

Vorlesung



Programmieren I und II

Unit 3

Selbstdefinierbare Datentypen, Arrays und Collections

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

1

1

Disclaimer



Zur rechtlichen Lage an Hochschulen:

Diese Handout und seine Inhalte sind durch den Autor selbst erstellt. Aus Gründen der Praktikabilität für Studierende lehnen sich die Inhalte stellenweise im Rahmen des Zitatrechts an Lehrwerken an.

Diese Lehrwerke sind explizit angegeben.

Abbildungen sind selber erstellt, als Zitate kenntlich gemacht oder unterliegen einer Lizenz die nicht die explizite Nennung vorsieht. Sollten Abbildungen in Einzelfällen aus Gründen der Praktikabilität nicht explizit als Zitate kenntlichgemacht sein, so ergibt sich die Herkunft immer aus ihrem Kontext: „Zum Nachlesen ...“.

Creative Commons:

Und damit andere mit diesen Inhalten vernünftig arbeiten können, wird dieses Handout unter einer Creative Commons Attribution-ShareAlike Lizenz (CC BY-SA 4.0) bereitgestellt.

<https://creativecommons.org/licenses/by-sa/4.0>



Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

2

2



**Prof. Dr. rer. nat.
Nane Kratzke**

*Praktische Informatik und
betriebliche Informationssysteme*

- Raum: 17-0.10
- Tel.: 0451 300 5549
- Email: nane.kratzke@th-luebeck.de



@NaneKratzke
Updates der Handouts auch über Twitter [#prog_inf](#) und [#prog_itd](#)



Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

3

3

Units



1. Semester	Unit 1 Einleitung und Grundbegriffe	Unit 2 Grundlagen imperativer Programmierung	Unit 3 Selbstdefinierbare Datentypen und Collections	Unit 4 Einfache I/O Programmierung
	Unit 5 Rekursive Programmierung, rekursive Datenstrukturen, Lambdas	Unit 6 Objektorientierte Programmierung und UML	Unit 7 Konzepte objektorientierter Programmiersprachen, Klassen vs. Objekte, Pakete und Exceptions	Unit 8 Testen (objektorientierter) Programme
2. Semester	Unit 9 Generische Datentypen	Unit 10 Objektorientierter Entwurf und objektorientierte Designprinzipien	Unit 11 Graphical User Interfaces	Unit 12 Multithread Programmierung

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

4

4

Abgedeckte Ziele dieser UNIT



Kennen existierender Programmierparadigmen und Laufzeitmodelle	Sicheres Anwenden grundlegender programmiersprachlicher Konzepte (Datentypen, Variable, Operatoren, Ausdrücke, Kontrollstrukturen)	Fähigkeit zur problemorientierten Definition und Nutzung von Routinen und Referenztypen (insbesondere Liste, Stack, Mapping)	Verstehen des Unterschieds zwischen Werte- und Referenzsemantik
Kennen und Anwenden des Prinzips der rekursiven Programmierung und rekursiver Datenstrukturen	Kennen des Algorithmusbegriffs, Implementieren einfacher Algorithmen	Kennen objektorientierter Konzepte Datenkapselung, Polymorphie und Vererbung	Sicheres Anwenden programmiersprachlicher Konzepte der Objektorientierung (Klassen und Objekte, Schnittstellen und Generics, Streams, GUI und MVC)
Kennen von UML Klassendiagrammen, sicheres Übersetzen von UML Klassendiagrammen in Java (und von Java in UML)	Kennen der Grenzen des Testens von Software und erste Erfahrungen im Testen (objektorientierter) Software	Sammeln erster Erfahrungen in der Anwendung objektorientierter Entwurfsprinzipien	Sammeln von Erfahrungen mit weiteren Programmiermodellen und -paradigmen, insbesondere Multithread Programmierung sowie funktionale Programmierung

Am Beispiel der Sprache JAVA

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

5

5

Themen dieser Unit





Referenzdatentypen

- Felder (Arrays)
- Klassen

Collections

- Listen
- Mappings

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

6

6

Zum Nachlesen ...



Kapitel 5

Referenzdatentypen

Abschnitt 5.1 Felder (Arrays)

Abschnitt 5.2 Klassen

Kapitel 8

Der grundlegende Umgang mit Klassen

Abschnitt 8.3 Statische Komponenten

Abschnitt 8.4 Instantiierung

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

7

7

Felder (Arrays)



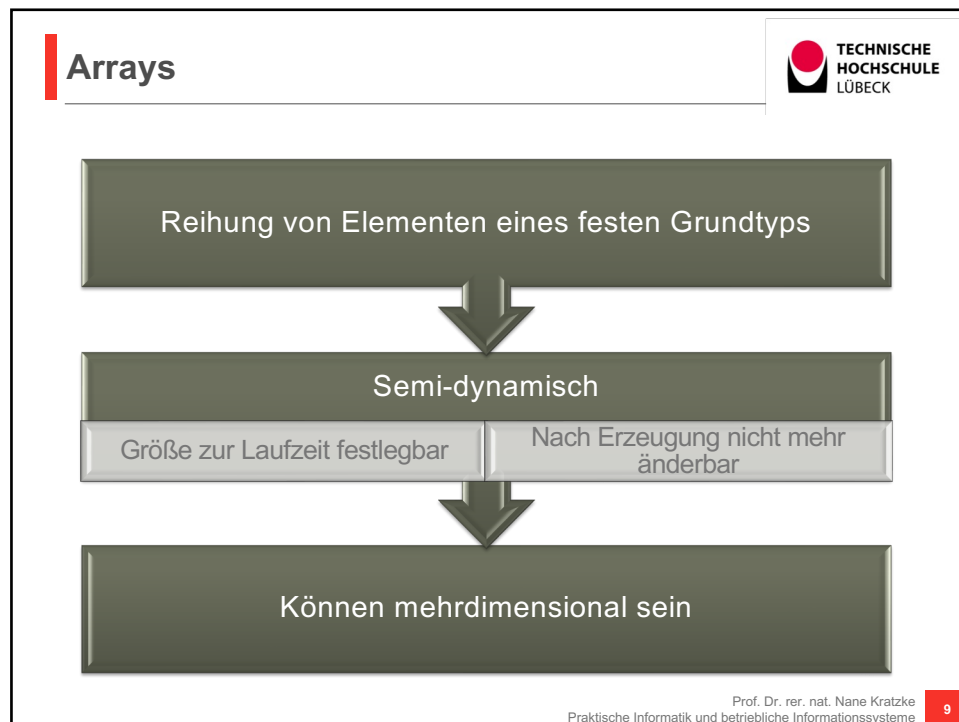
- Sie haben bislang nur primitive Datentypen kennengelernt. Korrespondierende Variablen können pro Variable genau einen Wert speichern.
- **Felder (Arrays)** gestatten es, **mehrere Variablen** über einen gemeinsamen **Namen** anzusprechen
- und lediglich durch einen Index zu unterscheiden.
- Der **Index** innerhalb eines Feldes (Arrays) ergibt sich dabei aus der Position innerhalb des Feldes (Arrays), **von null aufwärts gezählt**.

termin	
termin[0]	
	0
	1
	2
	3
	4
	5
termin[6]	Der Wecker klingelt
	6
	7
	8
	9
	10
	Vorlesung schlafn
	11
	Mittagessen mit Kalle
	12
	13
termin[14]	Aufgaben bearbeiten
	14
	15
	16
	Sport
	17
	18
	19
	20
	21
	22
termin[23]	Dösen
	23

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

8

8



9

Deklaration und Erzeugung von Arrays

Variante 1: Deklarieren und Erzeugen eines Arrays in zwei Schritten:

```
Typ[] var;           // Deklaration eines Arrays  
  
var = new Typ[n]; // Erzeugung eines Array mit n Elementen
```

Variante 2: Deklarieren und Erzeugen eines Arrays in einem Schritt.
Die Zuweisung muss dabei unmittelbar bei der Deklaration erfolgen.

```
Typ[] var = { new Typ(), ..., new Typ() };
```

Beide Varianten erzeugen eine Reihung von n Elementen des Typs **Typ**. Nach der Erzeugung kann n nicht mehr verändert werden, d.h. die Größe des Arrays ist nach Erzeugung unveränderlich.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

