Hochschule für
Wirtschaft und Recht Berlin
Berlin School of Economics and Law

Grundlagen der Programmierung

Einführung in die strukturierte Programmierung – Teil 5

Die Inhalte der Vorlesung wurden primär auf Basis der angegebenen Literatur erstellt. Darüber hinaus orientiert sich die Vorlesung an der von Prof. Dr.-Ing. Faustmann (ebenfalls HWR Berlin).

Verwendung von Feldern (Arrays)

Arrays

Ein Array stellt eine geordnete Gruppe von Variablen des selben Typs dar. Zum Erstellen eines Arrays werden 3 Schritte benötigt:

- Deklaration eines Arrays (bekannt gemacht)
- Erstellen (allokieren – Speicherplatz bzw. Ressourcen) eines Arrays
- Initialisierung bzw. programmtechnische Verwendung des Arrays

Beispiel (Deklaration und Allokierung eines Arrays mit 4 Elementen):

Deklaration → `int[] lottoZahlen;`

Erstellen → `lottoZahlen = new int [4];`

oder → `int[] lottoZahlen = new int [4];`

Arrays

Der Zugriff auf die Elemente eines Arrays erfolgt nicht über ihren Namen, sondern über einen Index.

- Indizierung durch nichtnegative Integervariablen (ganzzahliger Index)
- Feld mit n Elementen → Indizes von 0 bis n – 1 (inklusive)!!!
- maximale Länge eines Feldes ist auf 2^{31} (4 Byte) begrenzt

Beispiel für den Zugriff:

Element 1 → `lottoZahlen[0] = 22;`

Element 2 → `lottoZahlen[1] = 2;`

Element 3 → `lottoZahlen[2] = 14;`

oder : → `int[] lottoZahlen = {22,2,45,34,37};`

Arrays

Beispiel eines Arrays mit 5 Elementen:

```
// Deklaration und Allokation
int [] lottoZahlen = new int [5];
// Initialisierung der Elemente
lottoZahlen[0] = 22;
lottoZahlen[1] = 14;
lottoZahlen[2] = 34;
lottoZahlen[3] = 45;
lottoZahlen[4] = 30;
// Ausgabe der Inhalte
for (int x = 0; x <lottoZahlen.length;x++){
          System.out.print(lottoZahlen[x] + "\t");
}
```

Arrays

```
public static void getArrayTest(){
    System.out.println("Test von Arrays");
    int[] a = new int[4];
    int[] smallPrimes = a;
    a[0] = 10;
    a[1] = 20;
    a[2] = 30;
    a[3] = 40;
    for (int i=0; i<smallPrimes.length;i++){
        System.out.println(smallPrimes[i]);
    }
```

Kopieren von Array-Variablen:

- Dann verweisen allerdings beide auf dasselbe Array:
- z.B.

```
int[] smallPrimes = a;
a[3] = 40; // smallPrimes[3] ist jetzt auch 40
```

Elemente zwischen Arrays kopieren:

- Verwendung der Methode

```
System.arraycopy(from, fromIndex, to, toIndex, count)
```

Das Array „to“ muss dazu genügend Platz für die zu kopierenden Elemente bieten.

Arrays

Beispiel - Elemente zwischen Arrays kopieren:

```
int[] a = { 11, 22, 33, 44, 55, 66 };

int[] b = { 1, 11, 111, 1111, 11111, 111111, 1111111 };

System.arraycopy(a, 2, b, 3, 4);
```

Inhalte der Felder nach dem Kopieren:

- Feld a: `11, 22, 33, 44, 55, 66`
- Feld b: `1, 11, 111, 33, 44, 55, 66`

```
public static void getArrayCopyTest(){

    int[] a = { 11, 22, 33, 44, 55, 66 };
    int[] b = { 1, 11, 111, 1111, 11111, 111111, 1111111 };

    System.arraycopy(a, 2, b, 3, 4);

    for (int i=0; i<a.length;i++){
        System.out.println(a[i]);
    }

    for (int i=0; i<=6;i++){
        System.out.println(b[i]);
    }
}
```

