Lucerne University of Applied Sciences and Arts

Programming and Algorithms

Personal Documentation

Ervin Mazlagić

Horw, autumn 2013

Contents

Preface

This is a personal documentation and notebook for the first course in programming at the Lucerne University of Applied Sciences and Arts. The goal of this document is to collect useful informations and nice snippets of code out of the course.

This document shall not be provided as a official or unofficial cheatsheet for the course exam or similar.

1 Objects and classes

1.1 Summary exercises

Exercise 1.31

What are the types of the following values?

0	short, char, byte, int, long
"hello"	String
101	short, char, byte, int, long
-1	int, char, byte, int, long
true	boolean
"33"	String
3.1415	float, double

Exercise 1.32

What would you have to do to add a new filed, for example one called name, to a circle object?

```
private String name;
```

Exercise 1.33

Write the signature of a method named send that has one parameter of type String, and does not return a value.

```
public void send(String foo)
```

Exercise 1.34

Write a signature for a method named average that has two parameters, both of type int, and returns an int value.

```
public int average(int foo, int bar)
```

Exercise 1.35

Look at the book you are reading right now. Is it an object or class? If it is a class, name some objects. If it is an object, name its class.

The book is definitely an object, because it's a specific thing and in no way generic. The class could have a name like `SchoolBook`, `CodingBook` or just `Book`.

Exercise 1.36

Can an object have several different classes? Discuss.

No it can't.

2 Understanding class definitions

2.1 Start with Eclipse

In the first chapter we've worked with the BlueJ IDE but now I want to check Java-Coding with a common and popular Java-IDE like Eclipse To get the BlueJ-Projects work with Eclipse there are some things that have to be done.

1. Create a new project in Eclipse.
2. Import the source (BlueJ example-code).
3. Add a package-name to the source.
4. Create a main (replaces all interaction which were invoked by hand).

Listing 1: TicketMachine

```
1
2 package foobar;
3
4 public class TicketMachine
5 {
6     // The price of a ticket from this machine.
7     private int price;
8     // The amount of money entered by a customer so far.
9     private int balance;
```

Listing 2: Main (TicketMachine)

```
1  package foobar;
2
3  public class Main
4  {
5      public static void main(String[] args)
6      {
7          TicketMachine tm1;
8          tm1 = new TicketMachine(300);
9
10         tm1.insertMoney(200);
11
12         System.out.println("Balance: "+tm1.getBalance());
13
14         tm1.insertMoney(100);
15
16         tm1.printTicket();
17     }
18 }
```

2.2 Chapter Exercises

Exercise 2.21

Suppose that the class Pet has a field called name that is of type String. Write an assignment statement in the body of the following constructor so that the name field will be initialized with the value of the constructor's parameter.

```
public Pet(String petsName)
{
    name = petsName;
}
```

Exercise 2.22 (challenge)

The following object creation will result in the constructor of the Date class being called. Can you write the constructor's header?

```
new Date("March", 23, 1861)
```

Try to give meaningful names to the paramenters.

```
public Date(String month, int day, int year)
{
    ...
}
```

2.3 Selfstudy-Questions OOP2

Exercise 4

A class is build by three essential components. What are they?

- Instance variables (member variables, attributes)
- constructor
- methods

Exercise 5

What is the order of the three components?

The order doesn't matter technically but there is a common convention:

1. instance variables
2. constructor
3. methods

Exercise 6

What's their purpose?

instance variables are holding data of an object. All of this data together builds the object's state.

constructor is a special method that initializes objects.

methods are sequences which are defining the object's behaviour and characteristics.

Exercise 8

What is a variable?

A variable (or field) is a data storage inside an object that can be used for persistent data storage (limited by the lifetime of the object).

Exercise 9

What are the synonyms to instance variables?

- member variable
- attribute
- filed
- variable

Exercise 10

What do you think where the term instance variable comes from?

An instance is a realisation of an class by an object. The expression variable is well defined an known in computer science and if a variable explicitly belongs to an object, so it's clear that this is a variable of an instance or instance variable.

Exercise 11

How can you put comments into a Java-Code?

There are different ways to add comments in a Java source file without having trouble with the compiler.

- Use the single line comment by double slash.

```
// this method return the speed
private void getSpeed()
```

- Use the multiline comment by slash-dot

```
/**
 * This is a method that will return the
 * actual speed of the monstetruck that
 * is driven by the crazy clown IT .
 */
private void getSpeed()
```

Exercise 12 (important)

With which access-modification do you declare instance variables usually? Is it **`private`** *or* **`public`***? Do you have a reason for your answer?*

Usually we declare instance variables as private. The reason for this is a common pattern that is used to get or set these data form outside the objects by so called accessor and mutator methods (getSpeed, setSpeed, changeSpeed).

Exercise 13

Explain the relation between a constructor and the state of an onject.

The constructor is creating (initializing) an object and has nothing to do with the state of the object once it's set up.

Exercise 14

How do we name constructors?

Constructors are usually named after the class their used for.

Exercise 15

What's the lifetyme of instance variables, how long are they reachable/accessable?

The lifetime of variables is coupled to the lifetime of their objects. As long as the object is alive the variables are also alive.

Exercise 16

Why sould you (if possible) initialise instance variables explicit?

If we don't initialize variables explicit the compiler will use default values for the initialization. By explicit initialisation we don't have any disadwantage and it serves well to document what is sctually happening.

Exercise 17

What's the defualt value which is given to a `int` *variavle by its initialisation?*

The default value for an `int` is zero.

Exercise 19

What's the use of parameters?

Parameters provide additional information to a method or object. This is useful in many ways.

Exercise 20

What's the difference between a formal and a actual parameter?

A formal parameter is a parameter that is defined as parameter but has no actual value corresponding. A actual parameter is a parameter with a specific value.

Exercise 21

Is the following statement correct; "formal parameters are special variables"?

Parameters are temporary and restricted variables because their space is allocated by a call to the method or object and as soon as a value is transmitted to it. Once that call has completed its task, the formal parameter disappears and the values in it are lost.

Exercise 22

What's about the accessability of formal parameters?

The accessability of parameters are limited to the lifetime of the task which is creating them (method). Also parameter are only reachable from inside the box that they are used in (like a local variable).

Exercise 23

In which way this differs from instance variables?

Instance variables have a lifetime that is identical with the lifetime of their objects. Also parameters are only reachable from inside the block, instance variables are reachable from everywhere inside the class.

Exercise 24

How do the lifecycles of formal parameters and instance variables differ?

Instance variables are persistent (limited by lifetime of the object) and the lifetime of formal parameters is not really defined in runtime.

Exercise 26

How would you translate the expressions "assignment" and "expression" in german?

- assignment = Zuweisung
- expression = Ausdruck

Exercise 27

How does an assignment-instruction work exactly? What's about to be aware of in relation to data types?

An assignment can be done with the operator "=". For example:

```
// create a instance variable for speed
private int speed;

// set the speed
public void setSpeed(int newSpeed)
{
    speed = newSpeed;
}
```

By assigning data you have to be aware of data types. For example you can't assign a **int** to a **float** and so on. There are some strategies to "cast" or "parse" data between different data types but that's not our topic now.

