Java

Vererbung

Marcus Köhler

9. November 2018

Java-Kurs

Overview

1. Datenkapselung

2. Mehr über Arrays
   - Mehrdimensionale Arrays

3. Vererbung
   - Vererbung
   - Konstruktor
   - Implizite Vererbung
   - Praktisches Beispiel

Datenkapselung

Zugriffsrechte

Um zu regeln, welche Objekte bzw. Klassen in Java worauf zugreifen dürfen, gibt es die folgenden Keywords:

- `public`

Zugriffsrechte

Um zu regeln, welche Objekte bzw. Klassen in Java worauf zugreifen dürfen, gibt es die folgenden Keywords:

- `public`
- `private`

Zugriffsrechte

Um zu regeln, welche Objekte bzw. Klassen in Java worauf zugreifen dürfen, gibt es die folgenden Keywords:

- `public`
- `private`
- `protected`

Zugriffsrechte

```
1  public class Student {
2      public String getName() {
3          return "Peter";
4      }
5
6      private String getFavouritePorn() {
7          return "...";
8      }
9  }
10
11 // [...]
12 exampleStudent.getName(); // Works!
13 exampleStudent.getFavouritePorn(); // Error
```

Datenkapselung

Datenkapselung bezeichnet das „Verstecken“ der Attribute von Objekten.

```java
public class Student{
    private String name;
    public int matNr;

    [...]
    public void setName(String name) {
        if(name.isEmpty() || name == null) return;
        this.name = name;
    }
}
[...]
    Student s = new Student("Max", 4567283);
    s.name = ""; // throws exception
    s.setName(""); //doesn't change name
    s.matNr = -1; // works, but makes no sense
    s.setName("Peter"); //works
[...]
```

Mehr über Arrays

2-Dimensional Array

Arrays können mehrdimensional sein.

Ein n-dimensionales Array hat dementsprechend auch n Indizes.

```
1 public static void main(String[] args) {
2
3     // an array with 100 elements
4     int[][] intArray = new int[10][10];
5
6     intArray[0][0] = 0;
7     intArray[0][9] = 9;
8     intArray[9][9] = 99;
9 }
```

Arrays von Objekten

Es ist natürlich auch möglich, ein Array von Objekten anzulegen:

```
1  public static void main(String[] args) {
2
3      Student[][] studentArray = new Student[10][10];
4
5      for(int i = 0; i < 10; i++) {
6          for(int j = 0; j < 10; j++) {
7              studentArray[i][j] = new Student();
8          }
9      }
10 }
```

Vererbung

Eine Lieferung

Die folgende Klasse `Letter` ist eine Unterklasse von `Delivery`, erkennbar am Keyword `extends`.

- `Letter` ist eine *Unterklasse* bzw *Subklasse* der Klasse `Delivery`
- `Delivery` ist die *Superklasse* von `Letter`

```
1 public class Letter extends Delivery {
2
3 }
```

Eine Lieferung

Die folgende Klasse `Letter` ist eine Unterklasse von `Delivery`, erkennbar am Keyword `extends`.

- `Letter` ist eine *Unterklasse* bzw *Subklasse* der Klasse `Delivery`
- `Delivery` ist die *Superklasse* von `Letter`

```
1 public class Letter extends Delivery {
2
3 }
```

Eine Klasse kann mehrere Subklassen haben.
Umgekehrt kann eine Klasse maximal eine Superklasse haben.
Eine Subklasse *erbt* von ihrer Superklasse.

