Java lernen – Objektdatentypen

HPI-Team

Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

Objektdatentypen vs. Primitive Datentypen

Objektdatentypen	*Primitive Datentypen*
Jede Klasse ist ein Datentyp - Klassen aus der Java Bibliothek - Klassen von anderen Programmierern - Eigene Klassen	Fest von Java vorgegeben - short, int, long - float, double - boolean, char, byte
Besitzen Methoden und Attribute.	Keine Methoden und keine Attribute
Speicher: Referenz auf Objekt	Speicher: eigentlicher Wert
Beispiel: Point p = new Point();	Beispiel: int var = 0;

Sammlungen - Collections

- Bisher: Arrays um mehrere Elemente eines Typs zu speichern
  - fixe Größe
  - durchnummerierte Elemente

- Neu: Sammlungen
  - ArrayList
  - LinkedList
  - HashMap
  - HashSet

Generics

- Sammlungen akzeptieren Elemente beliebiger Datentypen
  - ➔ zugrundeliegender Datentyp: Object

- Genauere Typeinschränkung möglich:
  - Generics: Typ wird in "<>" für die Sammlung definiert

```
ArrayList<Typ> sammlung = new ArrayList<>();
```

- Generics funktionieren nicht mit primitiven Datentypen

Objektdatentypen – ”Wrapper”

Jeder primitive Datentyp hat einen passenden Objektdatentyp

primitiver Datentyp	*zugehöriger Objektdatentyp*
int	Integer
double	Double
float	Float
long	Long
boolean	Boolean
char	Character

Sammlungen - ArrayList

- Array mit veränderlicher Größe
- Wrapper um Object[]
- Verwaltet die Vergrößerung des zugrunde liegenden Arrays
- Stellt Methoden zur Verwaltung des Arrays zur Verfügung
  - add – Hinzufügen eines Elements
  - addAll – Hinzufügen einer anderen Sammlung
  - contains – Prüfen ob ein Element enthalten ist
  - ...
- Schneller Zugriff auf Elemente an beliebiger Position

Sammlungen - ArrayList

Unterschiede zu Arrays

Array	*ArrayList*
Feste Größe	Dynamische Größe
Zugriff über array[0]	Zugriff über arrayList.get(0);
Ermittlung der Größe über: array.length	Ermittlung der Größe über: arrayList.size()
Löschen nicht möglich	Löschen mit arrayList.remove(0)

Sammlungen - ArrayList - Beispiel

```
ArrayList<String> liste = new ArrayList<>();
System.out.println(liste.size());
liste.add("Hallo openHPI");
liste.add("Hallo Welt");
System.out.println(liste.size());
System.out.println(liste.get(0));
liste.remove(0);
System.out.println(liste.get(0));
```

0

2

Hallo openHPI

Hallo Welt

Sammlungen - ArrayList - Beispiel

```
ArrayList<String> liste = new ArrayList<>();
liste.add("Hallo openHPI");
liste.add("Hallo Welt");
System.out.println(liste.contains("Hallo Welt"));
liste.remove("Hallo Welt");
System.out.println(liste.contains("Hallo Welt"));
```

true

false

Sammlungen - LinkedList

- Verkettete Objekte
- Vorteilhaft wenn häufig Elemente am Anfang oder in der Mitte der Liste eingefügt werden müssen
- Beispiel:

Sammlungen - LinkedList

- LinkedList und ArrayList stellen die gleichen Methoden zur Verfügung
- Durch die unterschiedliche Struktur lassen sich bestimmte Operationen auf der einen oder der anderen Liste effizienter durchführen
- Beispiele:

ArrayList	*LinkedList*
Direkter Zugriff über Index möglich – get(int index)	Durchlaufen aller vorherigen Elemente notwendig – get(int index)
Hinzufügen von Elementen im vorderen Bereich ist teuer – add(int index, E element)	Hinzufügen von Elementen im vorderen Bereich ist günstig – add(int index, E element)