2.4 Team Exercise 1-4

Create a Balloon-Class and create some objects and interact with them.

../workspace/balloon/src/flight/Balloon.java

```java
package flight;

/**
 * Balloon models a simple abstraction of a physical balloon.
 */

public class Balloon
{
    // size of the balloon. The balloon is abstracted a perfect
    // bowl defined by its diameter.
    private float diameter;

    // horizontal position of the balloon
    private int posHorizontal;

    // altitude (vertical position) of the balloon
    private int posVertical;

    // color of the balloon
    private String color;

    // number of the ballon
    private int number;

    // simple constructor
    public Balloon()
    {
        diameter = 300f;
        posHorizontal = 300;
        posVertical = 300;
        color = "red";
    }

    // more detailed constructor
    public Balloon(String newColor)
    {
        color = newColor;
    }

    public void setPosition(int newHorizontal, int newVertical)
    {
        posHorizontal = newHorizontal;
        posVertical = newVertical;
    }

    public void setDiameter(float newDiameter)
    {
        diameter = newDiameter;
    }

```

```
51      public void setColor(String newColor)
52      {
53          color = newColor;
54      }
55
56      public void setNumber(int newNumber)
57      {
58          number = newNumber;
59      }
60
61      public int getHorizontal()
62      {
63          return posHorizontal;
64      }
65
66      public int getVertical()
67      {
68          return posVertical;
69      }
70
71      public float getDiameter()
72      {
73          return diameter;
74      }
75
76      public String getColor()
77      {
78          return color;
79      }
80
81      public int getNumber()
82      {
83          return number;
84      }
85  }
```

../workspace/balloon/src/flight/Main.java

```
1   package flight;
2
3   public class Main
4   {
5       public static void main(String[] args)
6       {
7           // create a new balloon (with the simple constructor)
8           Balloon b1 = new Balloon();
9           // get the current horizontal position
10          System.out.println("Horizontal: " + b1.getHorizontal());
11          // set a new horizontal position
12          b1.setPosition(400, 400);
13          // get the current horizontal position
14          System.out.println("Horizontal: " + b1.getHorizontal());
15
16          // create a new ballon with the detailed constructor
17          Balloon b2 = new Balloon("yellow");
```

```
18          // get the color of the new ballon
19          System.out.println("Color: " + b2.getColor());;
20      }
21  }
```

2.5 Team Exercise 5

You want to write records, so you have to write a class Book for this. This class shall have the following four attributes:

- Title (String)
- Author (String)
- Price (float)
- Year on buy (int)

The class shall also have two constructors.

- Title and author are parameters. The books is not bought yet and this is why the price is 0.0 and the "year of buy" is -1.
- All attributes are initialized by parameters.

The class shall have the following methods.

- Two methods to get the title and author.
- A method to get and to set the year of buy.
- A method to get and to set the price.

../workspace/Book/src/library/Book.java

```
1  package library;
2
3  public class Book
4  {
5      // title of the book
6      private String title;
7
8      // author of the book
9      private String author;
10
11     // price of the book
12     private float price;
13
14     // year of buy
15     private int year;
16
17     /**
18      * Create a new book with all attributes.
19      */
20     public Book(String newTitle, String newAuthor, float newPrice, int
           newYear)
21     {
22         title = newTitle;
23         author = newAuthor;
24         price = newPrice;
25         year = newYear;
26     }
27
28     public Book(String newTitle, String newAuthor)
```

```
29      {
30          title = newTitle;
31          author = newAuthor;
32          price = 0.0f;
33          year = -1;
34      }
35
36      public String getTitle()
37      {
38          System.out.println("Title: " + title);
39          return title;
40      }
41
42      public String getAuthor()
43      {
44          System.out.println("Author: " + author);
45          return author;
46      }
47
48      public void setYear(int newYear)
49      {
50          year = newYear;
51      }
52
53      public void setPrice(float newPrice)
54      {
55          price = newPrice;
56      }
57
58      public int getYear()
59      {
60          System.out.println("Year: " + year);
61          return year;
62      }
63
64      public float getPrice()
65      {
66          System.out.println("Price: " + price + " USD");
67          return price;
68      }
69
70  }
```

../workspace/Book/src/library/Main.java

```
1   package library;
2
3   public class Main
4   {
5       public static void main(String[] args)
6       {
7           Book b1 = new Book("Objects First with Java", "David Barnes",
                70.0f, 2013);
8           b1.getTitle();
9           b1.getAuthor();
```

```
10          b1.getYear();
11          b1.getPrice();
12      }
13  }
```

2.6 Team Exercise 5 - Optional

Think about banc accounts, their behaviour and attributes. Implement a class Account. To avoid round sum problems work with integer values. Play around with your class and get you some money!

../workspace/Account/src/money/Account.java

```
1  package money;
2
3  public class Account
4  {
5      private String ownerFirstName;
6      private String ownerLastName;
7      private String ownerAddress;
8      private String ownerEMail;
9      private int yearOfBirth;
10     private int yearOfAccount;
11     private int accountNumber;
12     private long accountBalance;
13     private long accountDebit;
14     private long accountCredit;
15     private boolean accountActive;
16
17     /**
18      * Create a new inactive account with default values.
19      */
20     public Account()
21     {
22         ownerFirstName = "Default";
23         ownerLastName = "Default";
24         ownerAddress = "Default";
25         ownerEMail = "Deafult";
26         yearOfBirth = -1;
27         yearOfAccount = -1;
28         accountNumber = -1;
29         accountBalance = 0;
30         accountDebit = 0;
31         accountCredit = 0;
32         accountActive = false;
33     }
34
35     /**
36      * Create a new active account.
37      */
38
39     public Account( String newFirstName,
40                     String newLastName,
41                     String newAddress,
42                     String newEMail,
43                     int newYearOfBirth,
44                     int newYearOfAccount,
45                     int newAccountNumber,
46                     long newAccountBalance)
47     {
48         ownerFirstName = newFirstName;
49         ownerLastName = newLastName;
```

```
50          ownerAddress = newAddress;
51          ownerEMail = newEMail;
52          yearOfBirth = newYearOfBirth;
53          yearOfAccount = newYearOfAccount;
54          accountNumber = newAccountNumber;
55          accountBalance = newAccountBalance;
56          accountDebit = 0;
57          accountCredit = 0;
58          accountActive = true;
59      }
60
61      public String getOwnerFirstName()
62      {
63          System.out.println("First name: " + ownerFirstName);
64          return ownerFirstName;
65      }
66
67      public void setOwnerFirstName(String newOwnerFirstName)
68      {
69          ownerFirstName = newOwnerFirstName;
70      }
71  }
```

../workspace/Account/src/money/Main.java

```
1   package money;
2
3   public class Main
4   {
5       public static void main(String[] args)
6       {
7           Account acc1 = new Account();
8           acc1.getOwnerFirstName();
9           acc1.setOwnerFirstName("David Barnes");
10          acc1.getOwnerFirstName();
11      }
12  }
```

2.7 Selfstudy-Questions OOP3

Exercise 1

What is a header? What is a body?

