

ISW: Software Engineering WS 2023/24

Java: Grundlagen und Beispiele Teil 2

Michael Anders

Institute of Computer Science
Chair of Software Engineering
Im Neuenheimer Feld 205
69120 Heidelberg, Germany

<https://se.ifi.uni-heidelberg.de>



Hinweis Teilnahme an Vorlesung

- Moodle-Umfrage zur Teilnahme bis **Dienstag 24.10. 12 Uhr** auszufüllen
- <https://moodle.uni-heidelberg.de/mod/feedback/view.php?id=965445>
- Nur, wer die Umfrage ausgefüllt hat, kann weiter an der Übung teilnehmen und Vorlesungsmaterial abrufen
 - Accounts finden Sie in Moodle unter “Zugangsdaten....”
- Wer die Umfrage nicht ausgefüllt hat:
- Bitte **direkt** nach der Vorlesung bei mir vorne melden
- Ansonsten bitte aus Moodle abmelden, wenn nicht an Übung teilnehmen

- Es gibt eine 5. Übungsgruppe
- Freitags 11:15 im CIP-Pool 3. Stock
- Bitte, wenn ihr Zeit habt umschreiben, damit die Gruppen nicht alle so voll sind

- Mittwoch 1. November ist Feiertag
- Übungen, die an diesem Tag stattfinden, werden in Absprache mit TutorInnen nachgeholt
- Vorlesungen Dienstag finden ganz normal statt



<https://de.wikipedia.org/wiki/Allerheiligen>

■ Teil 1: Überblick

- Erstkontakt
- Java Standard Edition
- Objektorientierte Programmierung (OOP)
- Unterschiede zwischen C++ und Java

■ Teil 2: Java Basics

- Klassen
- Felder (Arrays)
- Methoden
- Konstruktoren
- Vererbung
- Objekte
- Ausnahmen
- Collections, Generizität
- Callbacks, Lambda, Methoden-Referenzen, ...

Cutting corners to meet arbitrary management deadlines



Essential

Copying and Pasting
from

O'REILLY®

The Practical Developer
@ThePracticalDev

365



Inside a method that returns `IEnumerable<T>`, `yield return` has to return `T`, not an `IEnumerable<T>`.

Replace

```
yield return c.GetDeepControlsByType<T>();
```

with:

```
foreach (var x in c.GetDeepControlsByType<T>())
{
    yield return x;
}
```

[share](#) [improve this answer](#)

answered Jan 13 '10 at 10:29



Marcin Seredynski
4,719 ● 2 ● 14 ● 22

389 I just had exactly the same problem and googled this solution. I used it. I wanted to thank the author and got "You can't vote for your own post". I laughed when I realized what has just happened. Too much work kills your brain cells... – [Marcin Seredynski](#) Sep 12 '11 at 19:24

'14 at 12:35

6 [@KMX](#) - his vOcation is obviously 'Software Developer'! – [FastAI](#) Oct 6 '14 at 13:43

4 [@MarcinSeredynski](#) - This is without a doubt the best comment I've ever seen and is a poignant reminder of the nature of software development. – [bubbleking](#) Dec 29 '15 at 20:33

[show 4 more comments](#)

- Erben von einer Schnittstelle
 - Eine Schnittstelle kann von einer anderen Schnittstelle erben
- Wieviele Methoden muss eine Klasse implementieren, die die Schnittstellenklasse Yourself implementiert?

```
public interface Person {  
    public String sayHello();  
  
    Integer tellAge() throws Exception;  
}  
  
public interface Yourself extends Person {  
    public boolean isAwake();  
}
```

- Erben von einer Klasse
 - Klassen können vererben
 - Attribute
 - Konstruktoren
 - Operationen

```
public class ParentClass {  
  
    public String text = "ParentClass";  
  
    public ParentClass(String s) {  
        System.out.println("ParentClass");  
    }  
  
    public String saySomething() {  
        System.out.println("Hallo");  
        return ""; // macht keinen Sinn  
    }  
}
```

- Fortsetzung:

```
public class ChildClass extends ParentClass {  
  
    public ChildClass() {  
        System.out.println("ChildClass");  
  
        // Warum wird dies NICHT kompilieren?  
        // Was muss ergänzt werden?  
    }  
  
    // Ist das hier erlaubt?  
    public String saySomething() {  
        System.out.println("Hallo erstmal");  
        return " ";  
    }  
}
```