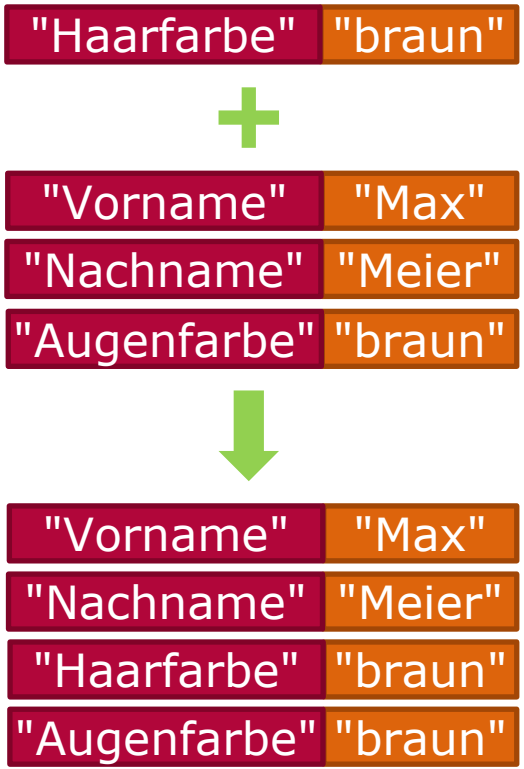
Sammlungen - HashMap

- Schlüssel und Werte
  - Generell:

  - Konkretes Beispiel:

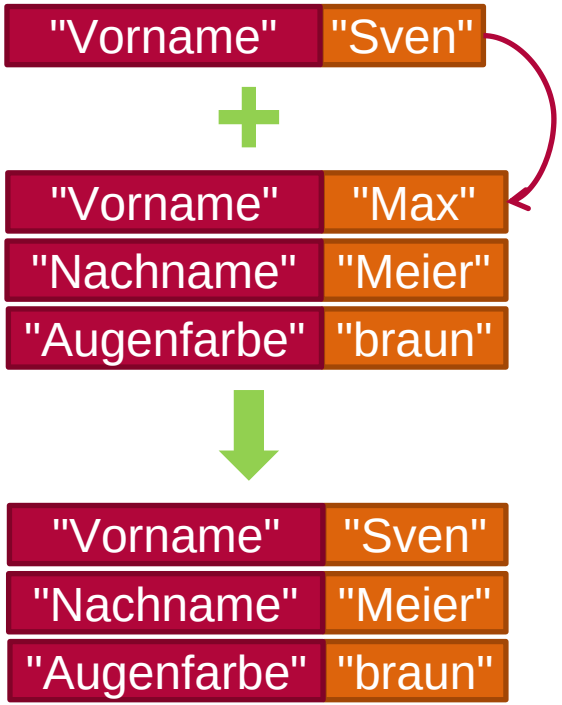
Sammlungen – HashMap

- Beliebig viele Paare
- Unsortiert



Sammlungen – HashMap

- Keine Duplikate



wird über-
schrieben

Sammlungen - HashMap

Beispiel

```
HashMap<String, String> map = new HashMap<>();
System.out.println(map.size());
map.put("Schluessel1", "Wert1");
map.put("Schluessel2", "Wert2");
System.out.println(map.get("Schluessel2"));
```