Beispiel

Für heute haben wir die Klassen `PostOffice`, `Delivery` and `Letter`. Im Verlauf des Kurses werden sie immer weiter wachsen.

```
1   public class Delivery {
2       private String address;
3       private String sender;
4
5       public void setAddress(String addr) {
6           address = addr;
7       }
8
9       public void setSender(String snd) {
10          sender = snd;
11      }
12
13      public void printAddress() {
14          System.out.println(this.address);
15      }
16  }
17
```

Methodenvererbung

Mithilfe von Vererbung kann man auf einer Instanz von `Letter` die Methoden von `Delivery` aufrufen:

```java
public class PostOffice {

    public static void main(String[] args) {

        Letter letter = new Letter();

        letter.setAddress("cafe ascii, Dresden");

        letter.printAddress();
        // prints: cafe ascii, Dresden
    }
}
```

Methoden-Override

Im folgenden Code-Snippet wird die Methode printAdress() in der Klasse `Letter` neu definiert:

```
1 public class Letter extends Delivery {
2     @Override
3     public void printAddress() {
4         System.out.println("a letter for " + this.address);
5     }
6 }
```

Methoden-Override

Im folgenden Code-Snippet wird die Methode printAdress() in der Klasse `Letter` neu definiert:

```
1 public class Letter extends Delivery {
2     @Override
3     public void printAddress() {
4         System.out.println("a letter for " + this.address);
5     }
6 }
```

`@Override` ist eine *Annotation*. Annotationen sind nicht zwingend notwendig, machen den Code allerdings leichter verständlich. Wofür Annotationen sonst noch gebraucht werden, besprechen wir in kommenden Kursen.

Methoden-Override

Durch den Override wird jetzt die `printAddress()`-Methode aus der Klasse `Letter` verwendet.

```
1  public class PostOffice {
2      public static void main(String[] args) {
3          Letter letter = new Letter();
4
5          letter.setAddress("cafe ascii, Dresden");
6
7          letter.printAddress();
8          // prints: a letter for cafe ascii, Dresden
9      }
10 }
```

Super()

Falls in der Superklasse(hier `Delivery`) einen **Konstruktor mit Parametern** definieren,

müssen wir das auch in jeder Unterklasse tun.

```
1  public class Delivery {
2      private String address;
3      private String sender;
4
5      public Delivery(String address, String sender) {
6          this.address = address;
7          this.sender = sender;
8      }
9
10     public void printAddress() {
11         System.out.println(address);
12     }
13 }
```

super()

Innerhalb des Konstruktors von `Letter` können wir mittels des Aufrufs `super()` den Konstruktor der Superklasse aufrufen.

```
1  public class Letter extends Delivery {
2      public Letter(String address, String sender) {
3          super(address, sender);
4      }
5
6      @Override
7      public void printAddress() {
8          System.out.println("a letter for " + this.address);
9      }
10 }
```

Super() - Test

```java
public class PostOffice {
    public static void main(String[] args) {
        Letter letter = new Letter("cafe ascii, Dresden", "");

        letter.printAddress();
        // prints: a letter for cafe ascii, Dresden
    }
}
```

java.lang.Object

Jede Klasse in Java erbt automatisch von `java.lang.Object`.
Deshalb erbt jede Klasse die Methoden von `Object`.

Alle Methoden von `Object` sind unter

`http://docs.oracle.com/javase/10/docs/api/java/lang/Object.html`

zu finden.

toString()

`Letter`ist eine Unterklasse von `Object`. Deshalb erbt `Letter` die Methode `toString()` von `Object`.
`System.out.println(argument)` ruft `argument.toString()` auf, um einen String zu erhalten, der ausgegeben werden kann.

```
1  public class PostOffice {
2      public static void main(string[] args) {
3          letter letter =
4              new letter("cafe ascii, dresden", "");
5
6          system.out.println(letter);
7          // prints: letter@_some_hex-value_
8          // for example: letter@4536ad4d
9      }
10 }
```

toString()-Override

```
1  public class Letter extends Delivery {
2      public letter(string address, string sender) {
3          super(address, sender);
4      }
5
6      @Override
7      public string tostring() {
8          return "a letter for " + this.address;
9      }
10 }
```

toString()-Override - Test

```
1 public class PostOffice {
2     public static void main(String[] args) {
3         Letter letter =
4             new Letter("cafe ascii, Dresden", "");
5
6         System.out.println(letter);
7         // a letter for cafe ascii, Dresden
8     }
9 }
```

Beispiel

-> IntelliJ