Arrays – Übung 1

Erstellen Sie ein Programm zur Verwaltung der Kalendertage eines Jahres. Verwenden Sie entsprechende Felder zur Verwaltung der Monate (Januar, Februar, März, April, …) und zur Berücksichtigung der je Monat zur Verfügung stehenden Tage. Bieten Sie dem Nutzer die Möglichkeit sich die Tage (Format: 02. März 2010) eines Monats anzeigen zu lassen. Berücksichtigen ggf. auftretende Schaltjahre!

```
1. Juni 2010
2. Juni 2010
3. Juni 2010
4. Juni 2010
5. Juni 2010
6. Juni 2010
7. Juni 2010
8. Juni 2010
9. Juni 2010
10. Juni 2010
11. Juni 2010
12. Juni 2010
13. Juni 2010
14. Juni 2010
15. Juni 2010
16. Juni 2010
17. Juni 2010
18. Juni 2010
19. Juni 2010
20. Juni 2010
21. Juni 2010
22. Juni 2010
23. Juni 2010
24. Juni 2010
25. Juni 2010
26. Juni 2010
27. Juni 2010
28. Juni 2010
29. Juni 2010
30. Juni 2010
```

Hinweis:

```
int tage [] {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}

String monate [] = {„Januar“, „Februar“, „März“, „April“, …}
```

Arrays – Übung 2

Erstellen Sie ein Programm, das Lottozahlen (z.B. 6 aus 49) zieht. Fragen Sie dazu die Anzahl der verfügbaren Zahlen und die Anzahl der zu ziehenden Zahlen ab. Verwalten Sie zwei Felder `int[] numbers` und `int[] result`.

Hinweis: Die Anweisung `(int)(Math.random() * n);` ermittelt eine zufällige Ganzzahl zwischen 0 und n-1.

Optional: Geben Sie dem Nutzer die Möglichkeit einen Tip abzugeben, der mit den gezogenen Zahlen verglichen wird.

Verwendung des Datentyps String

Der Datentyp String

Die Java-Klasse String wird so häufig in der Programmierung verwendet, dass man sie fast für einen einfachen Datentyp halten könnte. Die Tatsache, dass „String“ mit einem Grossbuchstaben beginnt, weist allerdings darauf hin, dass es sich um eine Java-Klasse handelt.

Quelle: Sanchez, J.; Canton, M. P.: Java 2 Wochenend Crashkurs, mitp-Verlag, Landberg

Der Datentyp String

- Zeichenketten werden in Java durch die Klasse String repräsentiert. Folgende Möglichkeiten gibt es, einen String zu erzeugen:
  - Zuweisung eines Literals: `String s = "Hallo";`
  - Erzeugen eines leeren Stringobjekts: `String t = new String();`
  - Duplizieren eines vorhandenen Stringobjekts: `String u = new String(s);`
- Die Anweisung `String v = s;` erzeugt keinen neuen String, sondern nur eine weitere Referenz (allgemein Adresse) auf den bereits vorhandenen String s!

Der Datentyp String

Methoden zum Auslesen von Strings:

- `char charAt(int index)`

  Liefert das Zeichen an Position `index` (0..length-1). Es wird `StringIndexOutOfBoundsException` geworfen, falls index nicht in diesem Bereich liegt!

- `String substring(int begin, int end)`

  Liefert den Teilstring, der an Position `begin` beginnt und an Position `end` endet. `end` zeigt auf das erste Zeichen hinter dem Teilstring!

- `String trim()`

  Liefert den String, der entsteht, wenn auf beiden Seiten der Zeichenkette alle zusammenhängenden Leerzeichen entfernt werden.

- `int length()`

  Liefert die aktuelle Länge des Stringobjekts.

Strings – Übung 3

- Welche Ausgaben werden exakt auf der Console erzeugt?
  - Eigene Analyse ohne Test!
  - Test mit Hilfe der kompletten Implementierung!
  - Experimentieren Sie mit den Möglichkeiten der eingesetzten Methoden der Klasse String!
- Einzelübung
- Dauer: ca. 10 min.

```
public static void main(String[] args) {
    // TODO Auto-generated method stub
    System.out.println(getString());
}

public static String getString(){
    String [] myStrAr = new String[5];
    myStrAr[0] = "Otto";
    myStrAr[1] = "Gaby";
    myStrAr[2] = "Frederik";
    myStrAr[3] = "Karl";
    myStrAr[4] = "Sveta";
    char c = myStrAr[1].charAt(0);
    System.out.println(c);
    String bsp = myStrAr[0].substring(0,3);
    for (int i=0; i<myStrAr.length; i++){
        System.out.print(" "+myStrAr[i]);
    }
    return bsp;
}
```

