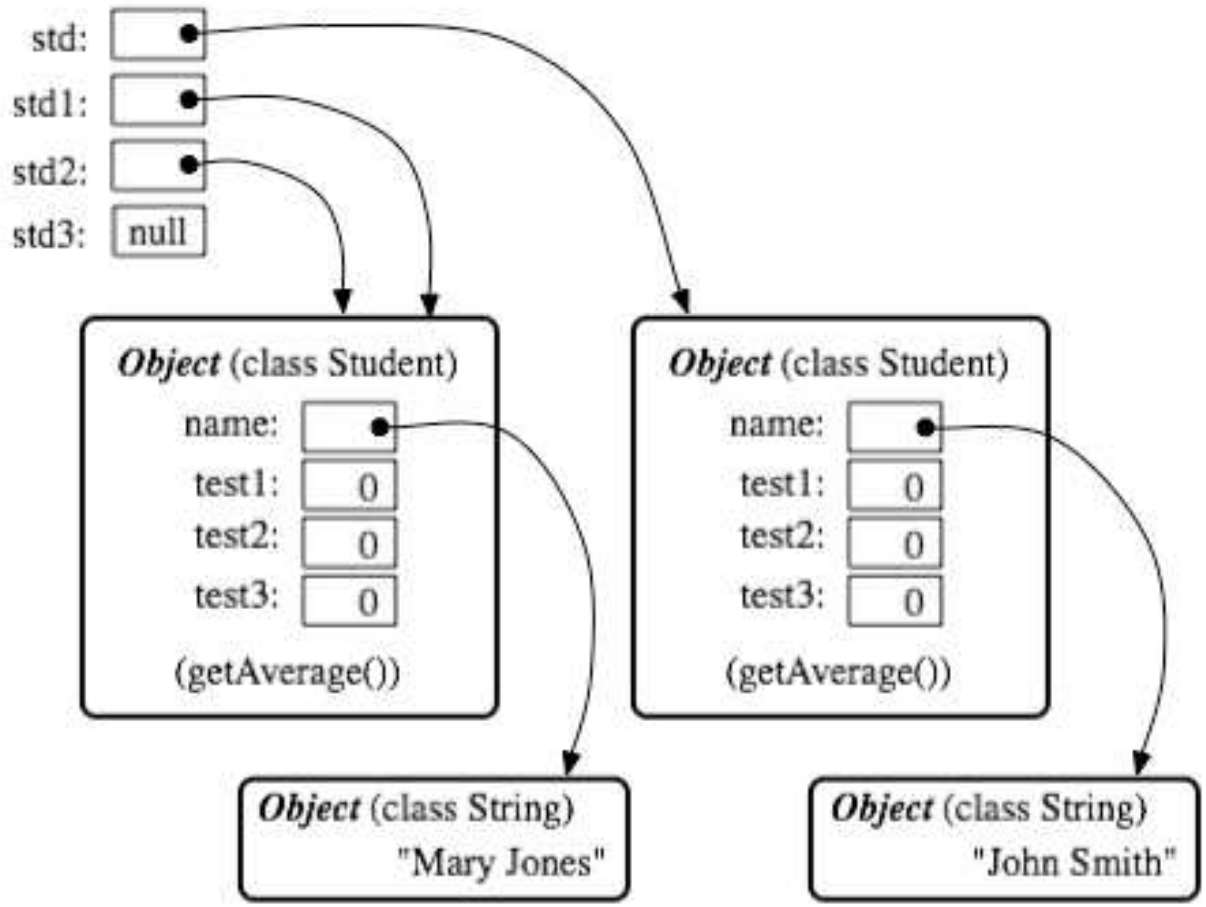
Student std, std1,          // Deklariere vier Variablen
        std2, std3;         // vom Typ Student.

std = new Student();        // Generiere ein Objekt der Klasse Student
                            // und speichere eine Referenz zu
                            // dem Objekt in der Variable std
std1 = new Student();       // Generiere ein zweites Objekt
                            // und speichere die Referenz in std1

std2 = std1;                // Kopiere die Referenz std1 nach std2
                            // Das Objekt wird nicht kopiert!
std3 = null;                // Setze die Referenz auf null

std.name = "John Smith";    // setze Attribute einer Instanz bzw.
                            // eines Objektes
std1.name = "Mary Jones";
```

Klassen



http://math.hws.edu/javanotes/c5/s1.html

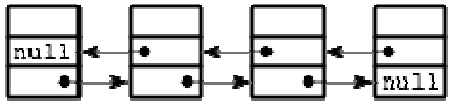
Verkettete Datenstrukturen

- rekursive Definition von Datentypen

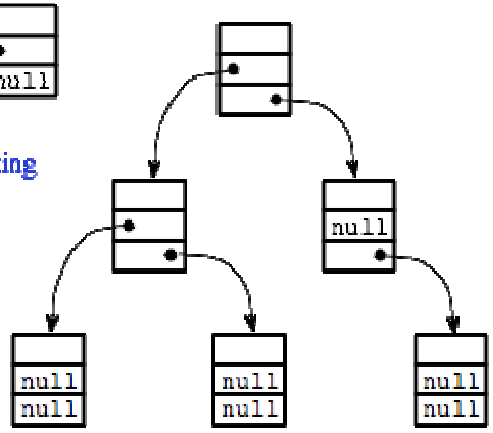
```
class Node { String item; Node next; }
```

When an object contains a reference to an object of the same type, then several objects can be linked together into a list. Each object refers to the next object.