10

10

Deklaration und Erzeugung von Arrays

Beispiele

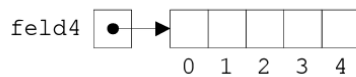


Festlegen des Typs:

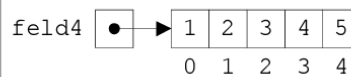
```
int[] feld4;
```

Anlegen der Größe:

```
feld4 = new int[5];
```



```
feld4 = new int[5]
```



```
feld4[0]=1;
feld4[1]=2;
feld4[2]=3;
feld4[3]=4;
feld4[4]=5;
```

Anlegen eines leeren Arrays

Befüllen eines Arrays mit Werten

Prof. Dr. rer. nat. Nane Kratzke
 Praktische Informatik und betriebliche Informationssysteme

11

11

Deklaration und Erzeugung von Arrays

Beispiele

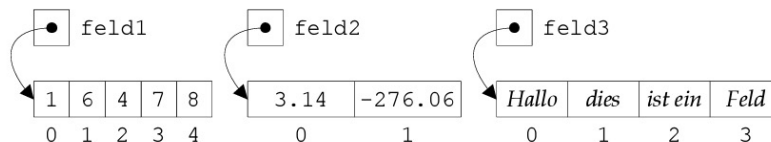


Festlegen des Typs:

```
int[] feld1;
double[] feld2;
String[] feld3;
```

Anlegen der Größe:

```
feld1 = new int[5];
feld2 = new double[2];
feld3 = new String[4];
```



Felddeklaration, Anlegen der Größe und initiale Befüllung kann auch mittels **Feldinitialisierer (array initializer)** in einem Schritt erfolgen.

```
int[] feld1={1 , 6 , 4 , 7 , 8};
double[] feld2={3.14 , -276.06};
String[] feld3={"Hallo" , "dies" , "ist ein", "Feld"};
```

Prof. Dr. rer. nat. Nane Kratzke
 Praktische Informatik und betriebliche Informationssysteme

12

12

Zugriff auf Array-Elemente



Schreibender Zugriff:

```
int[] prim = new int[5];           // Array mit 5 int Elementen

prim[0] = 2;                       // Setzen von Array-Elementen
prim[1] = 3;
prim[2] = 5;
prim[3] = 7;
prim[4] = 11;
```

prim:

2	3	5	7	11
---	---	---	---	----

// Kürzere foreach Variante
 for (int i : prim) {
 System.out.println(i);
 }

Typischer lesender Zugriff über einen Laufindex:

```
int length = prim.length           // Länge des Arrays
for (int i = 0; i < length; i++) {
    System.out.println(prim[i]);    // Zugriff auf Elem.
}
```

Prof. Dr. rer. nat. Nane Kratzke
 Praktische Informatik und betriebliche Informationssysteme

13

13

Referenzen



Arrays werden anders gespeichert, als die Ihnen bislang bekannten primitiven Datentypen. Eine Arrayvariable beinhaltet einen Verweis auf die Inhalte des Arrays, nicht die Inhalte selber! Der Unterschied fällt vor allem beim zuweisen von Werten auf.

x1

32

 x2

a

 x3

3.14

 (1)

int x1 = 32; char x2 = 'a'; double x3 = 3.14;

y1

32

 y2

a

 y3

3.14

 (2)

int y1 = x1; char y2 = x2; double y3 = x3;

Bei primitiven
 Datentypen werden
 die Werte kopiert.

feld

•

 →

1	6	4	7	8
---	---	---	---	---

 (1)

int[] kop = feld; kop

•

 →

1	6	4	7	8
---	---	---	---	---

 (2)

Bei Arrays die
 Referenz auf ein
 Array (nicht die
 Inhalte/Werte).

Prof. Dr. rer. nat. Nane Kratzke
 Praktische Informatik und betriebliche Informationssysteme

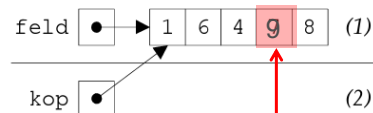
14

14

Referenzen



Was passiert, wenn Sie eine Referenzkopie verändern?



```
int[] kop = feld;
```

```
kop[3] = 9;
```

Sie ändern sowohl die „Kopie“ als auch das „Original“ !!!

(Eigentlich gibt es keine Kopie und Original nur zwei Referenzen auf denselben Hauptspeicherbereich)

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

15

15

Mehrdimensionale Arrays (I)



Beispiel für ein einfaches mehrdimensionales Array.



```
Schachfigur[][] schachbrett = new Schachfigur[8][8];
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

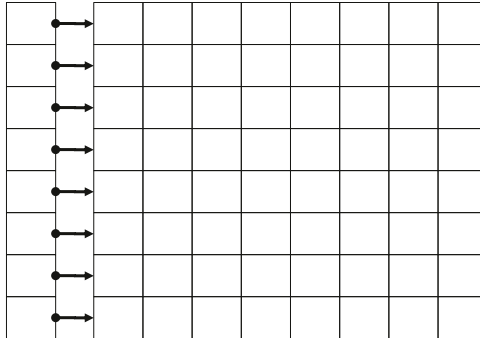
16

16

Mehrdimensionale Arrays (II)



Mehrdimensionale Arrays werden als Arrays von Arrays angelegt.



```
Schachfigur[][] schachbrett = new Schachfigur[8][8];
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

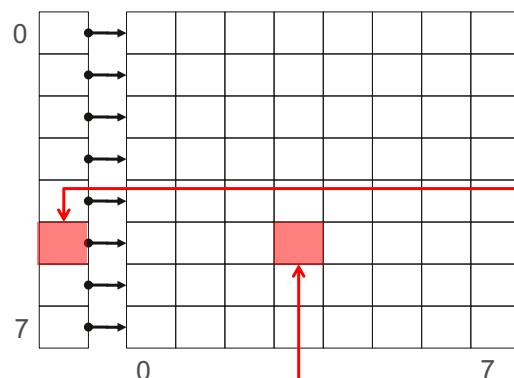
17

17

Mehrdimensionale Arrays (III)



Mehrdimensionale Arrays werden als Arrays von Arrays angelegt.



```
schachbrett[i][j]
```

```
Schachbrett[5][j]
```

```
Schachbrett[5][3]
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

18

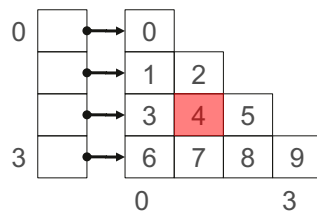
18

Mehrdimensionale Arrays (IV)



Es ist auch möglich nicht rechteckige Arrays anzulegen.

```
int[][] a = { { 0 },  
              { 1, 2 },  
              { 3, 4, 5 },  
              { 6, 7, 8, 9 } };
```



Welchen Wert hat
dieser Ausdruck?

`a[2][1] = ???`

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

19

19

Referenztypen



Referenztypen

- Alle selbst definierten Datentypen
- oder Arrays
- Werden mittels new Operator erzeugt

Variablen von Referenztypen

- enthalten Referenz auf erzeugte Objekte
- nicht die Inhalte der Objekte

Speicher- management

- nicht referenzierte Objekte
- werden durch einen Garbage Collector automatisch freigegeben



Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

20

20

Klassen



- Unter einer Klasse versteht man einen selbstdefinierten Datentyp, der üblicherweise mehrere Komponenten umfasst, die mittels primitiver Datentypen ausgedrückt werden können.
- Eine Klasse kann aber auch prinzipiell Komponenten beinhalten, die wiederum Klassen sind.
- Am einfachsten macht man sich eine Klasse am Beispiel einer Adresse deutlich.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

21

21

Definition eigener Datentypen



Eine Adresse ist sicher ein sinnvoller Datentyp für eine Vielzahl von Anwendungen, existiert jedoch nicht in JAVA.

Eine Adresse kann jedoch aus mehreren „atomaren“ Komponenten zusammengesetzt werden.

Adresse	
name:	String
strasse:	String
hausnummer:	int
postleitzahl:	int
wohnort:	String
mail:	String
kommentar:	String

```
class Adresse {  
    public String name;  
    public String strasse;  
    public int hausnummer;  
    public int postleitzahl;  
    public String wohnort;  
    public String mail;  
    public String kommentar;  
}
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

22

22

Erzeugen von Objekten eigener Datentypen



Ein Objekt eines Referenzdatentyps wird ähnlich erzeugt wie ein Array.
Mit Hilfe des `new` Operators.

```
Adresse adr = new Adresse();
```

adr

Diese Zeile erzeugt ein neues Objekt vom Datentyp `Adresse` und speichert die Referenz auf dieses Objekt in der Variablen `adr`.