Strings – Übung 4

- Welche Ausgaben werden exakt auf der Console erzeugt?
  - Eigene Analyse ohne Test!
  - Test mit Hilfe der kompletten Implementierung!
  - Experimentieren Sie mit den Möglichkeiten der eingesetzten Methoden!
- Warum funktioniert die erste Möglichkeit zum Prüfen der inhaltlichen Gleichheit nicht?
- Gruppenübung
- Dauer: ca. 15 min.

```
public static void main(String[] args) {
    // TODO Auto-generated method stub
    System.out.println(getString());
    char data [] = {'a', 'b', 'c', 'd'};
    getString2(data);
}

public static String getString(){

public static void getString2(char data[]){
    String s = new String (data);
    //Problem der Prüfung auf inhaltliche Gleichheit!
    if (s=="abcd"){
        System.out.println("Gleichheit");
    }else{
        System.out.println("Ungleichheit");
    }
    System.out.println(s);
    String t = "xyz";
    System.out.println(s+t);
    System.out.println(s+t.toUpperCase());
    String a = s.substring(1,3)+t;
    System.out.println(a);
    if (a.equals("bcxyz")){
        System.out.println("Gleichheit");
    }
}
```

Strings – Übung 5

Übung:

Schreiben Sie ein Java-Programm, das ein Wort von der Tastatur einliest und das Wort in Dreiecksform auf der Konsole ausgibt.

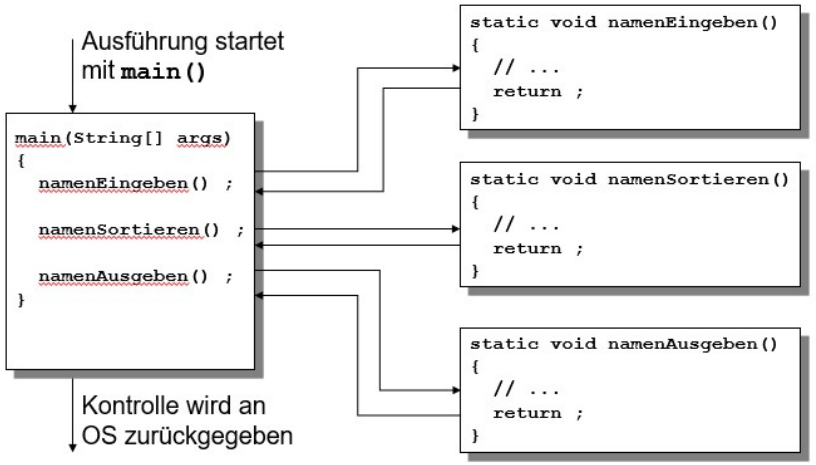
Beispiele:

Eingabe:	Berufsakademie	Hallo	Computer
Ausgabe:	ak	l	pu
	saka	all	mput
	fsakad	Hallo	ompute
	ufsakade		Computer
	rufsakadem		
	erufsakademi		
	Berufsakademie		

Reflexion der Verwendung von Methoden

Methoden eines Java-Programms

- Aufgaben werden durch definierte Funktionen (Methoden) erledigt.
- In Java existieren Funktionen nur als Methoden von Klassen, globale Funktionen sind nicht vorhanden.
- Vorwärtsverweise auf weiter unten im Programmtext stehende Methoden sind zulässig.



Methoden eines Java-Programms

Deklaration und Definition einer Methode (Bestandteile):

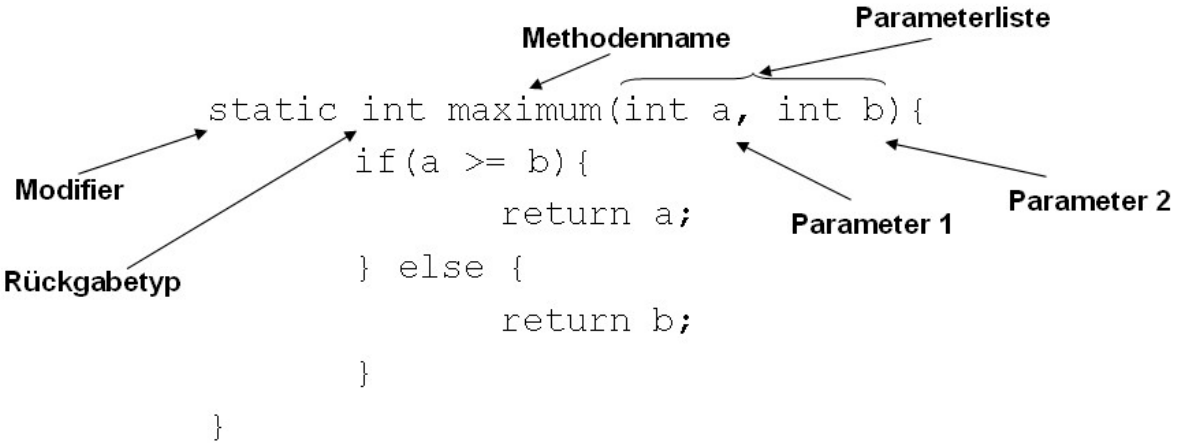
- Optionaler Modifier – Kennzeichnung von Methodeneigenschaften
- Rückgabetyp der Methode festlegen
- Name der Methode (Funktion)
- Parameterliste (auch als Argumente bezeichnet)
- Liste möglicher Exceptions (geworfene Ausnahmen)
- Rumpf der Methode (unmittelbar nach der Deklaration)

