

LU08a - Spezielle Methoden für Klassen

toString() - Methode

Sie haben sicherlich bereits einmal probiert, einen eigenen Datentyp oder eine eigene Klasse direkt mit `System.out.println()` auszugeben.

Das Resultat (hier am Beispiel der Klasse Boot) auf der Konsole sah bestimmt irgendwie in dieser Art aus:

```
Boot@4f023edb
```

Das liegt daran, das `System.out.println()` die Methode `toString()` der Klasse Boot aufruft, ist diese nicht implementiert, so wird die Methode in der Vaterklasse hier der Klasse `Object` aufgerufen. In der Klasse `Object` ist `toString()` wie folgt implementiert:

```
public String toString() {
    return getClass().getName() + "@" + Integer.toHexString(hashCode());
}
```

Es wird also der `hashcode()` des Boot-Objektes ausgegeben. Möchten wir diese Ausgabe nun anpassen, so müssen wir lediglich die Methode `toString()` in unserer Klasse importieren.

Hier am Beispiel der Klasse Boot dargestellt:

```
/**
 * Represents a simple Boat-Datatype with toString Method
 *
 * @author Kevin Maurizi
 * @since 02.12.2042
 */
public class Boot {
    String marke;
    double laenge;
    double kaufpreis;

    @Override
    public String toString() {
        return "Boot{" +
            "Marke='" + marke + '\'' +
            ", Laenge=" + laenge +
            ", Kaufpreis=" + kaufpreis +
            '}';
    }
}
```

```
import java.util.Scanner;

/**
 * Bootsliste um eine Liste von Booten zu verwalten
 *
 * @author Kevin Maurizi
 * @since 2042.12.02
 * @version 0.1
 */
public class Bootsliste {

    static Scanner scanner;

    public Bootsliste(){
        scanner = new Scanner(System.in);
    }

    public static void main(String[] args) {
        Bootsliste bootsliste = new Bootsliste();
        bootsliste.run();
    }

    private void run() {
        Boot boot = new Boot();
        System.out.println(boot); // Direkte Ausgabe des
        boot via sout
    }
}
```



Das `@Override` oberhalb der Methode ist eine sogenannte Annotation, sie hat in diesem Fall keinen Einfluss auf den Code, sondern zwingt den Compiler zu einer Fehlermeldung wenn die ausgewählte Methode nicht die Methode der Oberklasse überschreibt. In unserem Fall ist in der Oberklasse `Object` die Methode `toString()` vorhanden und es kommt zu keiner Fehlermeldung.

Ist die `toString()` Methode im `Boot` nun implementiert, kann das `Boot` direkt via `System.out.println(boot1)` ausgedruckt werden.

compareTo() - Methode

Sie kennen bereits die Möglichkeit Listen mit `Collections.sort(<listename>)` zu sortieren. Allerdings können Sie diese Sortierung nur machen, wenn die Liste aus String oder den Wrapper-Klassen der primitiven Datentypen (Integer, Double, Float etc.) besteht.

Wir möchten nun aber auch unseren eigenen Liste `ArrayList<Boot> bootsliste = new ArrayList<>()` durch `Collections.sort(bootsliste)` sortieren können. Der Compiler und die JVM können aber nicht wissen, nach welchen Kriterien Sie das Boot sortieren sollen. Um die Boote zu sortieren müssen wir die Boote vergleichen können. Dazu müssen wir folgende Schritte machen:

1. Das Interface Comparable<T> implementieren
2. Die Methode compareTo() implementieren

1. Das Interface "Comparable<T>" implementieren

Auf Deutsch bedeutet Interface Schnittstelle. In der Programmiersprache Java ist das Interface eine Schnittstellendefinition für jede Klasse die dieses Interface implementiert. Das bedeutet, dass im jeweiligen Interface genau beschrieben ist, welche Schnittstellen (Methoden) eine Klasse hat, wenn es dieses jeweilige Interface implementiert.

Das Interface Comparable<T> beinhaltet lediglich eine Methodensignatur, in diesem Fall die Methode `compareTo(T o)`:

```
public interface Comparable<T> {  
    public int compareTo(T o);  
}
```

Jede Klasse, die nun dieses Interface implementiert, ist also dazu gezwungen die Methode `compareTo(T o)` zu implementieren. Um der Klasse ein Interface zuzuweisen, muss das Interface in der Klassensignatur hinzugefügt werden.

```
public class Boot implements Comparable<Boot>{}
```

In diesem Fall, möchten wir Objekte vom Typ Boot vergleichbar machen und zwar mit anderen Objekten vom Typ Boot. Dazu ergänzen wir die Klasse Boot mit `implements Comparable<Boot>` in die <...> gehört hier die Klasse mit welcher diese Klasse verglichen werden soll (In der Regel identisch mit der Klasse in welcher das Interface ergänzt wird).

