Java

Inheritance

Max Langer, Leonard Follner, Alexander Hesse

6. November 2016

Java-Kurs

Overview

1. Last session
    - Visibilities
    - Arrays
    - Multi-Dimensional Array
    - Philosophic stuff
2. Inheritance
    - Inheritance
    - Constructor
    - Implicit Inheritance

Last session

Visibilities

- public
- private
- protected

Visibilities

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

Array

An array is a data-type that can hold a **fixed number** of elements. An Element can be any simple data-type or object.

```
1     public static void main(String[] args) {
2
3         int[] intArray = new int[10];
4         intArray[8] = 7; // assign 7 to the 9th element
5         intArray[9] = 8; // assign 8 to the last element
6
7         System.out.println(intArray[8]); // prints: 7
8     }
9
```

You can access every element via an index. A n-element array has indexes from 0 to (n-1).

Array Initialization

You can initialize an array with a set of elements.

```
1       public static void main(String[] args) {
2
3           int[] intArray = {3, 2, 7};
4
5           System.out.println(intArray[0]); // prints: 3
6           System.out.println(intArray[1]); // prints: 2
7           System.out.println(intArray[2]); // prints: 7
8       }
9
```

Alternative Declaration

There two possible positions for the square brackets.

```
1       public static void main(String[] args) {
2
3           // version 1
4           int[] intArray1 = new int[10];
5
6           // version 2
7           int intArray2[] = new int[10];
8       }
9
```

2-Dimensional Array

Arrays work with more than one dimension. An m-dimensional array has m indexes for one element.

```
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     }
10
```

Assignment with Loops

Loops are often used to assign elements in arrays.

```
1     public static void main(String[] args) {
2
3         int[][] intArray = new int[10][10];
4
5         for(int i = 0; i < 10; i++) {
6             for(int j = 0; j < 10; j++) {
7                 intArray[i][j] = i*10 + j;
8             }
9         }
10    }
11
```

Arrays with objects

Loops are often used to assign elements in arrays.

```
1     public static void main(String[] args) {
2
3         int[][] studentArray = new Student[10][10];
4
5         for(int i = 0; i < 10; i++) {
6             for(int j = 0; j < 10; j++) {
7                 intArray[i][j] = new Student();
8             }
9         }
10    }
11
```

Inheritance

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.

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       private String address;
4       private 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 additional definded 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 programer to identify overwritten methods. It is not neccessary 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         private String address;
4         private 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
```

Images stolen at:

- **BMW Series** Taken from bmwblog.com
- **CD-Player** Taken from wikimedia.com, Uploaded by Ukexpat