Auf die einzelnen Komponenten eines Datentyps kann dann auf folgende Art zugegriffen werden.

```
adr.strasse = "Mönkhofer Weg";
```

Schreibender Zugriff

```
System.out.println(adr.strasse);
```

Lesender Zugriff

`adr.strasse` ist zu lesen wie: Greife auf die Komponente `strasse` des Objekts zu, dessen Referenz in `adr` gespeichert ist.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

23

23

Konstruktor



Um ein Adressobjekt anzulegen, kann man also wie folgt vorgehen.

```
Adresse adr = new Adresse();  
adr.name = "Max Mustermann";  
adr.strasse = "Mönkhofer Weg";  
adr.hausnummer = 239;  
...
```

Um sich diese Einzelinitialisierungen der Komponenten zu ersparen, wird üblicherweise ein Konstruktor definiert, der im Rahmen des Anlegens eines Objekts wie folgt aufgerufen werden kann.

```
Adresse adr = new Adresse(  
    "Max Mustermann",  
    "Mönkhofer Weg",  
    239, ...);
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

24

24

Konstruktor (II)



Ein Konstruktor belegt dabei die Komponenten eines Datentyps mit Werten. Ein Konstruktor ist eigentlich nichts weiter als eine spezielle Methode die im Rahmen der Initialisierung eines Objekts aufgerufen wird.

```
class Adresse {
    public String name;
    public String strasse;
    public int hausnummer;

    ...
    // Konstruktor
    public Adresse(String n, String s, int h) {
        this.name = n;
        this.strasse = s;
        this.hausnummer = h;
        ...
    }
}
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

25

25

toString() (I)



Werte von Variablen primitiver Datentypen kann man einfach mit der `System.out.println` Methode ausgeben. Versucht man dasselbe mit Referenztypen (z.B. Adressen) erhält man Ausgaben der folgenden Art:

```
Adresse adr = new Adresse("Max Mustermann", "Mönkhofer
Weg", 239, 23562, "Lübeck");

System.out.println(adr);
```

Adresse@33f42b49

↑ ↑

Typ des Objekts Adresse des Objekts

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

26

26

toString() (II)



Dies ist im allgemeinen für den menschlichen Leser nicht geeignet. Daher gibt es in JAVA die Konvention anstelle des Typs und Adresse eines Objekts die Inhalte des Objekts auszugeben.

Die Art und Weise der Darstellung kann selber mittels einer `toString` Methode festgelegt werden.

```
class Adresse {  
    public String name;  
    public String strasse;  
    public int hausnummer  
    ...  
  
    public String toString() {  
        return this.name + "\n" + this.strasse + " " +  
            this.hausnummer + "\n"  
            this.postleitzahl + " " + this.wohnort + "\n";  
    }  
}
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

27

27

toString() (III)



Der Aufruf:

```
Adresse adr = new Adresse("Max Mustermann", "Mönkhofer  
Weg", 239, 23562, "Lübeck");  
  
System.out.println(adr);
```

Erzeugt dann nicht, Adresse@33f42b49
sondern, die für den Leser gebräuchlichere Form:

```
Max Mustermann  
Mönkhofer Weg 239  
23562 Lübeck
```

Merke: Die `toString` Methode definiert eine textuelle Repäsentation, der Wertebelegung eines Referenztyps. Sie wird immer aufgerufen, wenn ein Objekt als Zeichenkette dargestellt werden soll.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

28

28

Semantik bei Referenztypen



Zuweisung

- Zuweisung kopiert lediglich die Referenz nicht das Objekt
- Nach einer Zuweisung von *a* (Referenz auf ein Objekt *o*) an *b* zeigen also *a* und *b* auf das Objekt *o*.
- Soll tatsächlich kopiert werden, muss dies mit der **clone** Methode erfolgen.

Gleichheit

- Es wird getestet ob die Referenzen gleich sind,
- nicht ob die Inhalte gleich sind.
- Sollen nur die Inhalte verglichen werden, muss dies mit der **equals** Methode erfolgen.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

29

29

equals() (!)



Bei Referenztypen ist eine weitere Besonderheit zu beachten. Die Definition der Gleichheit. Werden zwei Referenzen miteinander verglichen, so prüft JAVA ob die Referenzen auf dieselbe Speicherstelle zeigen, nicht ob die Objekte dieselben Werte haben.

So ergibt der folgende Code die Ausgabe `true` (`adr1` ist gleich `adr2`)

```
Adresse adr1 = new Adresse("Max Mustermann", "Mönkhofer Weg", 239, 23562, "Lübeck");
```

```
Adresse adr2 = adr1;  
System.out.println(adr1 == adr2);
```

Dieser Code jedoch die Ausgabe `false` (`adr1` ungleich `adr2`)

```
Adresse adr1 = new Adresse("Max Mustermann", "Mönkhofer Weg", 239, 23562, "Lübeck");
```

```
Adresse adr2 = new Adresse("Max Mustermann", "Mönkhofer Weg", 239, 23562, "Lübeck");
```

```
System.out.println(adr1 == adr2);
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

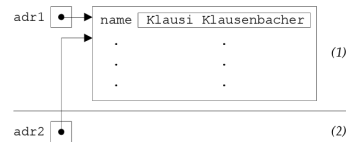
30

30

equals() (II)



Im ersten Fall wird einfach nur die Referenz kopiert.



Im zweiten Fall wird ein neues Objekt `adr2` angelegt, das (zufällig) dieselben Inhalte wie das Objekt `adr1` hat. Die Inhalte stehen jedoch an unterschiedlichen Stellen im Hauptspeicher.



Der Gleichheitsoperator `==` ist jedoch auf Referenzen (Identität) und nicht auf Inhalten definiert (Wertgleichheit). JAVA selber kann nicht auf Wertgleichheit vergleichen. Jedoch kann man dies für eigene Datentypen mittels einer `equals` Methode selber implementieren.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

31

31

equals() (III)



Hierzu muss im selber definierten Datentyp eine `equals` Methode eingebaut werden, die Komponenten (primitive Datentypen) paarweise miteinander auf Gleichheit vergleicht.

```
class Adresse {
    public String name;
    public String strasse;
    public int hausnummer
    ...

    public boolean equals(Adresse adr) {
        return this.name.equals(adr.name) &&
               this.strasse.equals(adr.strasse) &&
               this.hausnummer == adr.hausnummer;
    }
}
```

Der folgende Code erzeugt dann als Ausgabe `true`, d.h. `adr1` und `adr2` sind wertgleich jedoch nicht referenzgleich.

```
Adresse adr1 = new Adresse("Max Mustermann", "Mönkhofer Weg", 239, 23562, "Lübeck");

Adresse adr2 = new Adresse("Max Mustermann", "Mönkhofer Weg", 239, 23562, "Lübeck");

System.out.println(adr1.equals(adr2)); // Keine Anw. des == Operators
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

32

32

clone() (I)



Wie sie gesehen haben, wird mit dem Zuweisungsoperator = bei Referenztypen nur die Referenz, aber nicht die Inhalte dupliziert. Bei primitiven Datentypen werden hingegen tatsächlich die Inhalte dupliziert. Daher ist das Verhalten von Referenztypen und primitiven Datentypen bspw. bei Methodenaufrufen ein anderes (Stichwort: Call by Reference Verhalten, vgl. Unit 2).

Für unser Adressdatentyp sähe eine clone Methode beispielsweise wie folgt aus:

```
class Adresse {  
    public String name;  
    public String strasse;  
    public int hausnummer;  
  
    ...  
  
    public Adresse clone() {  
        return new Adresse(this.name, this.strasse, this.hausnummer);  
    }  
}
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

33

33

clone() (II)



Und könnte wie folgt aufgerufen werden:

```
Adresse adr1 = new Adresse("Max Mustermann", "Mönkhofer  
Weg", 239, 23562, "Lübeck");
```

```
Adresse adr2 = adr1.clone();
```



Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

34

34

Klassenvariablen



- Sie lernen nun die Bedeutung des Schlüsselworts **static** kennen.
- Sie haben bislang gelernt, dass Datenfelder (Variablen) eines Referenztyps ohne das Schlüsselwort **static** deklariert wurden.
- Derartige Datenfelder gehören immer zu genau einem Objekt welches mit dem **new** Operator erzeugt wird.

```
public class Adresse {  
    public String vorname;  
    public String nachname;  
    ...  
}
```

```
Adresse adr1 = new  
Adresse("Max", "Mustermann",  
...);  
  
Adresse adr2 = new  
Adresse("Maren",  
"Musterfrau", ...);
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

35

35

Klassenvariablen und -methoden



- Datenfelder können sich aber auch auf alle Objekte einer Klasse, also die Klasse selber beziehen.
- Diese Datenfelder gelten dann für alle Objekte einer Klasse und werden bei der Datenfeld Deklaration durch das Schlüsselwort **static** gekennzeichnet.
- Gleiches gilt für Methoden.