A header is a part of a method. For example `public int getSpeed()` is the header of the method

```
1   public int getSpeed()
2   {
3       return speed;
4   }
```

Exercise 2

Write down the signatures of the methods form class TicketMachine.

- getPrice(...)
- getBalance(...)
- insertMoney(int ...)
- printTicket(...)

Exercise 3

Where can you place expressions and definitions?

I don't understand that question.

Exercise 4

What is a block?

A block ist the part of a method which is between the curly braces.

Exercise 5

How many `return` *expressions do you find in Code 2.1?*

Exercise 7

What's the meaning of the return-type `void`*?*

The return type `void` indicates that the method has no return value.

Exercise 8

Fill out the table.

compound assignment	assignment
`a += b`	`a = a + b`
`a -= b`	`a = a - b`
`a *= b`	`a = a * b`
`a /= b`	`a = a / b`

Exercise 9

In the code of the TicketMachine, there are two places where you can place a compound assignment operator. Find those two places.

Exercise 12

Describe the conditional operator of the pseudo-code on page 42 in german. Try to translate the code in german (except for the keywords `if` *and* `else`*.*

Exercise 16

At pitfall on page 48 is a very important information. Translate the first sentence in german.

Exercise 17

Fill out the following table.

	Field	**formal parameter**	**local variable**
can store values?	Yes	No	limited
Where is/are they defined?	class	class	method
How long do they exist?	permanent	imaginary	limited
From where can you access them?	global	nowhere	localy

2.8 Team Exercise 1

```
2.5 * (2+3) = 12.5
(int) 2.5 * 2 + 3 = 7
(int) 2.5 * (2+3) = 10
(int) (2.5 * 2 + 3) = 8
(int) (2.5 * (2+3)) = 12
(int) 2.5 * 2 + (float) 3 = 7.0
```

2.9 Team Exercises 2

- *What are konventions?*
  Konventions are rules that are not strict.
- *For whatr are they good for?*
  They give the programmer a good orientation if every coder used the same conventions or if a coder used a convention consistantly. This improves the portability and make the code easy to maintain.
- *Give the signatures for the following attributes by konventions:*
  - `String secondName`
  - `float hours`
  - `int personalNumber`
  - `Object myObject`

  Answer

  - `public String getSecondName()`
  - `public float getHours()`
  - `public int getPersonalNumber()`
  - `public Object getMyObject()`
- *Can you define a konvention for mutator-methods?*
  Of course you can, just as for getter-methods.

2.10 Team Exercise 3

At the exercises 2.27 and 2.59 you had to note error messages.

- *Compare your Notes*
  "Missing return statement" and "unreachable statment".
- *Try to define rules out of these error messages.*
  If you use a return declaration in the header you have to use a return expression and the return expression has to be at the end of the block.

2.11 Team Exercise 4

Now you'll get into the **switch** expression on your own. You can use the appendix D of your book and the file `Selection.jar` from ILIAS

2.11.1 Team Exercise 4.1

Look at the following snippet.

```
1  public void output(int value)
2  {
3      System.out.println();
4      System.out.println("actual parameter: " + value);
5      switch(value)
6      {
7          case 1:
8              System.out.println("one");
9              break;
10         case 2:
11             System.out.println("two");
12             break;
13         case 3:
14             System.out.println("three");
15             break;
16         default:
17             System.out.println("other value");
18             break;
19     }
20 }
```

2.11.2 Team Exercise 4.2

../workspace/Selection/src/choose/Selection.java

```
1  /* Copyright 2012 Hochschule Luzern - Technik & Architektur */
2
3  package choose;
4
5  /**
6   * Klasse Selection für die Lernaufgabe zu switch.
7   * @author Peter Sollberger
8   */
9  public class Selection
```

```
10 {
11
12     /**
13      * Der Konstruktor von Selection ist "leer".
14      */
15     public Selection()
16     {
17     }
18
19     /**
20      * In Abhängigkeit des übergebenen Wertes erfolgt die
21      * Ausgabe eines Textes.
22      */
23     public void output(int value)
24     {
25         System.out.println("aktueller Parameter: " + value);
26
27         switch (value)
28         {
29             case 1:
30                 System.out.println("eins");
31             case 2:
32                 System.out.println("zwei");
33             case 3:
34                 System.out.println("drei");
35             default:
36                 System.out.println("anderer Wert");
37         }
38     }
39 }
```

../workspace/Selection/src/choose/Main.java

```
1  package choose;
2
3  public class Main
4  {
5      public static void main(String[] args)
6      {
7          Selection mySel = new Selection();
8          mySel.output(5);
9          System.out.println("END OF PROGRAM");
10     }
11 }
```

2.12 Summary exercises

3 Object interaction

3.1 Selfstudy-Questions OOP4

3.1.1 Chapter 3.6 - Class diagrams vs. object diagrams

Exercise 1

How do you declare a referencevariable?

A referencevariable is a variable that points to an object. For example `Account myAccount = new Account();` defines a referencevariable `myAccount`. This variable doesn't contain a value but a reference to the storage-space where the object lays (like a pointer in C).

Exercise 2

Draw the object diagram to the BlueJ project "house" from chapter 1.

Exercise 3

Draw the class diagram to the BlueJ project "house" from chapter 1.

Exercise 4

Solve the exercises 3.1 to 3.4

3.1.2 Chapter 3.8 - The ClockDisplay source code

Exercise 1

Solve the exercise 3.5

Exercise 2

What is the result of the following expressions?

Question			Result
`(3>2)`	`^`	`(4>5)`	**`true`**
`(3<2)`	`^`	`(4>5)`	`fasle`
`(3<2)`	`&&`	`(4>5)`	**`false`**
`(3>2)`	`||`	`(4>5)`	**`true`**
`!(3>2)`			**`false`**

Exercise 3

Solve the exercises 3.6 to 3.8

3.6 Nothing happens. This implemetation is not a good idea. To improve it we could use a error-message that is returned.

3.7 We could not set the value to zero.

3.8 If would be true for all inputs, because their either >0 or <limit.

Exercise 4

Solve the exercises 3.15 to 3.17 and 3.19

3.15 The modulo operator returns the remainder of an division.
3.16 `8%3` returns 2
3.17 `-10%3` returns -1, `10%-3` returns +1.
3.18 5-1
3.19 m-1

Exercise 5

Solve the exercise 3.21

3.21

```
if((value+1) < limit){
    value++;
}
else{
    value = 0;
}
```

3.1.3 Chapter 3.9 - Objects creating objects

Exercise 1

Solve the exercise 3.23

3.23 The time is "00:00". The constructor is responsible for this value.

3.1.4 Chapter 3.10 - Multiple constructors

Exercise 1

Create the singatures for all possible constructors which accord with the following object-creation.
`new Student("Peter", 34);`

```
// simple creator with no parameters
public Student()
{
    name = "No-Name";
    age = -1;
}

// creator with single-parameter name
public Student(String newName)
{
    name = newName;
    age = -1
}

// creator with single-parameter age
```

```
16 public Student(int newAge)
17 {
18     name = "No-Name";
19     age = newAge;
20 }
21
22 // creator with full paramterlist name, age
23 public Student(String newName, int newAge)
24 {
25     name = newName;
26     age = newAge;
27 }
```