0

Wert2

Sammlungen – HashMap

Beispiel

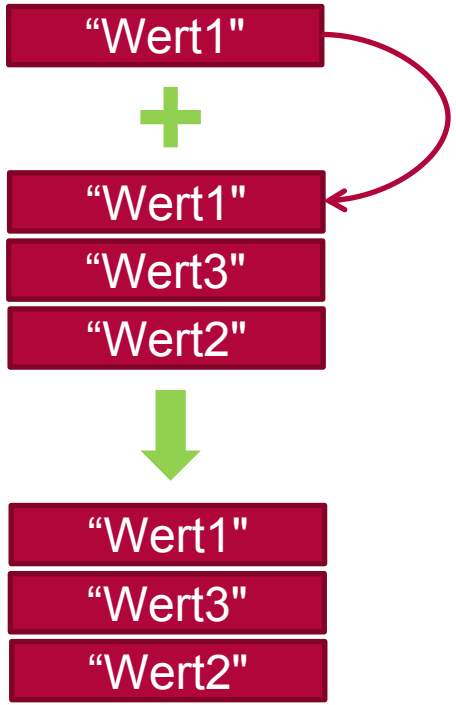
```
HashMap<String, String> map = new HashMap<>();
map.put("Schluessel1", "Wert1");
System.out.println(map.containsKey("Schluessel1"));
System.out.println(map.containsValue("Wert1"));
```

true

true

Sammlungen - HashSet

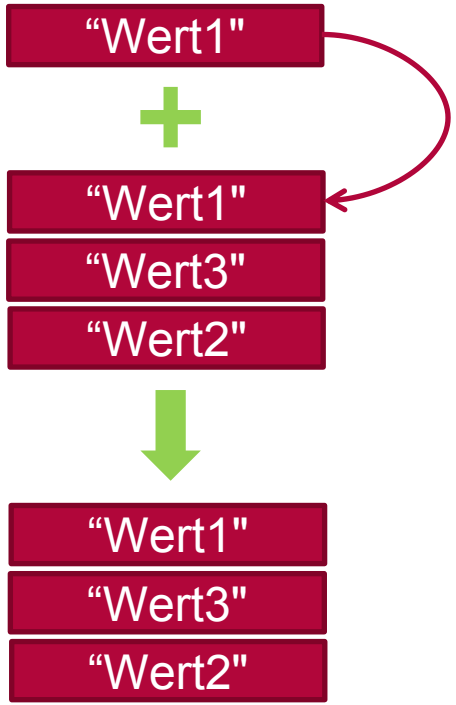
- Keine Duplikate
- Unsortiert
- Beispiel:



Konflikt:
Neuer Wert wird nicht hinzugefügt

Sammlungen - HashSet

- Keine Duplikate
- Unsortiert
- Beispiel:



Konflikt:
Neuer Wert wird nicht hinzugefügt

Java lernen – foreach

HPI-Team

HPI Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

Rückblick auf Woche 2 - Operationen auf Arrays

Effektives Verwalten

- Bekanntes Konzept: Schleifen
- Besonders geeignet: `for`-Schleife
- Attribut `length` beinhaltet die Größe des Arrays

Beispiel:

```
1 public class MeineKlasse {
2     public static void main(String[] args) {
3         int[] array = new int[10];
4         for(int i=0; i < array.length; i++){
5             array[i] = i;
6         }
7     }
8 }
```

Von der for-Schleife zur foreach-Schleife

for-Schleife

```
1  public class MeineKlasse {
2      public static void main(String[] args) {
3          int[] array = {1, 2, 3, 4, 5};
4          for(int i=0; i < array.length; i++){
5              System.out.println(array[i]);
6          }
7      }
8  }
```

```
1
2
3
4
5
```

foreach - Schleife

```
1  public class MeineKlasse {
2      public static void main(String[] args) {
3          int[] array = {1, 2, 3, 4, 5};
4          for(int k : array){
5              System.out.println(k);
6          }
7      }
8  }
```

```
1
2
3
4
5
```

foreach - Schleife

Für jedes **Element** **in** **Sammlung**

```
for (ArrayDatentyp bezeichner : arrayName) {
    // Fuehre etwas aus
    System.out.println(bezeichner);
}
```

- Zählvariable wird nicht benötigt
- Zugriff auf das Element über den Bezeichner
- Anwendbar auf:
  - Einfache Arrays
  - Sammlungen