```
public class Adresse {  
    public static String vorname;  
    public String nachname;  
    ...  
}
```

```
Adresse adr1 = new Adresse("Max",  
"Mustermann", ...);  
  
Adresse adr2 = new  
Adresse("Maren", "Musterfrau",  
...);  
  
System.out.println(adr1.vorname);
```

Maren (obwohl doch adr1 als Max Mustermann instantiiert wurde).

Merke: Änderungen an als **static** deklarierten Datenfeldern eines Referenztyps wirken sich auf **ALLE** Objekte dieses Referenztyps aus.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

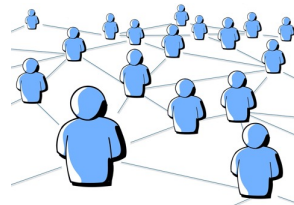
36

36

Beispiel: Personen zählen



- Sie sollen nun einen Referenztyp Person entwickeln, der den Vor- und Nachnamen einer Person speichern und ausgeben kann.
- Zusätzlich soll im Referenztyp mitgezählt werden, die wievielte von wie vielen insgesamt angelegten Personen diese Person ist. Auch diese Information soll in folgender Form ausgegeben werden.



Max Mustermann (3/1089)

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

37

37

Personen zählen - Lösung



```
class Person {
    public String vorname;
    public String nachname;
    public static int total;           // Zählt alle angelegten Personen Objekte
    public int meine_nr;

    public Person(String vn, String nn) {
        this.vorname = vn;
        this.nachname = nn;
        this.meine_nr = ++Person.total; // auch möglich this.total
    }

    public String toString() {
        return this.vorname + " " + this.nachname +
            " (" + this.meine_nr + "/" + Person.total + ")"; // auch möglich this.total
    }
}

public class Beispiel {
    public static void main(String[] args) {
        Person p1 = new Person("Max", "Mustermann");
        Person p2 = new Person("Maren", "Musterfrau");
        System.out.println(p1);
        System.out.println(p2);
    }
}
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

38

38

Miniübung:



Gegeben seien zwei Felder `a` und `b` vom Typ `int[]`.

- Warum kann man die beiden Felder `a` und `b` nicht mittels `a == b` vergleichen?
- Wie könnte ein Programmstück aussehen, das beide Felder miteinander vergleicht? Dabei seien zwei Felder genau dann gleich, wenn sie die gleiche Länge haben und alle ihre Komponenten paarweise übereinstimmen.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

39

39

Miniübung:



Welche der folgenden Ausdrücke wird zu `true` oder `false` ausgewertet?

```
Adresse adr1 = new Adresse("Mustermann", "23562 HL");
Adresse adr2 = adr1;
Adresse adr3 = adr1.clone();
Adresse adr4 = new Adresse("Mustermann", "23562 Lübeck");
Adresse adr5 = new Adresse("Mustermann", "23562 HL");
```

<code>adr1 == adr3</code>	false	<code>adr3 == adr1</code>	false
<code>adr2 == adr1</code>	true	<code>adr1.equals(adr2)</code>	true
<code>adr3.equals(adr1)</code>	true	<code>adr1.equals(adr3)</code>	true
<code>adr1 == adr4</code>	false	<code>adr4 == adr1</code>	false
<code>adr4.equals(adr1)</code>	false	<code>adr1.equals(adr4)</code>	false
<code>adr5.equals(adr1)</code>	true	<code>adr1.equals(adr5)</code>	true
<code>adr3.equals(adr3)</code>	true	<code>adr3 == adr3</code>	true

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

40

40

Miniübung:



Der Adressdatentyp hatte einen Email und Kommentarbestandteil. Diese Anteile sollen nur ausgegeben werden, wenn Sie auch definiert sind, d.h. ungleich der leeren Zeichenkette sind. Wie müssen Sie ihre toString Methode schreiben?

```
class Adresse {  
    public String name;           public String wohnort;  
    public String strasse;        public String mail;  
    public int hausnummer;        public String kommentar;  
    public int postleitzahl;  
  
      
  
}
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

41

41

Zusammenfassung



- **Arrays**
 - Semistatische Datenstruktur
 - Deklaration und Initialisieren von Arrays
 - Eindimensionale Arrays
 - Zwei- und mehrdimensionale Arrays
 - Verhalten wie ein Referenzdatentyp
- **Klassen**
 - Referenzdatentypen
 - Erzeugen von Objekten mit dem new Operator
 - Unterschiede zu primitiven Datentypen
 - Wertegleichheit und Referenzgleichheit
 - Clonen von Objekten
 - Statische und nicht statische Datenfelder



Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

42

42

Programmieren trainieren

Ergänzende Aufgaben zum Trimm-Dich-Pfad



Kapitel 6 (Arrays)

- W6.1: Wochentag
- W6.2: Tankfüllung
- W6.3: Rückwärtsausgabe
- W6.4: Bildvergrößerung
- W6.5: Partnervermittlung
- W6.6: Sitzplatzreservierung
- W6.7: Platztausch
- W6.8: Minimale Distanz
- W6.9: Morsecode
- W6.11: Spiegeln
- W6.12: Reflexion
- W6.14: Bild umdrehen und invertieren
- ...

Nur eingeschränkt zu empfehlen

- W6.10, 6.14, 6.19, 6.24

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

43

43

Programmieren trainieren

Ergänzende Aufgaben zum Trimm-Dich-Pfad



Kapitel 8 (Objektorientierung, Klassen)

- W8.1: Schrittzähler
- W8.2: Body-Mass-Index
- W8.3: Songtextsuche
- W8.4: Passwortklasse
- W8.5: Kopffitness
- W8.6: Fernbedienung
- W8.7: Stoppuhr
- W8.9: Tic Tac Toe
- W8.10: Zwischenablage
- W8.11: Temperaturgraph
- W8.13: Verschlüsselung
- ...

Nur eingeschränkt zu empfehlen

- W8.8, 8.12

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

44

44

Themen dieser Unit





Referenzdatentypen

- Felder (Arrays)
- Klassen

Collections

- Listen
- Mappings

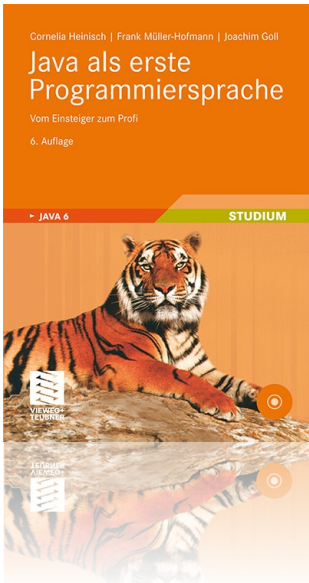
Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

45

45

Zum Nachlesen ...





Kapitel 18 Collections

Abschnitt 18.3 Listen

List

Abschnitt 18.6 Verzeichnisse

Map

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

46

46

Collections



- Im Rahmen der Programmierung benötigt man immer wieder Datenstrukturen, die sich stark ähneln.
- Z.B. muss man häufig eine Menge von Objekten in einer Liste speichern (z.B. alle Adressen von Studierenden), HTML Seiten werden in Web-Browsern bspw. häufig als Bäume verwaltet.
- Die Datenstrukturen sind dabei nicht abhängig davon, was für Arten von Objekten gespeichert werden.
- Die Art und Weise des Zugriffs ist ausschließlich abhängig von der Datenstruktur und nicht abhängig von den zu speichernden Objekttypen.
- In JAVA werden diese Arten von Datenstrukturen als Collections bezeichnet.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

47

47

Collections



Datenstrukturen

- Verwaltung von Mengen von Daten
- Daten werden gekapselt abgelegt
- Zugriff auf die Daten über spezielle Methoden

Unterschiede zu Arrays

- Müssen nicht typerein sein (sollten aber typerein genutzt werden)
- Können zur Laufzeit in Ihrer Größe verändert werden
- Und sind damit flexibler einsetzbar

Ausprägungen

- **List** – dynamische Liste
- **Map** – Key, Value Paare

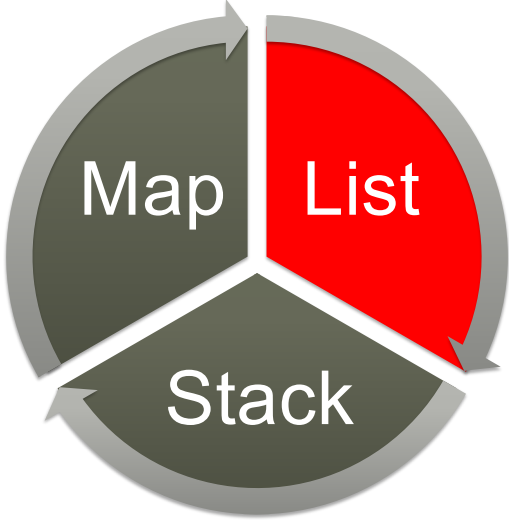
Weitere werden in dieser LV nicht behandelt.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

48

48

Collections



Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

49

49

Collections

List

- Lineare Liste
- Elemente beliebigen Typs (können jedoch auf einen Referenztyp eingeschränkt werden)
- Länge zur Laufzeit veränderbar
- Einfügen und Löschen von Elementen
- Sequentieller und wahlfreier Zugriff



Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

50

50

Collections

List – Einfügen und Löschen von Elementen



<pre>public boolean add(Object o)</pre>	Fügt ein Element <i>o</i> der Liste am Ende hinzu
<pre>public boolean add(int i, Object o)</pre>	Fügt ein Element <i>o</i> der Liste an der Stelle <i>i</i> hinzu. Alle folgenden Elemente werden weitergeschoben.
<pre>public Object set(int i, Object o)</pre>	Liefert als Rückgabe ob add erfolgreich war oder nicht.
<pre>set Ersetzt ein Element an der Stelle i (vorheriges Element wird zurück gegeben)</pre>	
<pre>public Object remove(int i)</pre>	Löscht das Element an der Stelle <i>i</i> in der Liste und liefert es zurück.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

51

51

Collections

List – Informationen über Listen



<pre>public Object get(int i)</pre>	Liefert das <i>i</i> . Element der Liste.
<pre>public boolean isEmpty()</pre>	Prüft, ob eine Liste leer ist.
<pre>public int size()</pre>	Gibt die Anzahl der Elemente in einer Liste zurück.

Alle weiteren Methoden dieser Datenstruktur finden Sie unter:
<http://download.oracle.com/javase/7/docs/api/java/util/List.html>

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

52

52

Collections

List – Miniübung I



