

Java

Visibility and inheritance

Vincent Gerber, Tilman Hinnerichs

Java Kurs

30. Mai 2018



Overview

1. One more OOP exercise!
2. Visibilities
3. Inheritance
 - Inheritance
 - Constructor
 - Implicit Inheritance

Repetition

<https://github.com/THinnerichs/Java-Course>

<https://github.com/THinnerichs/Java-Course/tree/master/Exercises/vehicle>

Comment and explain the code

Visibilities

- public
- private
- protected

Visibilities

```
1  public class Student {
2      public String getName() {
3          return "Peter";
4      }
5
6
7      private String getFavouriteFilm() {
8          return "...";
9      }
10     }
11
12     // [...]
13     exampleStudent.getName(); // Works!
14     exampleStudent.getFavouriteFilm(); // Error
15
16
```

A special Delivery

Our class *Letter* is a kind of *Delivery* denoted by the keyword **extends**.

- *Letter* is a **subclass** of the class *Delivery*
- *Delivery* is the **superclass** of the class *Letter*

```
1 public class Letter extends Delivery {  
2  
3 }  
4
```

As mentioned implicitly above a class can has multiple subclasses. But a class can only inherit directly from one superclass.

Hier könnten Sie Ihr UML-Diagramm für das Beispiel zeigen!

Example

We have the classes: *PostOffice*, *Delivery* and *Letter*. They will be used for every example in this section and they will grow over time.

```
1  public class Delivery {  
2  
3      protected String address;  
4      protected String sender;  
5  
6      public void setAddress(String addr) {  
7          address = addr;  
8      }  
9  
10     public void setSender(String snd) {  
11         sender = snd;  
12     }  
13  
14     public void printAddress() {  
15         System.out.println(this.address);  
16     }  
17 }  
18
```


Inherited Methods

The class *Letter* also inherits all methods from the superclass *Delivery*.

```
1 public class PostOffice {  
2  
3     public static void main(String[] args) {  
4  
5         Letter letter = new Letter();  
6  
7         letter.setAddress("cafe ascii, Dresden");  
8  
9         letter.printAddress();  
10        // prints: cafe ascii, Dresden  
11    }  
12 }  
13
```

Override Methods

The method `printAddress()` is now additionally defined in *Letter*.

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

`@Override` is an annotation. It helps the programmer to identify overwritten methods. It is not necessary for running the code but improves readability. What annotations else can do we discuss in a future lesson.

Override Methods

Now the method `printAddress()` defined in *Letter* will be used instead of the method defined in the superclass *Delivery*.

```
1 public class PostOffice {  
2  
3     public static void main(String[] args) {  
4  
5         Letter letter = new Letter();  
6  
7         letter.setAddress("cafe ascii, Dresden");  
8  
9         letter.printAddress();  
10        // prints: a letter for cafe ascii, Dresden  
11    }  
12 }  
13
```

Super()

If we define a **constructor with arguments** in *Delivery* we have to define a constructor with the same list of arguments in every subclass.

```
1  public class Delivery {  
2  
3      protected String address;  
4      protected String sender;  
5  
6      public Delivery(String address, String sender) {  
7          this.address = address;  
8          this.sender = sender;  
9      }  
10  
11     public void printAddress() {  
12         System.out.println(address);  
13     }  
14 }  
15
```

Super()

For the constructor in the subclass *Letter* we can use `super()` to call the constructor from the superclass.

```
1 public class Letter extends Delivery {  
2  
3 public Letter(String address, String sender) {  
4     super(address, sender);  
5 }  
6  
7 @Override  
8 public void printAddress() {  
9     System.out.println("a letter for " + this.address);  
10 }  
11 }  
12
```

Super() - Test

```
1 public class PostOffice {  
2  
3     public static void main(String[] args) {  
4         Letter letter =  
5         new Letter("cafe ascii, Dresden", "");  
6  
7         letter.printAddress();  
8         // prints: a letter for cafe ascii, Dresden  
9     }  
10 }  
11
```

Object

Every class is a subclass from the class *Object*. Therefore every class inherits methods from *Object*.

See <http://docs.oracle.com/javase/7/docs/api/java/lang/Object.html> for a full reference of the class *Object*.

toString()

Letter is a subclass of *Object*. Therefore *Letter* inherits the method `toString()` from *Object*.

`System.out.println(argument)` will call `argument.toString()` to receive a printable String.

```
1 public class PostOffice {
2
3     public static void main(String[] args) {
4         Letter letter =
5         new Letter("cafe ascii, Dresden", "");
6
7         System.out.println(letter);
8         // prints: Letter@_some_HEX-value_
9         // for example: Letter@4536ad4d
10    }
11 }
12
```


Override toString()

```
1 public class Letter extends Delivery {  
2  
3     public Letter(String address, String sender) {  
4         super(address, sender);  
5     }  
6  
7     @Override  
8     public String toString() {  
9         return "a letter for " + this.address;  
10    }  
11 }  
12
```

Override toString() - Test

```
1 public class PostOffice {  
2  
3     public static void main(String[] args) {  
4         Letter letter =  
5         new Letter("cafe ascii, Dresden", "");  
6  
7         System.out.println(letter);  
8         // a letter for cafe ascii, Dresden  
9     }  
10 }  
11
```

Extending our vehicle class

Extend our current model with some inheritance! e.g

- Bicycle
- Bus
- Truck
- Extend methods and attributes
- ...