Weiteres Beispiel

```
1  import java.util.ArrayList;
2  public class HalloWelt {
3      public static void main(String[] args) {
4          ArrayList<String> aliste = new ArrayList<String>();
5          aliste.add("Hans");
6          aliste.add("Peter");
7          for(String element : aliste){
8              System.out.println(element);
9          }
10     }
11 }
```

Peter

Weiteres Beispiel

```
1  import java.util.ArrayList;
2  public class HalloWelt {
3      public static void main(String[] args) {
4          ArrayList<String> aliste = new ArrayList<String>();
5          aliste.add("Hans");
6          aliste.add("Peter");
7          for(String element : aliste){
8              System.out.println(element);
9          }
10     }
11 }
```

Peter

Java lernen - Typecasting

HPI-Team

Hasso Plattner Institut
IT Systems Engineering | Universität Potsdam

Rückblick

Verschiedene Datentypen

Unterscheidung in

- Primitive (int, double, ...)
- Objektdatentypen (String, ArrayList, Papagei,...)

Jedes Element (Attribut/ Parameter) besitzt festen Datentyp

Unter bestimmten Umständen kann/muss der Datentyp temporär der Situation angepasst werden

➔ Typecasting

Implizites Typecasting

Typecasting erfolgt automatisch falls daraus kein Datenverlust resultiert (z.B.: int zu double):

- Zuweisung einer Variable:

```
int x = 1;
double y = x;
```

- Aufrufen einer Funktion:

```
1  public class MeineKlasse {
2      public void machEtwas(double x) {
3          /* mach etwas */
4      }
5
6      public static void main(String[] args) {
7          MeineKlasse mk = new MeineKlasse();
8          mk.machEtwas(7);
9      }
10 }
```

Explizites Typecasting

Falls Datenverlust droht kann man Typecasting erzwingen

```
double y  = 1.0;
int x = y;
```

Fehler

- In runden Klammern: Typ der angenommen werden soll

```
int x = (int) y;
```

- Ein weiterer Anwendungsfall für explizites Typecasting

```
double x = 1 / 2.0;

int x = 1;
int y = 2;
double z = x / y;
double z = x / (double) y;
```

0.0

0.5

Einschränkungen

Kompatibilität

- Zahlentypen (z.B. int, double) untereinander kompatibel
- Zahl (int) und Zeichenkette (String) untereinander nicht kompatibel

```
int x = (int)"1";
String zahl = (String)1234;
```

Fehler

Fehler

- Primitive (int, double) & Objektdatentypen nie kompatibel

String → Zahl

Zur Umwandlung von Strings in Zahlen können Methoden der Wrapper Klassen benutzt werden

- parseDouble/parseInt/...: String → double/int/...

```
double x = Double.parseDouble("1.23");
int y = Integer.parseInt("123");
boolean b = Boolean.parseBoolean("true");
```

1.23

123

true

- Vorsicht: die Zeichenkette muss zum Zieldatentyp passen

```
double x = Double.parseDouble(“Hallo Welt”);
```

Fehler

NumberFormatException

Typecasten von Objektdatentypen

- Jedes Objekt kann jeden Datentyp innerhalb seiner Verbungshierarchie annehmen
- Alle Objektdatentypen → Object

```
Object einString = new String("Hallo");
char c = einString.charAt(1);
char c = ((String) einString).charAt(1);
System.out.println(c);
```

Fehler

a

```
Object einPapagei = new Papagei();
String p = (String) einPapagei;
```

Fehler

ClassCastException

Typecasten von Objektdatentypen

- Jedes Objekt kann jeden Datentyp innerhalb seiner Verbungshierarchie annehmen
- Alle Objektdatentypen → Object

```
Object einString = new String("Hallo");
char c = einString.charAt(1);
char c = ((String) einString).charAt(1);
System.out.println(c);
```

Fehler

a

```
Object einPapagei = new Papagei();
String p = (String) einPapagei;
```

Fehler

ClassCastException