Exercise 2

Solve the exercises 3.28 and 3.29

3.28 It creates two `NumberDisplay` objects with the overrolllimits 24 and 60.

3.29 Because it is set by the parameters given to the constructor.

3.1.5 Chapter 3.11 - Method calls

Exercise 1

Solve the exercise 3.30

3.30

```
1  // print the Payroll-Summary on Printer p1, two-sided
2  p1.print("Payroll-Summary.txt", true)
3
4  // print the Phone-List on Printer p1, single-sided
5  p1.print("Phone-List.txt", false)
6
7  // show the status of Printer p1 on the console
8  System.out.println(p1.getSatus(20))
9
10 // return the status of Printer p1
11 p1.getStatus(10)
```

3.1.6 Chapter 3.12 - Another example of object interaction

Exercise 1

Solve the exercises 3.33 and 3.34

3.33

../workspace/Mail–System/src/mails/Main.java

```
1 package mails;
2
3 public class Main
```

```
{
    public static void main(String[] args)
    {
        // create a MailServer
        MailServer MS1 = new MailServer();

        // create two MailClients
        MailClient MC1 = new MailClient(MS1, "Homer");
        MailClient MC2 = new MailClient(MS1, "Fry");

        // send a message from MC1 to MC2
        MC1.sendMailItem("Fry", "Hello Fry! How are you?");

        // show the mail at MC2
        MC2.printNextMailItem();

        // give an answer
        MC2.sendMailItem("Homer", "Hi Homer! I'm fine, thanks.");

        // show the mail at MC1
        MC1.printNextMailItem();
    }
}
```

../workspace/Mail–System/src/mails/MailServer.java

```
package mails;

import java.util.ArrayList;
import java.util.List;
import java.util.Iterator;

/**
 * A simple model of a mail server. The server is able to receive
 * mail items for storage, and deliver them to clients on demand.
 *
 * @author David J. Barnes and Michael Kölling
 * @version 2011.07.31
 */
public class MailServer
{
    // Storage for the arbitrary number of mail items to be stored
    // on the server.
    private List<MailItem> items;

    /**
     * Construct a mail server.
     */
    public MailServer()
    {
        items = new ArrayList<MailItem>();
    }

    /**
     * Return how many mail items are waiting for a user.
```

```
30      * @param who The user to check for.
31      * @return How many items are waiting.
32      */
33     public int howManyMailItems(String who)
34     {
35         int count = 0;
36         for(MailItem item : items) {
37             if(item.getTo().equals(who)) {
38                 count++;
39             }
40         }
41         return count;
42     }
43 
44     /**
45      * Return the next mail item for a user or null if there
46      * are none.
47      * @param who The user requesting their next item.
48      * @return The user's next item.
49      */
50     public MailItem getNextMailItem(String who)
51     {
52         Iterator<MailItem> it = items.iterator();
53         while(it.hasNext()) {
54             MailItem item = it.next();
55             if(item.getTo().equals(who)) {
56                 it.remove();
57                 return item;
58             }
59         }
60         return null;
61     }
62 
63     /**
64      * Add the given mail item to the message list.
65      * @param item The mail item to be stored on the server.
66      */
67     public void post(MailItem item)
68     {
69         items.add(item);
70     }
71 }
```

../workspace/Mail–System/src/mails/MailClient.java

```
1  /**
2   * A class to model a simple email client. The client is run by a
3   * particular user, and sends and retrieves mail via a particular server.
4   *
5   * @author David J. Barnes and Michael Kölling
6   * @version 2011.07.31
7   */
8 
9  package mails;
10 
```

```java
public class MailClient
{
    // The server used for sending and receiving.
    private MailServer server;
    // The user running this client.
    private String user;

    /**
     * Create a mail client run by user and attached to the given server.
     */
    public MailClient(MailServer server, String user)
    {
        this.server = server;
        this.user = user;
    }

    /**
     * Return the next mail item (if any) for this user.
     */
    public MailItem getNextMailItem()
    {
        return server.getNextMailItem(user);
    }

    /**
     * Print the next mail item (if any) for this user to the text
     * terminal.
     */
    public void printNextMailItem()
    {
        MailItem item = server.getNextMailItem(user);
        if(item == null) {
            System.out.println("No new mail.");
        }
        else {
            item.print();
        }
    }

    /**
     * Send the given message to the given recipient via
     * the attached mail server.
     * @param to The intended recipient.
     * @param message The text of the message to be sent.
     */
    public void sendMailItem(String to, String message)
    {
        MailItem item = new MailItem(user, to, message);
        server.post(item);
    }
}
```

../workspace/Mail–System/src/mails/MailItem.java

```java
/**
 * A class to model a simple mail item. The item has sender and recipient
 * addresses and a message string.
 *
 * @author David J. Barnes and Michael Kölling
 * @version 2011.07.31
 */

package mails;

public class MailItem
{
    // The sender of the item.
    private String from;
    // The intended recipient.
    private String to;
    // The text of the message.
    private String message;

    /**
     * Create a mail item from sender to the given recipient,
     * containing the given message.
     * @param from The sender of this item.
     * @param to The intended recipient of this item.
     * @param message The text of the message to be sent.
     */
    public MailItem(String from, String to, String message)
    {
        this.from = from;
        this.to = to;
        this.message = message;
    }

    /**
     * @return The sender of this message.
     */
    public String getFrom()
    {
        return from;
    }

    /**
     * @return The intended recipient of this message.
     */
    public String getTo()
    {
        return to;
    }

    /**
     * @return The text of the message.
     */
    public String getMessage()
    {
```

```
55          return message;
56      }
57
58      /**
59       * Print this mail message to the text terminal.
60       */
61      public void print()
62      {
63          System.out.println("From: " + from);
64          System.out.println("To: " + to);
65          System.out.println("Message: " + message);
66      }
67  }
```

3.34

3.1.7 Chapter 3.13 - Using a debugger

Exercise 1

Solve the exercises 3.35 to 3.42

3.35 to 3.42

../workspace/Mail–System/src/mails/Sophie.java

```
1  package mails;
2
3  public class Sophie
4  {
5      public static void main(String[] args)
6      {
7          // create a MailServer
8          MailServer MS1 = new MailServer();
9
10         // create two clients
11         MailClient sophie = new MailClient(MS1, "Sophie");
12         MailClient juan = new MailClient(MS1, "Juan");
13
14         // send a message from sophie to juan
15         sophie.sendMailItem("Juan", "Hello Juan. How are you?");
16
17         // print the message at juans client
18         juan.printNextMailItem();
19
20         // check for new messages
21         juan.printNextMailItem();
22     }
23 }
```

3.2 Team-Exercises

3.2.1 Exercise 1 - Using a debugger

Exercise 3.43, page 90

Exercise 3.44, page 90

3.2.2 Exercise 2 - Some random exercises

Exercise 3.9, page 71

Which of the following expressions return true?