```
List v = new LinkedList();  
System.out.println(v);
```

[]

```
v.isEmpty();
```

true

```
v.size();
```

0

```
v.add(5);
```

```
v.add(6);
```

[5, 6]

```
v.isEmpty();
```

false

```
v.size();
```

2

```
v.get(1);
```

6

```
v.add(1, "Einschub");
```

[5, Einschub, 6]

```
v.remove(1);
```

[5, 6]

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

53

53

Collections

List – Sequentieller Elementzugriff



Sequentieller Elementzugriff auf eine Liste ist mittels eines Iterators möglich.

```
public Iterator iterator()
```

Liefert einen
Iterator über eine
Liste.

Ein Iterator bietet zwei Methoden an:

```
public boolean hasNext()
```

True, wenn noch
weitere Elemente in der
Aufzählung, sonst
False.

```
public Object next()
```

Liefert das
nächste Element
der Aufzählung

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

54

54

Collections

List – Miniübung III



```
List v = new ArrayList();  
v.add(1);  
v.add(2);  
v.add(3);
```

[1, 2, 3]

```
Iterator e = v.iterator();
```

[1,2,3]

```
e.hasNext();
```

true

```
e.next();
```

1 und [1, 2, 3]

```
e.hasNext();
```

true

```
e.next();
```

2 und [1, 2, 3]

```
e.hasNext();
```

true

```
e.next();
```

3 und [1, 2, 3]

```
e.hasNext();
```

false

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

55

55

Collections

List – Iterator Schleife über eine Liste



```
List v = new LinkedList();  
v.add("Eins");  
v.add("Drei");  
v.add(1, "Zwei");
```

```
Iterator it = v.iterator();
```

```
while (it.hasNext()) {  
    System.out.println(it.next());  
}
```

Eins

Zwei

Drei

Diese Variante ohne Enumerator ginge auch:

```
for (int i = 0; i < v.size(); i++) {  
    System.out.println(v.get(i));  
}
```

Diese Variante mit foreach Schleife ginge auch:

```
for (Object o : v) System.out.println(o);
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

56

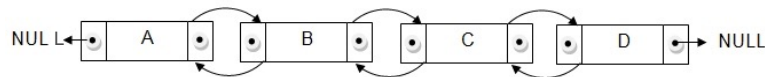
56

Unterschiedliche Implementierungen von Listen hinter derselben Schnittstelle



Softwaretechnisch können Listen auf unterschiedlichste Arten implementiert werden.

In JAVA bietet es sich an, Objekte jeweils mit einer Referenz auf den Vorgänger und den Nachfolger in einer Liste zu verlinken. Dieses Prinzip nennt man `LinkedList`.



Man kann aber auch die Datenstruktur Array dazu nutzen, um eine Liste im Hauptspeicher zu speichern. Dieses Prinzip nennt man dann `ArrayList`.



Für den Zugriff auf die Liste und den Umgang mit der Liste ändert dies nichts.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

57

57

LinkedList und ArrayList Vorteile und Nachteile



	Vorteil	Nachteil
<code>LinkedList</code>	• Schnelles Einfügen	• Langsamer wahlfreier Zugriff • höherer Speicherverbrauch
<code>ArrayList</code>	• Schneller wahlfreier Zugriff • geringer Speicherverbrauch	• Langsames Einfügen

Da eine `ArrayList` auf die semidynamische Datenstruktur Array zurückgreift, kann eine Einfügeoperation nur sehr aufwändig realisiert werden, da ein Array zur Laufzeit nicht vergrößert werden kann.

Es muss erst ein größeres Array angelegt werden, dann alle Daten aus dem alten Array umkopiert werden, und dann das alte Array gelöscht werden.

Da Listen häufig sequentiell (also nicht wahlfrei) durchlaufen werden, ist es daher ratsam, `LinkedLists` zu nutzen. Erst wenn Laufzeitprobleme oder Speicherprobleme bei großen Listen auftreten, sollte man den Einsatz von `ArrayList` in Erwägung ziehen.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

58

58

LinkedList und ArrayList implementieren beide die List Schnittstelle



Egal welche Implementierung genutzt wird. Auf die Datenstruktur sollte man immer nur über die Schnittstelle `List` zugreifen.

Dies ermöglicht es, nachträglich die zugrunde liegende Listenimplementierung in einer Zeile zu ändern, ohne den Rest der Programmierung anpassen zu müssen.