Methoden eines Java-Programms

Allgemeiner Aufbau einer Methode:

```
[Modifier]
Typ Methodenname (Parameterliste)
    [throws Exception-Liste]
{
    Rumpf der Methode
}
```

Konkreter Aufbau einer Methode:



Methoden eines Java-Programms

Modifier-Attribut `static`:

- Methode wird als Klassenmethode gekennzeichnet
- Aufruf kann erfolgen ohne Objekte der Klasse zu erzeugen
- Zugriffe auf Variablen aus einer **`static`**-Methode:
  - Können nur auf eigene lokale Variablen innerhalb der Methode erfolgen
  - Zugriffe können ebenfalls auf Klassenvariablen erfolgen
  - Auf Instanzvariablen eines Objektes kann nicht zugegriffen werden
- Weitere Modifier-Attribute: **`public, protected, abstract,`** …

Methoden eines Java-Programms

Aufruf einer Methode:

- Der Aufruf einer Methode erfolgt mit dem Punktoperator:

```
objekt.methode();
```

- Statische Methoden können direkt aus der main-Methode der Klasse aufgerufen werden:

```
static public void main(String[] args){
    int zahl1 = 10;
    int zahl2 = 20;
    System.out.println("Maximum ist: " +
                                maximum(zahl1, zahl2);
}
```

- Die Namen der Aufrufvariablen und der Methodenparameter müssen nicht übereinstimmen. Es findet ein Parametermatching statt (im Beispiel: a=zahl1, b=zahl2).

Java-Methoden – Übung 6

Schreiben Sie eine Methode die den Wert einer übergebenen Integervariablen für den Bereich 500 bis -500 überprüft. Liegt der übergebene Wert in diesem Bereich soll er zurückgegeben werden, andernfalls auf 500 (wenn der Wert > 500 ist) bzw. -500 (wenn der Wert < -500 ist) begrenzt werden. Verwenden Sie innerhalb der Methode mehrere return-Anweisungen und testen Sie die Methode innerhalb einer entsprechenden Java-Applikation.

Java-Methoden – Übung 7

- Testen Sie das nebenstehende Quellcodefragment.
- Löschen Sie das Modifier-Attribut `static` und beobachten Sie die Änderungen im Programm (Fehler).
- Beseitigen Sie bitte den aufgetretenen Fehler durch eine adäquate Anpassung des Quellcodes.

```
 * @param args
public static void main(String[] args) {
    // TODO Auto-generated method stub
    System.out.println(TestMeth.getMaximum(9, 5));

    TestMeth myTMObj =new TestMeth();
    System.out.println(myTMObj.getMaximum(44, 99));
}

public static int getMaximum(int a, int b) {
    if (a >= b) {
        return a;
    } else {
        return b;
    }
}
```

Methoden eines Java-Programms

Übergebene Parameter (primitive Datentypen) :

- Jede Methode kann eine Parameterliste besitzen.
- Jede Parameterangabe besteht aus einem Typnamen und dem Parameternamen.
- Für alle Parameter der Methode erfolgt während des Aufrufs die Bereitstellung von Speicherplatz und die Initialisierung
- Die Parameterübergabe geschieht in Java als Kopie (*call by value*).
- Innerhalb einer Methode können Parameter verändert werden, dies bleibt für den Aufrufer unsichtbar.
- Funktionen mit Seiteneffekten (wie z.B. bei C möglich) sind in Java nicht möglich.