Expression	Result
`!(4<5)`	**true**
`!false`	**true**
`(2>2)|| ((4==4)&& (1<0))`	**false**
`(2>2)|| (4==4)&& (1<0)`	**false**
`(34 != 33)&& ! false`	**true**

Exercise 3.10, page 71

Write an expression using boolean variables a and b that evaluates to true when a and b are either true or both false.

```
!(a^b)
```

Exercise 3.11, page 71

Write an expression using boolean variables a and b that evaluates to true when only one of a and b is true, and that is false if a and b are both false or both true.

```
(a^b)
```

Exercise 3.12, page 71

Consider the following expression. Write an equvalent expression (one that evaluates true at exactly the same values for a and b) without using the AND Operator.
`(a&&b)`

3.2.3 Exercise 3 - Challanges

3.2.4 Exercise 4 - Programming (optional)

Exercise 3.45, page 91

Add a subject line for an e-mail to mail items in the mail-system project. Make sure printing messages also prints the subject line. Modify the mail client accodringly.

Exercise 3.46, page 91

Given the following class write some lines of java code that create a Screen object. Then call its clear method if (and only if) its number of pixels is greater than two million. (Don't worry about things being logical here; the goal is only to write something that is syntactically correct - i.e., that would compile if we typed it in.)

4 Grouping objects

4.1 Selfstudy-Questions OOP5

4.1.1 Chapter 4.1 to 4.3 - An organizer for music files

Exercise 1

Solve the exercises 4.1 to 4.3

4.1

../workspace/Music–Organiser–V1/src/music/MusicOrganizer.java

```
1  package music;
2
3  import java.util.ArrayList;
4
5  /**
6   * A class to hold details of audio files.
7   *
8   * @author David J. Barnes and Michael Kölling
9   * @version 2011.07.31
10  */
11 public class MusicOrganizer
12 {
13     // An ArrayList for storing the file names of music files.
14     private ArrayList<String> files;
15
16     /**
17      * Create a MusicOrganizer
18      */
19     public MusicOrganizer()
20     {
21         files = new ArrayList<String>();
22     }
23
24     /**
25      * Add a file to the collection.
26      * @param filename The file to be added.
27      */
28     public void addFile(String filename)
29     {
30         files.add(filename);
31     }
32
33     /**
34      * Return the number of files in the collection.
35      * @return The number of files in the collection.
36      */
37     public int getNumberOfFiles()
38     {
39         return files.size();
40     }
41
42     /**
43      * List a file from the collection.
```

```
44      * @param index The index of the file to be listed.
45      */
46     public void listFile(int index)
47     {
48         if(index >= 0 && index < files.size()) {
49             String filename = files.get(index);
50             System.out.println(filename);
51         }
52     }
53
54     /**
55      * Remove a file from the collection.
56      * @param index The index of the file to be removed.
57      */
58     public void removeFile(int index)
59     {
60         if(index >= 0 && index < files.size()) {
61             files.remove(index);
62         }
63     }
64 }
```

../workspace/Music–Organiser–V1/src/music/Main.java

```
1  package music;
2
3  public class Main
4  {
5      public static void main(String[] args)
6      {
7          // create a MusicOrganiser object
8          MusicOrganizer myOrg = new MusicOrganizer();
9
10         // store some tracks to it
11         myOrg.addFile("Free Software Song");
12         myOrg.addFile("Hacker after all");
13         myOrg.addFile("CRE197 - IPV6");
14
15         // check the number of tracks that are stored
16         System.out.println("Number of Files: " +
17                            myOrg.getNumberOfFiles());
18
19         // show the name of the first, second and third track
20         myOrg.listFile(0);
21         myOrg.listFile(1);
22         myOrg.listFile(2);
23
24         // remove the first track and ask for the first track
25         myOrg.removeFile(0);
26         myOrg.listFile(0);
27     }
28 }
```

4.2 We don't get an error. As wee look at the code of the method we'll see, that the method is examinating if the index is "valid". If not it does not perform a remove action.

../workspace/Music–Organiser–V1/src/music/MusicOrganizer.java

```
1     public void removeFile(int index)
2     {
3         if(index >= 0 && index < files.size()) {
4             files.remove(index);
5         }
6     }
```

4.3 The list is shifted, so that the previously second track is the first track after removing the first track.

Exercise 2

What do you understand by "Java-Package"?

A Java-Package is just a collection of classes. It is a namespace to organize classes.

Exercise 3

You want to use the library-class ArrayList. What expression makes it possible to use that library-class in your source code?

If we want to use a library-class, we have to import it to our source with
`import java.nameOfTheLibraryClass;`

So to use the ArrayList we just have to write in our source
`import java.util.ArrayList;`

4.1.2 Chapter 4.4 to 4.7 - Numbering within collections

Exercise 4

Solve the exercises 4.4 to 4.7

Exercise 4.4, page 100

Write a declaration of a private field named library that can hold an ArrayList. The elements of the ArrayList are of type Book.

```
private ArrayList<Book> library = new ArrayList<String>();
```

Exercise 4.5, page 100

Write a declaration of a local variable called cs101 that can hold an ArrayList of Student.

```
private ArrayList<Student> cs101 = new ArrayList<Student>();
```
I'm not really sure about the "local" in the exercise. Do I have to specify that this is private or not?

Exercise 4.6, page 101

Write a declaration of a private field called tracks for sorting a collection of MusicTrack objects.

```
private ArrayList<MusicTrack> tracks = new ArrayList<MusicTrack>();
```

Exercise 4.7, page 101

Write assignments to the library, cs101 and track variables (which you defined in the previous three exercises) to create the appropriate ArrayList objects. Write them once without using diamond and once with diamond notation if you are using Java 7 compiler.

```
1 library.add("Objects First with Java");
2 cs101.add("Leonardo DaVinci");
3 track.add("Free Software Song");
```

I'm not really sure what is the question here ...

Exercise 5

Solve the exercises 4.8 to 4.11

Exercise 4.8, page 102

If a collection stores 10 objects, what value would be returned from a call to its size method?

It would return 9.

Exercise 4.9, page 102

Write a method call using get to return the fifth object stored in a collection called items.

```
items.get(4)
```

Exercise 4.10, page 102

What is the index of the last item stored in a collection of 15 objects?

This would be 14.

Exercise 4.11, page 102

Write a method call to add the object held in the variable favoriteTrack to a collection called files.

```
addFavorite(favoriteTrack, files)
```
I'm not really sure about this question ...

Exercise 6

Solve the exercises 4.12 to 4.13

Exercise 4.12, page 103

Write a method call to remove the third object stored in a collection called dates.

```
dates.remove(2)
```

Exercise 4.13, page 103

Suppose that an object is stored at index 6 in a collection. What will be its index immediately after the objects at index 0 and 9 are removed?