Die `List` Schnittstelle ist eine klassische objektorientierte Lösung. Wie diese Mechanismen im Einzelnen funktionieren, werden wir im weiteren Verlauf der Vorlesung noch behandeln.

```
// Listen daher bitte immer so anlegen
List alist = new ArrayList();
List llist = new LinkedList();

// Niemals so (obwohl der Compiler nicht meckern würde)
ArrayList alist = new ArrayList();
LinkedList llist = new LinkedList();
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

59

59

Miniübung:



Gegeben sei folgende Liste:

```
List v = new LinkedList();
for (int i = 0; i < 10; i++) v.add(i);
```

Entwickeln Sie eine Methode `invert`, um eine `Liste` oben angegebener Art rückwärts in folgender Form als `String` zurückzugeben:

9-8-7-6-5-4-3-2-1-0

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

60

60

Miniübung:



Entwickeln Sie bitte zwei Methoden **firstElement** und **lastElement**, die das jeweils erste und letzte Element einer Liste zurückliefern.

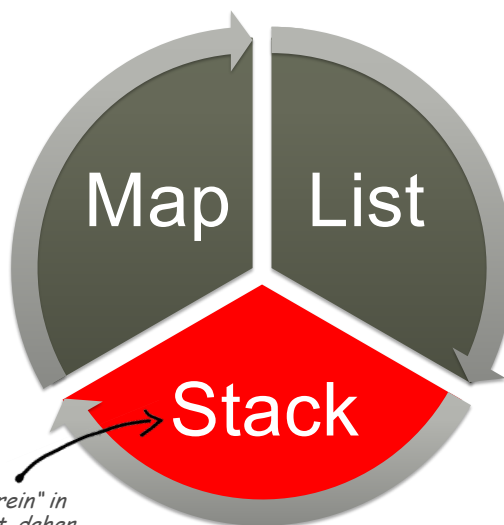
Ist die Liste leer oder `null` sollen beide Methoden `null` als Ergebnis liefern.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

61

61

Collections



*Nicht so ganz "astrein" in
Java implementiert, daher
überspringen wir das hier.*

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

62

62

Collections Stack



Kellerspeicher

Elemente beliebigen Typs (kann auf einen Referenztyp eingeschränkt werden)

Länge zur Laufzeit beliebig veränderbar.

Zugriff nach dem LIFO Prinzip
Last In First Out



Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

63

63

Collections Stack – Operationen



```
public void push(Object o)
```

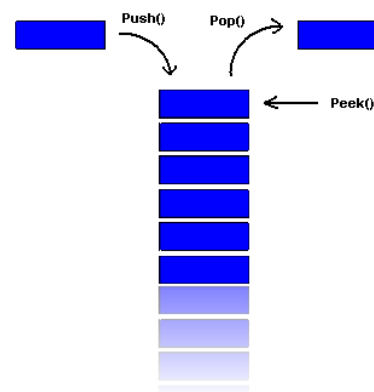
Legt ein Element *o* auf den Stack.

```
public Object peek()
```

Liest das oberste Element des Stacks.

```
public Object pop()
```

Liest das oberste Element des Stacks und löscht es vom Stack.



Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

64

64

Collections

Stack – Informationen über Stack



```
public boolean isEmpty()
```

Prüft, ob ein Stack leer ist.

```
public int size()
```

Gibt Anzahl der Elemente eines Stack zurück.

Alle weiteren Methoden dieser Datenstruktur finden Sie unter:

<http://download.oracle.com/javase/7/docs/api/java/util/Stack.html>

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

65

65

Collections

Stack – Miniübung



```
Stack s = new Stack();  
s.push(1);  
s.push(2);  
s.push(3);
```

s:[1, 2, 3]

```
s.size()
```

3

```
s.isEmpty();
```

false

```
s.peek();
```

3 und s:[1, 2, 3]

```
s.pop();
```

3 und s:[1, 2]

```
s.pop();
```

2 und s:[1]

```
s.pop();
```

1 und s:[]

```
s.isEmpty();
```

true

```
s.pop();
```

Fehler: EmptyStackException

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

66

66

Miniübung:



Gegeben sei folgende Liste:

```
List v = new LinkedList();  
for (int i = 0; i < 10; i++) v.add(i);
```

Entwickeln Sie eine Methode `invert`, um eine `List` oben angegebener Art rückwärts in folgender Form als `String` zurückzugeben. Nutzen Sie die Datenstruktur `Stack`, um diese Methode zu implementieren.

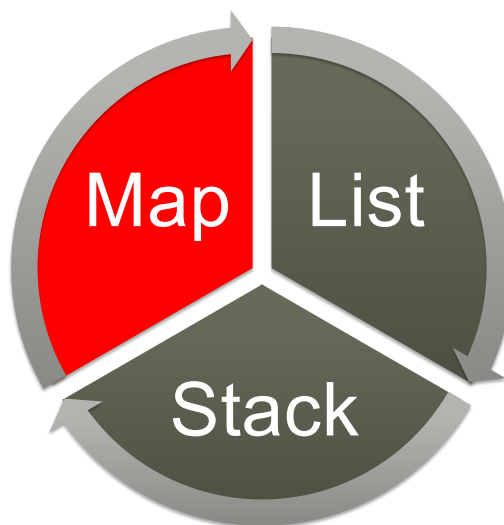
9-8-7-6-5-4-3-2-1-0

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

67

67

Collections



Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

68

68

Collections

Map

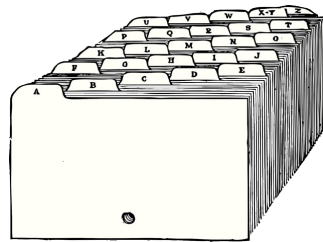


Assoziativer Speicher (A verweist auf B)

Elemente beliebigen Typs (können auf einen Referenztyp eingeschränkt werden)

Länge zur Laufzeit beliebig veränderbar.

Zugriff nach dem Key Value Prinzip



Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

69

69

Collections

Map – Einfügen und Löschen von Elementen

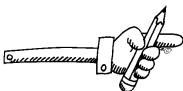


```
public Object put(Object key, Object value)
```

Fügt das **key value** Paar in die Map ein. Falls bereits ein **key value** Paar existiert wird **value** ausgetauscht und der alte **value** Wert zurückgegeben. Andernfalls wird **null** zurückgegeben.

```
public Object remove(Object key)
```

Löscht das **key value** Paar mit dem Schlüssel **key**. Gibt den unter **key** gespeichert Wert zurück oder **null**, falls der **key** nicht unter den Schlüsseln der Map gespeichert war.

Key  Value



KEY VALUE

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

70

70

Collections

Map – Zugriff auf Elemente



```
public Object get(Object key)
```

Liefert den **value** aus der Map, der unter **key** abgelegt wurde. Liefert **null**, wenn der Schlüssel nicht in der Hashtabelle abgelegt ist.

```
public Object getOrDefault(Object key, Object default)
```

Liefert den **value** aus der Map, der unter **key** abgelegt wurde, bzw. ein Defaultwert.

```
public boolean containsValue(Object value)
```

Liefert **true**, wenn **value** in den Werten der Map vorhanden ist. Andernfalls **false**.

```
public boolean containsKey(Object key)
```

Liefert **true**, wenn **key** in den Schlüsseln der Map vorhanden ist. Andernfalls **false**.

```
public List values()
```

Liefert alle Values der Map.

```
public Set keySet()
```

Liefert alle Schlüssel der Map.

```
public List<Map.Entry> entrySet()
```

Liefert alle Schlüssel/Wert Paare der Map.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

71

71

Collections

Map – Miniübung I



```
Map h = new HashMap();  
h.put('a', "Hello");  
h.put('c', "World");  
h.put('x', "_");  
System.out.println(h);
```

{ a=Hello, c=World, x=_ }

```
h.remove('c');
```

World und { a=Hello, x=_ }

```
h.remove('z');
```

null und { a=Hello, x=_ }

```
h.put('x', "World");
```

_ und { a=Hello, x=World }

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

72

72

Collections

Map – Miniübung II



```
Map h = new HashMap();  
h.put('a', "Hello");  
h.put('c', "World");  
h.put('x', "Strange");
```

```
{ a=Hello,  
  c=World,  
  x=Strange  
}
```

```
for(Object v : h.values()) {  
    System.out.println(v);  
}
```

```
Strange  
World  
Hello  
Reihenfolge kann variieren
```

```
for (Object k : h.keySet()) {  
    System.out.println(k);  
}
```

```
c  
a  
x  
Reihenfolge kann variieren
```

```
for (Map.Entry e : h.entrySet()) {  
    System.out.print(e.getKey() + ": ");  
    System.out.println(e.getValue());  
}
```

```
c: World  
a: Hello  
x: Strange  
Reihenfolge kann variieren
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

73

73

Unterschiedliche Implementierungen von Maps hinter derselben Schnittstelle



Softwaretechnisch können Maps (ähnlich wie Listen) auf verschiedene Arten realisiert werden. Die Frage ist, wie man mit dem Schlüssel umgeht.

