Object Oriented Programming

Class Student

```
1  public class Student {
2
3  // Attributes
4  private String name;
5  private int matriculationNumber;
6
7
8  // Methods
9  public void setName(String name) {
10 this.name = name;
11 }
12
13 public int getMatriculationNumber() {
14 return matriculationNumber;
15 }
16
17 }
18
```

Creation

We learned how to declare and assign a primitive datatype.

```
1 int a; // declare a
2 a = 273; // assign 273 to a
```

The creation of an object works similar.

```
1 Student example = new Student();
2 // create an instance of Student
```

The **object** derived from a **class** is also called **instance**. The variable is called the **reference**.

Calling a Method

```
1  public class Student {
2
3  private String name;
4
5  public String getName() {
6  return name;
7  }
8
9  public void setName(String newName) {
10 name = newName;
11 }
12
13 }
```

The class *Student* has two methods: *void printTimetable()* and *void printName()*.

Calling a Method

```java
public class Main {

public static void main(String[] args) {
Student example = new Student(); // creation
example.setName("Jane"); // method call
String name = example.getName();
System.out.println(name); // Prints "Jane"
}

}
```

You can call a method of an object after its creation with **reference.methodName();**.

Calling a Method

```
1  public class Student {
2
3  private String name;
4
5  public void setName(String newName) {
6  name = newName;
7  printName();   // Call own method
8  this.printName(); // Or this way
9  }
10
11 public void printName() {
12 System.out.println(name);
13 }
14
15 }
```

You can call a method of the own object by simply writing **methodName();** or **this.methodName();**

Methods with Arguments

```
1  public class Calc {
2
3  public void add(int summand1, int summand2) {
4  System.out.println(summand1 + summand2);
5  }
6
7  public static void main(String[] args) {
8  int summandA = 1;
9  int summandB = 2;
10 Calc calculator = new Calc();
11 System.out.print("1 + 2 = ");
12 calculator.add(summandA, summandB);
13 // prints: 3
14 }
15
16 }
```

Methods with Return Value

A method without a return value is indicated by **void**:

```
1 public void add(int summand1, int summand2) {
2 System.out.println(summand1 + summand2);
3 }
```

A method with an **int** as return value:

```
1 public int add(int summand1, int summand2) {
2 return summand1 + summand2;
3 }
```

Calling Methods with a return value

```java
public class Calc {

public int add(int summand1, int summand2) {
return summand1 + summand2;
}

public static void main(String[] args) {
Calc calculator = new Calc();
int sum = calculator.add(3, 8);
System.out.print("3 + 8 = " + sum);
// prints: 3 + 8 = 11
}

}
```

Constructors

```
1  public class Calc {
2
3  private int summand1;
4  private int summand2;
5
6  public Calc() {
7  summand1 = 0;
8  summand2 = 0;
9  }
10
11 }
```

A constructor gets called upon creation of the object

Constructors with Arguments

```
1  public class Calc {
2
3  private int summand1;
4  private int summand2;
5
6  public Calc(int x, int y) {
7  summand1 = x;
8  summand2 = y;
9  }
10
11 }
```

```
1 [...]
2 Calc myCalc = new Calc(7, 9);
```

A constructor can have arguments as well!

An Example

You want to program an enrollment system, for a programming course.

Your classes are:

student	who wants to attend the course
lesson	which is a part of the course
tutor	the guy with the bandshirt
room	where your lessons take place
...	

Class Student

```
1 public static void main(String[] args) {
2 Student peter = new Student();
3 peter.changeName("Peter");
4 }
```

Java

Introduction to OOP and inheritance

Vincent Gerber, Tilman Hinnerichs

Java Kurs

16. Mai 2018

Overview

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       Student[][] 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
```

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
```