After removing index 0, the whole collection is shifted by -1, so the index of the element, which was at index 5 in the beginning would now be at $6 - 1 = 5$. Removing an index after the queried one has non effect of the indexing, so it would still be 5.

Exercise 7

Explain the following declaration:

```
private ArrayList<Balloon> list = new ArrayList<>();
```

A private collection (ArrayList) of type Balloon is set up.

Exercise 8

What is the connection between abstraction and ArrayLists?

Abstraction has the goal to simplify as far as possible. The class ArrayList provides great funcionalities that are helping to use the conecept of abstraction by minimizing effort in programming a complex class but using a library-class (ArrayList) which is giving a lot of useful and powerful methods. In other words, if we want to manage a collection, we don't want to use our time to implement complex code to manage our collection, we just want to specify how we want to manage. So we use preexisting code (from a library-class like java.util.ArrayList).

Exercise 9

What is the difference of the methods remove() and get() on ArrayLists?

The `remove()` method of ArrayList is removing the specified objects out of the collection. This is causing a reindexation of all items of the collection that have a higher index as the removed once. These following items will get a new index which is "actual index - 1".

The `get()` method of ArrayList is returning the element at the specified index. The element is in general a object but it could also be a simple type.

4.1.3 Chapter 4.8 to 4.12 - The Iterator type

Exercise 10

Solve the exercises 4.18 to 4.19

Exercise 4.18, page 106

What might the header of a listAllFiles method in the MusicOrganizer class look like? What sort of return type should it have? Does it need to take any parameters

The header of such a method would certainly not have a parameter and probably the return type void since it would use `System.out.println()` so the header could look like

```
public void listAllFiles()
```

Exercise 4.19, page 106

We know that the first file name is stored at index zero in the ArrayList and the list stores the file names as strings, so could we write the body of listAllFiles along with the following lines?

```
System.out.println(files.get(0));
System.out.println(files.get(1));
System.out.println(files.get(2));
```

We could do so if we would be sure that there won't be more than three used indexes or three tracks. If we would like to have an arbitrary number of tracks we should use a other body, for instance with a loop.

A easy and intended loop for this job is a so called *for-each loop*. The course book is suggesting the following code.

```
public void listAllFiles()
{
    for(String filename : files)
    {
        System.out.println(filename);
    }
}
```

Exercise 11

Solve the exercise 4.22

Just something to play in BlueJ (creating an ArrayList and add, remove, . . . , some objects to it).

Exercise 12

Explain as detailed as possible the source code on page 108.

```
public void listAllFiles()
{
    for(String filename : files)
    {
        System.out.println(filename);
    }
}
```

In the first line we see the header of the method. It is declaring that the method is of type public, that it has no return type (void) and does not take any parameters.

In the third line we see the header of the so called for-each loop. This means translated "for each element of the ArrayList files, do the following body". Since the ArrayList files is containing Strings, we need to declare "element" as String. So for every round thru the loop, the incremented index of the element that it contains at this index is stored to the local variable "element" that we defined to be of type String.

In the fifth line of code we see the print method, which is printing the "element" to the console, so it prints the String that is at the actual index of the loop.

Exercise 13

Is it possible, that the body of an while-loop is never executed?

Yes of course it is possible because there is a condition that is deciding if the body is run or not.

Exercise 14

Show two alternative expressions for no++

```
n += n;
```

```
n = n + n;
```

Exercise 15

An ArrayList can be traversed by an foreach-loop. Do you know other ways to do the same?

Yes, there is a so called Iterator.

Exercise 16

Is hasNext() a method of ArrayList or Iterator? How do you have to understand/interpret the return-value of hasNext()?

`hasNext()` is a method of Iterator. This method returns a **`true`** if there is a element in the collection after the current one and **`false`** if not.

4.1.4 Chapter 4.14 - Summary of the music-organizer project

Exercise 17

DO NOT READ THIS CHAPTER, JUST READ THE CONCEPT-BOX AT PAGE 130.

Exercise 18

A variable that is declared for a classtype (or so called reference-variable) can store the special value null. Explain the situation with a drawing/sketch. What does it look like, if it's storing an object?

The following sketch 1 shows a comparison of two variables of type Balloon. One is just declared with `Balloon a;` and the other is initialized with `Balloon b = new Balloon();`.
The difference is that `b` is showing or pointing to the object in the memory and since `a` has no object to show to is has a pointer to **`null`**.

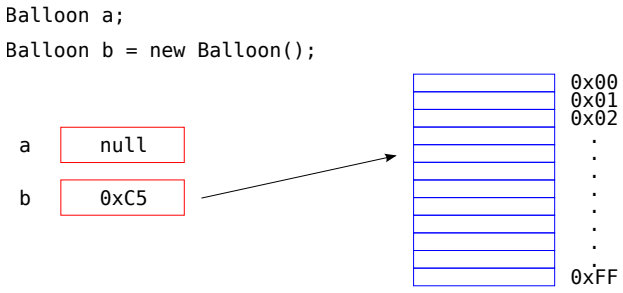

Figure 1: A declared and initialized object in comparison.

4.1.5 Chapter 4.15 to 4.17 - Summary

Exercise 19

Solve the exercises 4.62 to 4.65

4.62 `Person[] people;`
4.63 `boolean[] vacant;`
4.64 ???
4.65 The declaration is wrong (syntax). It should look like
`int[] counts;`
`boolean[] occupied = new boolean[5000];`

Exercise 20

Solve the exercises 4.66 to 4.68

4.66

```
double[] readings;
String[] urls;
TicketMachine[] machines;

readings = new double[6];
urls = new String[90];
machines = new TicketMachine[5];
```

4.67 With `String[] labels = new String[20]` 20 String objects are created.
4.68 The correct expression is `double[] piece = new double[50];`

Exercise 21

What are the pros and cons of Arrays?

- ✓ Arrays are more efficient in access.
- ✓ Arrays can store primitive-types as well as objects.
- ✗ Arrays have a fixed size.

Exercise 22

How do you get the length of an Array?