Methoden eines Java-Programms

Übergebene Parameter (primitive Datentypen) :

```java
public class FacultyTest {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        int value = 12;
        // Call by Value (Wertübergabe)
        long erg = Faculty.getFaculty(value);
        System.out.println(erg);
    }

}
```

```java
public class Faculty {


    public static long getFaculty(long n) {
        long erg = 1L;
        while (n > 1) {
            erg = erg * n;
            n--;
        }
        return erg;
    }

}
```

- Aufruf:
```
int value = 12;
System.out.print(getFaculty(value));
```
- Beim Aufrufer beträgt der Wert der Integervariable value auch nach dem Aufruf den Wert 12!

Methoden eines Java-Programms

Übergebene Parameter (komplexe Datentypen) :

- Variablen, die Arrays, Strings, Klassen oder Schnittstellen speichern, werden als Referenztypen bezeichnet.
- Derartige Referenztypen speichern nicht den Inhalt der Variablen, sondern die Adresse des Objekt im Speicher.
- Für Parameter eines Referenztyps wird die Kopie der Referenz (allg. Zeiger) auf das entsprechende Objekt übergeben.
- Wird eine solche Variable an eine Methode übergeben, kann die Methode das Objekt verändern (ähnlich: *call by reference*)!

Methoden eines Java-Programms

Übergebene Parameter (komplexe Datentypen) :

```
public static void main(String[] args){
        int[] feld = new int[100];
        initialisiere(feld); // alle Element von feld auf 0 gesetzt
}

static void initialisiere(int[] f){
        for(int i = 0; i < f.length; i++){
                f[i] = 0;
                System.out.println("Prüfausgabe"+f[i]);
        }}
```

- Obwohl hier nur der Wert der Array-Variablen `feld` als Kopie übergeben wird, können in der Methode `initialisiere` Änderungen an den Feldelementen durchgeführt werden.
- Da `feld` eine Referenz (Adresse) darstellt, werden diese Änderungen am Original durchgeführt.

Methoden eines Java-Programms

```
public static void main(String[] args) {
    // TODO Auto-generated method stub
    StringBuilder mySB = new StringBuilder("HWR");
    // Methode haengt Zeichen an
    getRefTest1(mySB);
    System.out.println(mySB);
    getRefTest2(mySB);
    System.out.println(mySB);
    mySB = getRefTest3(mySB);
    System.out.println(mySB);
}

public static void getRefTest1(StringBuilder a){
    a.append(" Berlin");
}

public static void getRefTest2(StringBuilder a){
    a = new StringBuilder("Fachbereich");
}

public static StringBuilder getRefTest3(StringBuilder a){
    return new StringBuilder("Fachbereich");
}
```

Methoden eines Java-Programms

Rückgabewerte von Methoden:

- Bei der Deklaration einer Methode muß der Rückgabewert, d.h. der Typ der Methode vereinbart werden.
- Mögliche Rückgabewerte:
  - Elementare Datentypen
  - Beliebige Referenztypen
  - Kein Rückgabewert → void
- Der Rückgabewert einer Methode wird mit der return-Anweisung spezifiziert, bei deren Ausführung wird die Methode beendet.
- Genau wie bei der Parameterübergabe werden Ergebniswerte als Kopie zurückgeliefert.

Feldvergrößerung – Übung 8

Realisieren Sie eine Methode

```
int[] vergroessern(int[] feld, int n),
```

die ein Integer-Array `feld` als Parameter erhält und das Array um `n` Speicherplätze vergrößert zurückgibt.

Die Methode soll z.B. auf folgende Weise benutzt werden können:

```
int[] test = {5, 1, 8, 6}; // Index 0..3
test = vergroessern(test, 1); // Feld um 1 vergroeessern
test[4] = 2;          // Zugriff auf Index 4 jetzt moeglich
```

Feldvergrößerung

```
public class TestClassA {

    public static void main(String[] args) {
        int[] test = {5, 1, 8, 6};
        // bisherige Indexe 0..3
        test = vergroessern(test, 2);
        // Feld wurde um 2 Elemente vergroeessert
        test[4] = 2;
        test[5] = 2;
        // Zugriff auf Indexe 4 und 5 jetzt moeglich
        for (int i=0;i<test.length;i++){
            System.out.println(test[i]);
        }
    }

    public static int[] vergroessern(int[] feld, int n){
        int a = feld.length;
        int test[] = new int[a+n];
        System.arraycopy(feld, 0, test, 0, feld.length);
        return test;
    }
}
```