In JAVA werden u.a. die folgenden zwei Varianten hierzu angeboten.

- Die Schlüssel werden mittels eines Baums gespeichert und sortiert (wie dies im Einzelnen funktioniert werden wir noch im weiteren Verlauf der Vorlesung sehen). Hierzu nutzt man die Datenstruktur **TreeMap**.
- Die Schlüssel werden gehashed. Dies ermöglicht einen unsortierten aber sehr schnellen Zugriff auf die Werte innerhalb der Map. Hierzu nutzt man die Datenstruktur **HashMap**.

Wie bei `LinkedList` und `ArrayList` über die Schnittstelle `List` angesprochen werden können, so können auch `HashMap` und `TreeMap` über die Schnittstelle **Map** angesprochen werden.

Der Unterschied soll an einem kleinen Beispiel deutlich gemacht werden.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

74

74

Beispiel: HashMap



```
// Anlegen einer Map
Map<String, String> telefonbuch = new HashMap<String, String>();

// Eintragen von Werten in die Map
telefonbuch.put("Peter", "0451-123456");
telefonbuch.put("Klaus", "0451-234156");
telefonbuch.put("Armin", "0451-623145");

// Ausgeben der Map (Telefonbuch)
for (String name : telefonbuch.keySet()) {
    System.out.print ("Nummer von " + name + ":\t");
    System.out.println(telefonbuch.get(name));
}
```

Ausgabe des Programms:

Nummer von Klaus: 0451-234156
Nummer von Armin: 0451-623145
Nummer von Peter: 0451-123456

Die Ausgabe der Telefonnummer erfolgt
in zufälliger Reihenfolge der Keys.

**Die Reihung ist bei einer HashMap
nicht mal vorhersagbar!**

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

75

75

Beispiel: TreeMap



```
// Anlegen einer Map
Map<String, String> telefonbuch = new TreeMap<String, String>();

// Eintragen von Werten in die Map
telefonbuch.put("Peter", "0451-123456");
telefonbuch.put("Klaus", "0451-234156");
telefonbuch.put("Armin", "0451-623145");

// Ausgeben der Map (Telefonbuch)
for (String name : telefonbuch.keySet()) {
    System.out.print ("Nummer von " + name + ":\t");
    System.out.println(telefonbuch.get(name));
}
```

Ausgabe des Programms:

Nummer von Armin: 0451-623145
Nummer von Klaus: 0451-234156
Nummer von Peter: 0451-123456

Die Ausgabe der Telefonnummer
erfolgt nun in Ordnung der Keys.

**Die Reihung ist somit bei einer
TreeMap vorhersagbar!**

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

76

76

Miniübung:



Gegeben sind diese beiden Arrays von Matrikelnummern und Namen. Führen Sie diese beiden Arrays in einer Map zusammen, so dass die Namen den Matrikelnummern zugeordnet werden.

Die Arrays sind korrekt geordnet – so dass die Zuordnungen passen.

```
int[] matrnrns = { 565432, 675938, 889554, 886532 };  
String[] names = { "Max Mustermann", "Maren Musterfrau",  
                  "Tessa Loniki", "Wilder Wutz" };
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

77

77

Typsicherheit bei Collections



- Sie haben Collections bislang als einen Sammelbehälter von Werten kennengelernt, der Werte beliebigen Typs (Object) aufnehmen kann.
- Häufig ist dies gar nicht erforderlich und auch nicht gewollt.
- Alle vorgestellten Datentypen können in einer typfreien bzw. typgebundenen Variante genutzt werden.
- Sie haben bislang nur die typfreie vorgestellt bekommen.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

78

78

Wie mache ich Collections typsicher?



Beispiel für List

```
List v = new LinkedList(); // Wertetypen in der Liste egal
```

```
List<Integer> v = new LinkedList<Integer>();  
// Es sind nur noch Integers (int) in der Liste zugelassen
```

Beispiel für Map

```
Map h = new HashMap(); // Wertetypen im Verzeichnis egal
```

```
Map<String, Double> h = new HashMap<String, Double>();  
// Es sind nur noch Zeichenketten als Key  
// und Fließkommazahlen (double) als Wert in der Map  
// zugelassen
```

Auf die genauen Hintergründe dieses Konzepts werden wir in der Vorlesung im Teil Generizität (2. Semester) eingehen.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

79

79

Wie mache ich Collections typsicher?



Primitiver Datentyp	Zu nutzender Datentyp in Collection (Referenztypentsprechung)
char	Character
int	Integer
short	Short
byte	Byte
boolean	Boolean
String	String
double	Double
float	Float

Der Java Compiler sorgt für die automatische Umsetzung von primitiven Datentypen in Referenztypen (**Autoboxing**).

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

80

80

Miniübung:



Gegeben seien folgende Quelltexte. Wie können Sie diese typischerer machen?

```
List v = new LinkedList();  
for (int i = 1; i <= 1000; i++) v.add((double)i);
```

```
Map h = new HashMap();  
for (int i = 1; i <= 1000; i++) h.put("Key " + i, Math.pow(i, 2));
```

```
Map h = new TreeMap();  
for (int i = 1; i <= 1000; i++) h.put("" + i, (" " + i).length());  
for (Map.Entry e: h.entrySet())  
    System.out.println(e.getKey() + ":" + e.getValue());
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

81

81

Miniübung:



Ziehen Sie nun 1000 Zufallszahlen zwischen 0 und 999. Zählen sie mittels einer Map, wie viele kleine [0, 333[, mittlere [333, 666[und große Zahlen [666, 1000] sie gezogen haben und geben sie das Ergebnis als Map aus.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

82

82

Miniübung:



Ziehen Sie nun nur noch 10 Zufallszahlen zwischen 0 und 999. Ordnen sie die gezogenen Zufallszahlen mittels einer Map, den Kategorien ‚small‘ [0, 333[, ‚medium‘ [333, 666[und ‚big‘ [666, 1000] zu. Geben sie diese Zuordnung als Map aus.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

83

83

Miniübung:



Gegeben sind eine Liste von Objekten des Typs Adresse:

```
public class Adresse {  
    public String vorname;  
    public String nachname;  
    public String strasse;  
    public int nummer;  
    public int plz;  
    public String ort;  
  
    public Adresse(...) { ...; }  
  
    public String toString() { return ...; }  
}
```

Entwickeln Sie nun bitte eine Methode zum Gruppieren von Listen von Adressen nach Postleitzahlen.

Die Methode soll eine Map zurückliefern, d.h. eine Abbildung von Postleitzahlen auf Liste von Adressen.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

84

84

Miniübung:





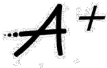
Lösung:

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

85

85

Zusammenfassung





- **List**
 - Volldynamische Datenstruktur einer Liste
 - Ausprägung als `LinkedList` und `ArrayList`
- **Stack**
 - Volldynamische Datenstruktur eines Kellerspeichers
- **Map**
 - Volldynamische Datenstruktur eines Assoziativspeichers
 - Key-Value Paare
 - Ausprägung als `HashMap` und `TreeMap`
- **Typsicherheit bei Collections**



Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

86

86

LET'S SUM IT ALL UP

*Programming
is simply about repetition*

repetition

repetition

repetition

...