The length of an array is accessable by the `length` operator.

```
// create an array of type int with the length 50
int[] i = new int[50];

// return the length of the array
public int getLength(int[] array)
{
    return array.length;
}
```

Exercise 23

Solve the exercises 4.69, 4.71, 4.73 and 4.74

4.69 This will fail in an error called *Out Of Bound Execption.*

4.71

```
public void printGreater(double[] marks, double mean)
{
    for(index = 0; index < marks.length;  index++)
    {
        if(marks[index] > mean)
        {
            System.out.println(marks[index])
        }
    }
}
```

4.73

```
public int numberOfAccesses()
{
    int total = 0;
    for(ctr = 0; ctr < hourCounts.length; ctr++)
    {
        total += hourCounts[ctr];
    }
    return total;
}
```

4.74

5 More-sophisticated behavior

5.1 Selfstudy-Questions OOP6

5.1.1 Chapter 5.2 - The TechSupport system

Exercise 1

Solve the exercise 5.1

../workspace/TechSupport/src/support/Main.java

```
1  package support;
2
3  public class Main
4  {
5      public static void main(String[] args)
6      {
7          // create a support system and start it
8          SupportSystem sys = new SupportSystem();
9          sys.start();
10
11     }
12 }
```

Exercise 2

At page 157 and 158 there is a Method start() that uses a while-loop. Create a code snippet with the same functionality by using a do-while loop.

```
1  public void start()
2  {
3      boolean finished = false;
4
5      printWelcome();
6
7      do{
8          String input = reader.getInput();
9          if(input.startsWith("bye")){
10             finished = true;
11         } else{
12             String response = responder.generateResponse();
13             System.out.println(response);
14         }
15     }while(!finished)
16
17     printGoodbye();
18 }
```

5.1.2 Chapter 5.3 - Reading class documentation

Exercise 3

Solve the exercises 5.2 to 5.5 as well as 5.7 to 5.11

5.2 The Java documentation has a clear design. Every class like String is described in different sections, to get as fast as possible to the information needed. This documentation is arranged by the following sections:

- Summary
  - Nested
  - Field
  - Constructor
  - Method
- Detail
  - Field
  - Constructor
  - Method

5.3 The class String has two methods called startsWith, where one method has only one parameter (String) and the other has two parameters (String and int). The method that has only one parameter is returning **true** if the specified string is found at the very beginning of the specified String object. The other method with two parameters is returning **true** if the specified string is found after the specified offset. The offest is a int value representing the number of characters.

Here is an example for the behavior. Both of these **if** statements are returning **true**.

```
String s = "Hi, my name is Avaj Elcaro.";

// check if the very beginning of the String object is "Hi"
if(s.startsWith("Hi")){
    System.out.println("found match at the very beginning");
}

// check if the String object has the string "my" after the 4th character
if(s.startsWith("my", 4)){
    System.out.println("found match after the 4th character");
```

5.4 There is a method in the class String that checks if a String ends with a specified suffix.

```
public boolean endsWith(String suffix)
```

5.5 There is a method in the class String that returns the number of unicode characters in a specified string.

```
public int length()
```

Exercise 4

In which package do you suppose the class FileWriter? Check your guess with the Java API documentation.

The class FileWriter is a part of the package java.io where the io stands for Input-Output.

Exercise 5

With the class BufferReader you can read files line by line. How does that work? You can find the answer in the Java API documentation.

We can use the method readLine() for that job. Here is an example form `http://www.roseindia.net/java/beginners/java-read-file-line-by-line.shtml`

```
1  import java.io.*;
2  class FileRead
3  {
4      public static void main(String args[])
5      {
6          try{
7              // Open the file
8              FileInputStream fstream = new FileInputStream("textfile.txt");
9              // Get the object of DataInputStream
10             DataInputStream in = new DataInputStream(fstream);
11             BufferedReader br = new BufferedReader(new
                   InputStreamReader(in));
12             String strLine;
13             //Read File Line By Line
14             while ((strLine = br.readLine()) != null)   {
15                 // Print the content on the console
16                 System.out.println (strLine);
17             }
18             //Close the input stream
19             in.close();
20         }catch (Exception e){//Catch exception if any
21             System.err.println("Error: " + e.getMessage());
22         }
23     }
24 }
```

5.1.3 Chapter 5.4 - Adding random behavior

Exercise 6

Solve the exercises 5.12 and 5.13

5.12 The Random class is from the package java.util. It is used to generate a stream of pseudorandom numbers. An instance can be constructed with the following code.

```
1  import java.util.Random;
2
3  public class Main
4  {
```

```
5       public static void main(String[] args)
6       {
7           // create an instance of Random
8           Random rand = new Random();
9
10          // print out 10 random numbers between 0 and 20
11          for(int i = 0; i < 10; i++)
12          {
13              System.out.println(rand.nextInt(21));
14          }
15      }
16  }
```

5.13 See 5.12

Exercise 7

Solve the exercise 5.15

If we use `rand.nextInt(100)` we will get any integer number between 0 and 99.

Exercise 8

Solve the exercise 5.18

Simple example of a random response program.

../workspace/Snippets/src/randomResponse/RandomResponse.java

```
1  package randomResponse;
2
3  import java.util.ArrayList;
4  import java.util.Random;
5
6  public class RandomResponse
7  {
8      public static void main(String[] args)
9      {
10         // create an ArrayList for String objects
11         ArrayList<String> responses = new ArrayList<String>();
12
13         // create an Random instance
14         Random rand = new Random();
15         int ans = 0;
16
17         // add some responses to the ArrayList
18         responses.add("Have you tried turning it off and on again?");
19         responses.add("Are you sure you have the latest update installed?");
20         responses.add("Maybe you should upgrade to platinum version.");
21         responses.add("Have you tried a complete new setup?");
22         responses.add("I'm on pause now, I'll be back in 5 minutes.");
23         responses.add("I'm new to this company and I didn't see such a
               problem yet.");
24
25         // give a random responses
26         System.out.println(responses.get(rand.nextInt(responses.size())));
```

```
27        }
28  }
```

5.1.4 Chapter 5.5 - Packages and import

Exercise 9

Solve the exercises 5.21 and 5.22

5.21 See Exercise 8 - 5.18

5.22 If we implement the response generation like in the example code above, it will work for any number of responses in the ArrayList. This is because the code is flexible for the size of the ArrayList.

5.2 Selfstudy-Questions ALG1

Exercise 10

Describe in words or as pseudocode one algorithm per problem.

(a) *Calculate the product of two integers without a multiplication-operator.*

One possible approach to solve this problem could be to use simple addition and iteration. This is because a multiplication, for example $3 \cdot 5 = 15$, can be substituted with $(3 + 3 + 3 + 3 + 3) = (5 + 5 + 5) = 15$.

../workspace/Snippets/src/multiply/Multiplier.java

```
1  package multiply;
2
3  public class Multiplier
4  {
5      public Multiplier()
6      {
7
8      }
9
10     public int multiply(int a, int b)
11     {
12         int result = 0;
13         if((a==0) || (b==0)){
14             return 0;
15         } else if(a < b){
16             for(int i = 0; i < b; i++){
17                 result += a;
18             }
19         } else{
20             for(int i = 0; i < a; i++){
21                 result += b;
22             }
23         }
24         return result;
25     }
26 }
```

../workspace/Snippets/src/multiply/Main.java

```
1  package multiply;
2
3  import java.util.Random;
4
5  public class Main
6  {
7      public static void main(String[] args)
8      {
9          int product = 0;
10         int factor1 = 0;
11         int factor2 = 0;
12
```

```
13          // create a Random instance
14          Random rand = new Random();
15          Multiplier multi = new Multiplier();
16
17
18          // do a multiplication
19          factor1 = rand.nextInt(10);
20          factor2 = rand.nextInt(10);
21          product = multi.multiply(factor1, factor2);
22          System.out.println(factor1 + " times " + factor2 + " = " +
                product);
23      }
24  }
```

An alternative approach is to think of the factors as binary numbers. In the binary system we multiply also by a addition. See the following example: Lets multiply $11 \cdot 14 = 154$. In binary this looks like following:

	1011_b	11_d
$\cdot$	1110_b	14_d
	0000_b	$0 \cdot 1011_b$
$+$	1011_b	$1 \cdot 1011_b$ and leftshift by 1
$+$	1011_b	$1 \cdot 1011_b$ and leftshift by 2
$+$	1011_b	$1 \cdot 1011_b$ and leftshift by 3
$=$	10011010_b	154_d

If we wanted to implement that alorithm, we would have to operate binary, but in fact that would be not really different form the fist approach.