- Schlüsselwörter this und super
 - Beides referenziert Objekte
 - this(...) und super(...) zum Aufrufen von Konstruktoren
 - this.<name> und super.<name> zum Aufrufen von Methoden und zum Referenzieren von Attributen

```
public class ParentClass {  
    public String text = "ParentClass";  
  
    public ParentClass(String s) {  
        System.out.println("ParentClass");  
    }  
  
    public String saySomething() {  
        System.out.println("Hallo");  
        return ""; // macht keinen Sinn  
    }  
}
```

```
public class ChildClass extends ParentClass {  
    public ChildClass(String s) {  
        super(""); // erlaubt?  
  
        System.out.println("ChildClass");  
  
        super.saySomething();  
    }  
  
    public static void main(String[] args) {  
        // new ChildClass();  
        // warum kompiliert dies nicht?  
  
        new ChildClass("");  
        // Ausgabe?  
    }  
  
} // end of class
```

- Alle Klassen erben automatisch von java.lang.Object
- Fragen
 - [Woher kennt man die Operationen der Klasse Object?](#)
 - Welche muss man kennen?
- Objekte kopieren (flach)
 - In java.lang.Object ist Operation protected Object clone()

```
Point p1 = new Point(100, 200);  
Point p2 = p1.clone();
```

```
public class HiWorld {  
  
    public void greetings() {  
        System.out.println(toString());  
    }  
  
    public String toString() {  
        return("Hi world!");  
    }  
  
    public static void main(String[] args) {  
        HiWorld w = new HiWorld();  
        w.greetings();  
    }  
}
```

■ Objekte erzeugen

- geschieht immer unter Verwendung des Schlüsselwortes `new` und eines der Konstruktoren
- Java allokiert Speicherplatz für das Objekt

■ Konstruktoren

- überschreiben und überladen möglich

■ Beispiel

- Was wurde vorausgesetzt, damit die dritte Zeile kompiliert?
- Welches sind die Objekte?

```
new Person();
```

```
Person p1 = new Person();
```

```
Person p2 = new Person("Hans", "Meier");
```

```
p1 = p2;
```

- Aufruf von Attributen und Operationen an Objekten

- dot-Operator .

```
Person p = new Person("Hans", "Meier");  
  
String s1 = p.mNachname;  
  
p.mNachname = "Meier";  
  
p.sayHello();  
  
String vornameUndNachname = p.toString();  
  
String s3 = p.getClass().getName();  
// p.getClass() liefert ein Objekt vom Typ  
Class
```

- Aufruf von Attributen und Operationen an Klassen

- Wie müssen diese nochmal deklariert sein?

```
int i = Person.AnzahlArme; // 2  
  
Person.sayHello();  
// alle Personen können das
```

- Objekte als Parameter von Methoden
 - Gibt man den Wert eines primitiven Datentyps an eine Methode, dann spricht man von „**pass by value**“ → **Kopie des Wertes**
 - Gibt man ein Objekt an eine Methode, dann erhält diese keine Kopie des Objekts, sondern IMMER eine KOPIE der **Referenz** → **eine Art „pass by reference“**

```
// Ausgabe "hello" oder "bleibt so"?  
// Vanessa oder Denise?  
public class PassByReference {  
  
    public static void main(String[] args) {  
        String str = "bleibt so";  
  
        Person p = new Person(„Vanessa“);  
  
        PassByReference pbr = new PassByReference();  
        pbr.sayHello(str, p);  
        System.out.println("str=" + str);  
        System.out.println("p=" + p.getName());  
    }  
  
    public void sayHello(String s, Person p) {  
        s = "hello";  
        p.setName(„Denise“);  
    }  
  
} // end of class
```

■ Objekte löschen

- Das Löschen übernimmt **Java's Garbage Collector**, wenn keine Referenzen mehr auf das Objekt vorhanden sind
- Man löscht eine Referenz explizit mit Schlüsselwort `null`
- Operation `finalize()` in `Object` zum Aufräumen

■ Beispiel

```
Person p = new Person();  
Person p2 = p;  
p = null;
```

- Sichtbarkeit von Attributen und Operationen
 - public Operationen und Attribute:
 - Operationen und Attribute sind von allen anderen Klassen verwendbar
 - default Operationen und Attribute:
 - Operationen und Attribute sind nur von anderen Klassen in demselben package verwendbar, sonst compile error
 - protected Operationen und Attribute:
 - Operationen und Attribute sind nur von VERERBTEN Klassen in demselben package verwendbar, sonst compile error
 - private Operationen und Attribute:
 - Operationen und Attribute sind nur von der eigenen Klasse verwendbar

- Typkonvertierung von Klassen (type casting)
 - zu konvertierende Klasse muss von Ziel-Klasse erben, aber die Attribute und Operationen gehen “verloren”
- Beispiel

```
public class Apple {  
    public void fallDown() {}  
}  
  
public class GreenApple extends Apple {  
    public String color;  
    public void becomeRed() {  
    }  
}
```

```
GreenApple ga = new GreenApple();  
Apple a1 = (Apple) ga; // explicit cast  
Apple a2 = ga; // implicit cast is  
// possible because GreenApple extends Apple
```

```
Apple a3 = new GreenApple();
```

```
ga.becomeRed();  
//a1.becomeRed();  
// compile error, method not known  
//a2.becomeRed();  
// compile error, method not known  
//a3.becomeRed();  
// compile error, method not known
```

```
ga.color = new String("");  
//a1.color = "";  
// compile error  
//a2.color = "";  
// compile error  
//a3.color = "";  
// compile error
```

```
ga = new Apple(); // compile error  
ga = (GreenApple) new Apple();  
// runtime error ClassCastException
```