 In der Definition des Interface Comparable<T> sehen Sie die spitzen Klammern <...> mit dem grossen T. Das ist ein sogenanntes Generic, Generics sind Stellvertreter im Code. In dem wir in der Klassendefinition dem Generic T den Datentyp Boot zuweisen, wird überall im Code der Interfacedefinition das T durch Boot ersetzt. Dadurch wird



aus der Methode `compareTo(T o)` die Methode `compareTo(Boot o)`.

2. Die Methode "compareTo()" implementieren

Nachdem Sie die Klassensignatur `public class Boot implements Comparable<Boot>{}` angepasst haben, wird sich Ihre IDE mit einer Fehlermeldung revanchieren. Die Fehlermeldung sagt aus, dass Sie ein Interface implementiert haben, aber die enthaltene Methode noch nicht implementiert haben. Eine einfach Lösung ist ein Autoimport der Methode mit Alt+Entr auf der Klasse und dann Implements methods.

The screenshot shows a Java code editor with the following code:

```
* @author Kevin Maurizi
* @since 2042.01.01
* @version 0.1

public class Boot implements Comparable<Boot>{
    private String marke;
    private String farbe;
    private double kaufpreis;
}
```

A code completion menu is open over the line `implements Comparable<Boot>`. The first item in the menu is `Implement methods`, which is highlighted in blue. Other items include `Make 'Boot' abstract`, `Safe delete 'Boot'`, `Create Test`, `Create subclass`, `Make 'Boot' package-private`, and `Rollback changes in current line`. A tooltip at the bottom of the menu says `Press Strg+Umschalt+I to open preview`.

Die IDE wird Ihnen dann die leere Methode in der Klasse ergänzen.

```
@Override
public int compareTo(Boot o) {
    return 0
}
```

An Ihnen liegt es nun, diese mit der Logik zu ergänzen.

Wie Sie der Methodensignatur entnehmen können, muss die Methode eine

3. Endresultat

```
public class Boot implements Comparable<Boot>{
    private String marke;
    private String farbe;
    private double kaufpreis;
```

```

    private String seriennummer;
    private boolean imWasser;

    @Override
    public int compareTo(Boot o) {
        // Sortieren nach Kaufpreis
        if(kaufpreis < o.getKaufpreis()){
            return -1;
        }else if(kaufpreis == o.getKaufpreis()){
            return 0;
        }else{
            return 1;
        }
    }
}

```

getWertOhneAttribut() - Methode

In manchen Klassen macht es Sinn einen Getter für einen berechneten Wert zu ergänzen, anstatt ein Attribut für diesen Wert zu haben. Betrachten wir das Beispiel der Klasse Person, eine Person hat ein geburtsdatum, das Alter der Person ist aber jeden Tag ein anderes. Es macht also keinen Sinn, das Attribut `alter` in der Klasse zu speichern, denn der Wert des Attributes ist spätestens nach 24h wieder veraltet.

```

/**
 * Ein einfaches Objekt für Person
 *
 * @author Kevin Maurizi
 * @since 02.12.2042
 */
public class Person{
    private String vorname;
    private String nachname;
    private LocalDate geburtsdatum;
}

```

Trotzdem kann es ganz praktisch sein, das Alter einer Person mit einer Methode abfragen zu können. Wir ergänzen den Code also um die Methode `getAlter()` und berechnen in der Methode das Alter der Person und geben dieses zurück.

```

/**
 * Ein einfaches Objekt für Person
 *

```

```
* @author Kevin Maurizi
* @since 02.12.2042
*/
public class Person{
    private String vorname;
    private String nachname;
    private LocalDate geburtsdatum;

    public Period getAlter(){
        LocalDate jetzt = LocalDate.now();
        Period alter = geburtsdatum.until(jetzt);
        return alter;
    }
}
```



Für das Speichern von Datumswerten ist der Datentyp `LocalDate` und für das Speichern von Zeitperioden der Datentyp `Period` aus der Bibliothek `java.time` hervorragend geeignet. Mehr dazu finden Sie im Kapitel [LU08b - Java und die Zeit](#)



© Kevin Maurizi

From:

<https://wiki.bzz.ch/> - BZZ - Modulwiki

Permanent link:

<https://wiki.bzz.ch/modul/archiv/m319/learningunits/lu08/lu08a-spezialmethoden>

Last update: **2025/06/23 07:45**