87

Das (nicht so) „geheime“ Koch-Rezept



```
/**
 * Methodenkopf          => Parameter a, b, data, ...
 * 1. Was geht rein?      => Return type
 * 2. Was kommt raus?     => Return type
 * 3. Wie heißt die Logik? => Methodenname
 */
public static String[] nonNullStrings(String[] data) {
    // Methodenrumpf: Bearbeiten Sie (1) => (2) => (3) => (4)
    // Nutzen Sie nur bei Bedarf die [optionalen] Platzhalter

    // => [ggf. Guard Conditions, was kann nicht verarbeitet werden?]
    if (data.length == 0) return new String[0];

    // (1) Aggregatvariable (ist meist vom Return type)
    List<String> result = new ArrayList<>();

    // => [ggf. Hilfsvariablen, ergeben sich meist aus Schritt (3 + 4)]
    // => [ggf. Preprocessing, ergibt sich meist aus Schritt (3 + 4)]

    // (3) Data loop (oft foreach über eine Sequenz der Parameterliste)
    for (String d : data) {
        // (4) Logik (hier wird es problemspezifisch, Kopf anschalten!)
        if (d != null) {
            result.add(d);
        }
    }

    // => [ggf. Postprocessing, ergibt sich meist aus/im Anschluss an Schritt (3 + 4)]
    String[] r = result.toArray(new String[0]);

    // (2) Return (Rückgabe der Methode, ist immer vom Return type)
    return r;
}
```

Anschließend können
Sie diese Basis-Schritte

- zusammenfassen,
- optimieren,
- umordnen,
- usw.

um Ihren persönlichen
Stil und Grad der
„Eleganz“ zu erreichen.

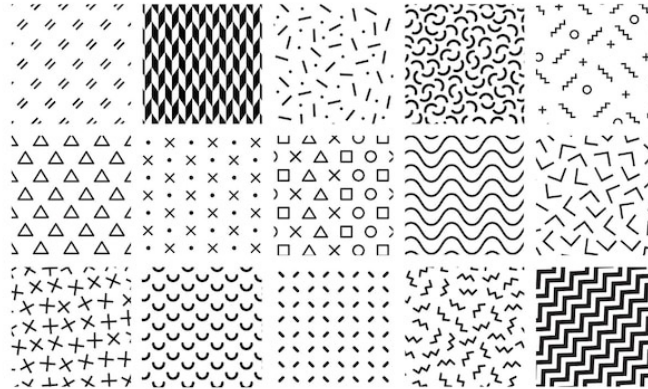
*Alle Aufgaben lieben
sich bisher mit diesem
Pattern lösen!!!*

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

88

88

Suchen Sie nach „Abkürzungen“



Anmerkung: Das gezeigte Grundpattern wird meist in **8** höherwertigen Pattern angewendet. Es macht Sinn, sich bei Problemlösungen von diesen Pattern leiten zu lassen und entlang dieser Pattern Probleme zu dekomponieren.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

89

89

1. Generator - Pattern:



Erzeugt aus nicht
sequentiellen Eingabedaten
eine Sequenz von Werten



```
public static List<Integer> range(int n) {  
    List<Integer> values = new ArrayList<>();  
    for (int i = 0; i < n; i++) {  
        values.add(i);  
    }  
    return values;  
}
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

90

90

2. Filter - Pattern:



Selektiert aus einer Sequenz von Werten diejenigen Werte, die einem spezifischem Kriterium entsprechen.



```
public static List<Integer> evens(List<Integer> values) {  
    List<Integer> filtered = new ArrayList<>();  
    for (int v : values) {  
        if (v % 2 == 0) { // Filterbedingung  
            filtered.add(v);  
        }  
    }  
    return filtered;  
}
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

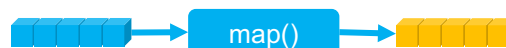
91

91

3. Mapping - Pattern:



Wendet auf einer Sequenz von Werten für jeden Wert, eine Transformation an.



```
public static List<Integer> lengths(List<String> words) {  
    List<Integer> processed = new ArrayList<>();  
    for (String w : words) {  
        processed.add(w.length()); // Transformation  
    }  
    return processed;  
}
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

92

92

4. Reduction - Pattern:



Wendet auf einer Sequenz von Werten elementweise eine Reduktionsoperation an, die einen einzigen Wert liefert.



```
public static int totalLength(List<String> words) {  
    int length = 0;  
    for (String w : words) {  
        length += w.length(); // Reduktion  
    }  
    return length;  
}
```

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

93

93

5. For all ($\forall$) - Pattern:



Prüft auf einer Sequenz von Werten, ob alle Elemente eine Bedingung erfüllen.



```
public static boolean nonNull(String[] words) {  
    for (String w : words) {  
        if (w == null) return false; // Verstoß  
    }  
    return true; // Kein Verstoß gefunden  
}
```

Anmerkung: Das forAll-Pattern ist eigentlich eine Spezialform des Reduction Pattern dessen Rückgabetypp boolean lautet.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

94

94

6. Exists ($\exists$)- Pattern:



Prüft auf einer Sequenz von Werten, ob mindestens ein Element eine Bedingung erfüllt.



```
public static boolean existsNull(String[] words) {  
    for (String w : words) {  
        if (w == null) return true; // Gefunden  
    }  
    return false; // Gesuchtes nicht gefunden  
}
```

Anmerkung: Das Exists-Pattern ist eigentlich eine Spezialform des Reduction Pattern dessen Rückgabetyt boolean lautet.

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

95

95

7. Counting - Pattern:



Zählt in einer Sequenz von Werten, wie viele Werte ein Kriterium erfüllen.



```
public static Map<String, Integer> countWords(List<String> words) {  
    Map<String, Integer> counted = new TreeMap<>();  
    for (String w : words) {  
        counted.put(w, counted.getOrDefault(w, 0) + 1);  
    }  
    return counted;  
}
```

Anmerkung: Das Counting Pattern ist eigentlich eine Spezialform des Reduction Pattern dessen Rückgabetyt Map<?, Integer> lautet.

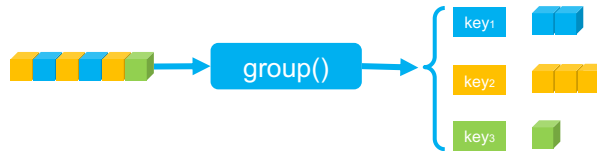
Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

96

96

8. Grouping - Pattern:

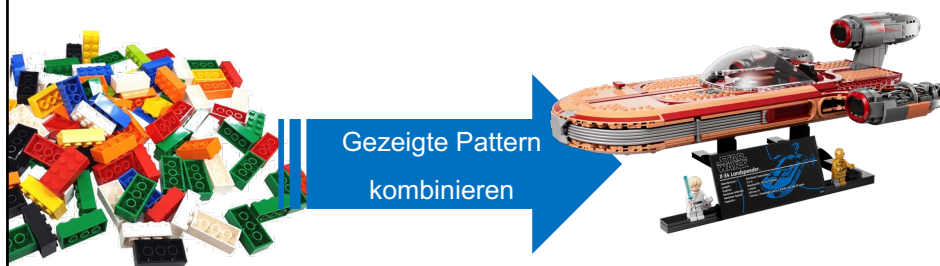
Gruppert Werte in einer Sequenz anhand eines Kriterium.



```
public static Map<Integer, List<String>> groupByWordLength(List<String> words) {  
    Map<Integer, List<String>> grouped = new TreeMap<>();  
    for (String w : words) grouped.put(w.length(), new ArrayList<>()); // Keys anlegen  
    for (String w : words) grouped.get(w.length()).add(w); // values einfügen  
    return grouped;  
}
```

Anmerkung: Das Grouping Pattern ist eigentlich eine Spezialform des Reduction Pattern dessen Rückgabebetyp `Map<?, List<?>>` lautet.

Mit diesen Basis Pattern können Sie „Lego“ spielen, um höherwertige Logik zu implementieren



Challenge: Gehen Sie alle Praktikumsaufgaben und Codepads der Vorlesung durch und prüfen Sie welches Pattern welchen Aufgaben zuzuordnen wäre.

Programmieren trainieren

Ergänzende Aufgaben zum Trimm-Dich-Pfad



Kapitel 9 (Referenzdatentypen)

- W9.1: Kreisklasse
- W9.2: Mathematischer Bruch
- W9.3: Songtextsuche
- W9.5: Digitaler Bilderrahmen
- W9.6: Musikalbenanwendung
- W9.7: Koch-Website
- W9.9: Flughafen-Check-in
- W9.12: Fototagebuch

Nur eingeschränkt zu empfehlen

- W9.3, 9.4, 9.8, 9.10, 9.11, 9.13

Prof. Dr. rer. nat. Nane Kratzke
Praktische Informatik und betriebliche Informationssysteme

99