■ Ausnahmen = Exceptions

- Ausnahmen sind Subklassen der Klasse `java.lang.Exception`
- Von vielen Operationen der Java API werden Ausnahmen geworfen
- Zusätzlich gibt es Ausnahmen zur Laufzeit. (z.B. bei der Division durch 0)
- Diese Ausnahmen muss man entweder auffangen (`try/catch`) oder weiter werfen (`throws`)
- Im Rahmen der eigenen Anwendung definiert und verwendet man eigene Ausnahmen zwecks Robustheit von Operationen
- Häufig verwendetes Konzept

■ Ausnahmen in einer eigenen Operation werfen => Design-Frage

```
public class ClassWithException {  
    public void anExceptionalMethod()  
        throws Exception {  
        if (someThingUnusualHasHappened())  
        {  
            throw new Exception(„something wrong“);  
            // execution never reaches here  
        }  
    }  
} // end of class
```

■ Ausnahmen auffangen und werfen

```
public void responsibleMethod() throws Exception {  
    ClassWithException cwe = new ClassWithException();  
    try {  
        cwe.anExceptionalMethod();  
    } catch (Exception e) {  
        // do something responsible  
        // re-throw the exception  
        throw e;  
        // or another exception!  
    }  
}
```

■ Collection Framework

- Bibliothek, um eine Gruppe von Objekten zu verwalten
- Paket `java.util`

■ Gemeinsamkeiten

- Einfügen von Elementen
- Entnehmen von Elementen
- Herausfinden, ob ein Element in der Collection ist
- Zugriff auf ein Element in der Collection
- Durch die Collection iterieren

■ Interface Set

- Eindeutige Liste von Objekten
 - Mengencharakter
 - Keine Duplikate
 - → Klasse `HashSet`

■ Interface List

- Liste von Objekten
 - Über `int`-Werte indiziert
 - Duplikate erlaubt
 - Einfügen in der Mitte
 - → Klasse `ArrayList`

■ Interface Map (keine Collection)

- Objekte mit eindeutiger ID
 - Zuordnung von Schlüsseln zu Werten
 - Keine Duplikate bei Schlüsseln
 - Duplikate bei Werten erlaubt
 - → Klasse `HashMap`

- Arten von Collections:
sorted und unsorted

- Beispiel ArrayList

```
ArrayList mixed = new ArrayList();

mixed.add("Foo");
mixed.add(new Complex(2, 3));
mixed.add(new int[] {1, 2, 4, 19});
mixed.add(new ArrayList());

for (int i = 0; i < mixed.size()) {
    Object o = mixed.get(i);

    System.out.println(o.hashCode());
}
```

generische Parameter

```
List<Pokemon> pokemons = new ArrayList<>();
```

```
Map<String, Long> phonebook = new HashMap<>();

phonebook.put("Lara", new Long(1225100));
phonebook.put("Schiedermeier", new Long(22222));
phonebook.put("Sekretariat", new Long(111111));

Long larasNumber = (Long) phonebook.get("Lara");
```

- Beispiel HashMap

- <https://confluence-se.ifi.uni-heidelberg.de/x/CwCmIQ>

- <https://java.oracle.com>
- <https://openjdk.java.net>
- <https://docs.oracle.com/en/java/javase/17/>
- <https://docs.oracle.com/javase/tutorial/java/index.html>
- Hanspeter Mössenböck, *Sprechen Sie Java?*, dpunkt.verlag, 2014
- Ben Evans, David Flanagan, *Java in a Nutshell*, O'Reilly, 2018
- Christian Ullenboom, *Java ist auch eine Insel*, Galileo Computing, 15. Auflage, 2020, <http://www.tutego.de/javabuch/>
- Guido Krüger, *Handbuch der Java-Programmierung*, Addison-Wesley, 2009, <http://www.javabuch.de>
- Joshua Bloch, *Effective Java, 3rd Edition*, Addison-Wesley, 2017
- Podcast: Chaosradio Express, *CRE090: Java – Ein Überblick über die Java-Plattform*, <https://cre.fm/cre090-java>

Michael Anders

Institute of Computer Science
Chair of Software Engineering
Im Neuenheimer Feld 205
69120 Heidelberg, Germany
<https://se.ifi.uni-heidelberg.de>



RUPRECHT-KARLS-UNIVERSITÄT HEIDELBERG