```
class Node2 { int key; Node2 left, right; }
```



Things get even more interesting when an object contains two references to objects of the same type. In that case, more complicated data structures can be constructed.



http://math.hws.edu/javanotes/c9/s2.html

Verkettete Datenstrukturen

- Beispiel

```
class Employee {
   String name; // Name des Mitarbeiters
   Employee supervisor; // Name des Chefs
}

Employee Klaus = new Employee();
Employee Martin = new Employee();
Employee Alina = new Employee();

Klaus.supervisor = Martin;
Martin.supervisor = Alina;
Alina.supervisor = null;
```

Verkettete Datenstrukturen

- Beispiel

```
Employee emp = new Employee();

…

if ( emp.supervisor == null)
{
   System.out.println( emp.name + " hat keinen Chef." );
}
else
{
   System.out.print( "Der Chef von " +emp.name+ " ist " );
   System.out.println( emp.supervisor.name );
}
```

Verkettete Datenstrukturen

- Beispiel

```
Employee emp = new Employee();
Employee pointer;     // nur eine Referenz, kein Objekt
…
pointer = emp.supervisor;

if ( pointer.supervisor == null)
   System.out.println( emp.name + " ist der Chef." );
else {
   int count = 0;
   while ( pointer.supervisor != null ) {
      count++; // Zähle Hierarchiestufen
      pointer = pointer.supervisor; // nächste Stufe
   }
   System.out.println(count + " Stufen sind zwischen " +
   emp.name + " und dem Chef." );
}
```

Überblick

- Beispielprogramm
- Pseudocode
- Java
  - Programmierumgebung
  - Datentypen
  - Operatoren, Kontrollstrukturen
- weiterführende Quellen

Operatoren

- `+, -, *, /, %` (Rest)
- `=, +=, -=` (Zuweisung)
- `==, !=` (Gleichheit, Ungleichheit)
- `&&, ||, !` (logisches Und, Oder, Negation)
- `++, --` (Inkrement, Dekrement)
- und viele mehr

```
int a1 = 1, a2 = 1, b1, b2;
b1 = a1++; // b1 = 1, a1 = 2
b2 = ++a2; // b2 = 2, a2 = 2
```

if-then-else

```
void applyBrakes()
{
   if (isMoving==true)
   {
      currentSpeed--;
   }
   else
   {
      System.err.println("The bicycle has
      already stopped!");
   }
}
```

switch

```
switch (month)
{
   case  1: System.out.println("Januar");
            break;
   case  2: System.out.println("Februar");
            break;
   case  3:
   case  4: System.out.println("März oder April");
            break;
   …
   case 12: System.out.println("Dezember");
            break;
   default: System.out.println("ungültiger Monat");
            break;
}
```

while, do-while

- ```
  int count = 1;
  while (count < 11)
  {
     System.out.println("Zähler ist: " + count);
     count++;
  }
  ```

- ```
  int count = 1;
  do
  {
     System.out.println("Zähler ist: " + count);
     count++;
  } while (count <= 11);
  ```

for

Initialisierung | Abbruch-bedingung | Inkrement

- ```
  for (int i=1; i<11; i++)
  {
     System.out.println("Count is: " + i);
  }
  ```
  ```

Überblick

- Beispielprogramm
- Pseudocode
- Java
  - Programmierumgebung
  - Datentypen
  - Operatoren, Kontrollstrukturen
- weiterführende Quellen

Vertiefende Veranstaltungen

- Programmierung in C++ (Hannah Bast)
- Programmieren in Java (Peter Thiemann)
- Sommercampus
  - von Studenten und wissenschaftlichen Mitarbeitern organisiert
  - Java, C, C++, C#, ..., latex, vi
- BOK-Kurse zu Java, C, C++
  - Berufsfeldorientierte Kompetenzen am Zentrum für Schlüsselqualifikation
  - http://www.zfs.uni-freiburg.de/
  - teilweise anrechenbar
  - innerhalb und außerhalb der Vorlesungszeit

Tutorials

- Java
  - http://java.sun.com/docs/books/tutorial/
  - http://math.hws.edu/javanotes/
- C / C++
  - http://www.cplusplus.com
  - http://cplus.about.com
  - http://www.cprogramming.com
  - http://www.tutorialpage.de

Nochmal

- Betriebssysteme, Programmiersprachen und Entwicklungsumgebungen sind wichtige Werkzeuge in vielen Informatik-Veranstaltungen.
- Wissen in diesen Bereich möglichst vielseitig und selbständig erwerben
- nicht zuviel Spezialwissen in einer Programmiersprache / Entwicklungsumgebung
- Alle Betriebssysteme, Programmiersprachen, Entwicklungsumgebungen und Computer- bzw. Mac-Hersteller sind gleich.

Nochmal

- unterschiedliche, sich teilweise überlappende Aspekte
  - Algorithmen und Datenstrukturen
  - Softwaretechnik
  - Programmiersprachen

Zusammenfassung

- Beispielprogramm
- Pseudocode
- Java
  - Programmierumgebung
  - Datentypen
  - Operatoren, Kontrollstrukturen
- weiterführende Quellen

Nächstes Thema

- Analyse von Algorithmen
  - Korrektheit und Effizienz
- Korrektheit
  - Ein korrekter Algorithmus stoppt (terminiert) für jede Eingabeinstanz mit der durch die Eingabe-Ausgabe-Relation definierten Ausgabe.
- Beweisregeln