(b) *Find the lowest number out of a sequence like* $S = 4, -1, 50, 10, 0, 1, -2, 5, 10$

../workspace/Snippets/src/search/Main.java

```
1  package search;
2
3  import java.util.Random;
4
5  public class Main
6  {
7      public static void main(String[] args)
8      {
9          // create a random sequence
10         Random rand = new Random();
11         int size = 10;
12         int range = 100;
13         int[] seq = new int[size];
14         String sequence = "Sequence: ";
15         for(int i = 0; i < size; i++){
16             seq[i] = rand.nextInt(range) - (range/2);
17             sequence += seq[i] + ", ";
18         }
19
20         // show the sequence
21         System.out.println(sequence);
22
```

```
23          // get the biggest value
24          System.out.println("Biggest value in sequence: " +
                getBiggest(seq));
25
26          // get smallest value
27          System.out.println("Smallest value in sequence:" +
                getSmallest(seq));
28      }
29
30      public static int getBiggest(int[] sequence)
31      {
32          int biggest = sequence[0];
33          for(int i = 0; i < sequence.length; i++){
34              if(sequence[i] > biggest){
35                  biggest = sequence[i];
36              }
37          }
38          return biggest;
39      }
40
41      public static int getSmallest(int[] sequence)
42      {
43          int smallest = sequence[0];
44          for(int i = 0; i < sequence.length; i++){
45              if(sequence[i] < smallest){
46                  smallest = sequence[i];
47              }
48          }
49          return smallest;
50      }
51  }
```

Exercise 11

Implement an algorithm to calculate the greatest common divisor with the modulo operator (see ALG1 presentation, page 12).

../workspace/Snippets/src/euklid/Main.java

```
1  package euklid;
2
3  import java.util.Random;
4
5  public class Main
6  {
7      public static void main(String[] args)
8      {
9          // create two random numbers
10         Random rand = new Random();
11         int range = 20;
12         int num1 = rand.nextInt(range) + 1;
13         int num2 = rand.nextInt(range) + 1;
14
15         // get the greatest common divisor
```

```
16          System.out.println( "GCD of "
17                              + num1
18                              + " and "
19                              + num2
20                              + " is "
21                              + gcd(num1,num2));
22      }
23
24      public static int gcd(int a, int b)
25      {
26          while((a != 0) && (b != 0)){
27              if(a > b){
28                  a = a % b;
29              }
30              else{
31                  b = b % a;
32              }
33          }
34          return (a + b);
35      }
36  }
```

(a) Test your algorithm with some examples.

(b) Change your algorithm so that you don't use the modulo operator. Change the modulo operation with a combined expression and test your implementation again.
Hint: The modulo operator can be substuituted by a sequence of subtractions that is finished as the result is smaller than the subtrahend.

../workspace/Snippets/src/euklid2/Main.java

```
1  package euklid2;
2
3  import java.util.Random;
4
5  public class Main
6  {
7      public static void main(String[] args)
8      {
9          // create two random numbers
10         Random rand = new Random();
11         int range = 20;
12         int num1 = rand.nextInt(range) + 1;
13         int num2 = rand.nextInt(range) + 1;
14
15         // get the greatest common divisor
16         System.out.println( "GCD of "
17                             + num1
18                             + " and "
19                             + num2
20                             + " is "
21                             + gcd(num1,num2));
22     }
23
24     public static int gcd(int a, int b)
```

```
25      {
26          // the arguments must not be 0
27          if((a == 0) || (b == 0)){
28              return 0;
29          }
30          while(a != b){
31              if(a > b){
32                  a = a-b;
33              } else{
34                  b = b-a;
35              }
36          }
37          return a;
38      }
39  }
```

Exercise 12

What's the order of the algorithm to calculate the n-th pseudo random number z (see OOP6 presentation page 28 and ALG1 presentation page 15)?

$$z_{n+1} = (a \cdot Z_n + r)\%m$$

5.3 Programming Exercise

Write a class FormLetter that is able to produce serial letters. The class has an ArrayList of type FormalAddress (see OOP3). With the method addAddress() a new object of type FormalAddress is generated and added to the ArrayList. With the method void print(int actualYear, String subject, String textBody) a new serial letter is generated for all elements in the ArrayList. The letter contains a greeting, a subject and of cource the text itself. This shall be printed to the console.

../workspace/FormLetter/src/letter/Main.java

```
1  package letter;
2
3  public class Main
4  {
5      public static void main(String[] args)
6      {
7          FormLetter myLetter = new FormLetter();
8
9          myLetter.addAddress("Richard", "Stallman", "Freeroad 91",
               "Freetown");
10         myLetter.addAddress("Linus", "Torvalds", "Unixstreet 101",
               "Helsinki");
11         myLetter.addAddress("Ervin", "Knuth", "Stanfordstreet 5",
               "Brooklyn");
12
13         myLetter.print(2013, "New Kernel Realease",
14                 "I'd like to inform you that there is a new Kernel
                       release.");
15     }
16 }
```

../workspace/FormLetter/src/letter/FormLetter.java

```
package letter;

import java.util.ArrayList;

public class FormLetter
{
    private ArrayList<FormalAddress> fa = new ArrayList<>();

    public FormLetter(){

    }

    public void addAddress( String firstname,
                            String lastname,
                            String address,
                            String city)
    {
        FormalAddress newAdress = new FormalAddress(firstname,
                                                    lastname,
                                                    address,
                                                    city);
        fa.add(newAdress);
    }

    void print(int actualYear, String subject, String textBody)
    {
        for(FormalAddress a : fa){
            System.out.println("====================");
            System.out.println(a.getName());
            System.out.println(a.getAddress());
            System.out.println(a.getCity());
            System.out.println("");
            System.out.println(subject);
            System.out.println("");
            System.out.println("Dear" + a.getName() + ",");
            System.out.println(textBody);
            System.out.println("");
            System.out.println("====================");
        }

    }
}
```

../workspace/FormLetter/src/letter/FormalAddress.java

```
package letter;

public class FormalAddress
{
    private String firstname = "Default";
    private String lastname = "Default";
    private String address = "Default";
    private String city = "Default";
```

```
9
10      public FormalAddress(   String newFirstName,
11                              String newLastName,
12                              String newAddress,
13                              String newCity)
14      {
15          firstname = newFirstName;
16          lastname = newLastName;
17          address = newAddress;
18          city = newCity;
19      }
20
21      public String getName(){
22          return firstname + " " + lastname;
23      }
24
25      public String getAddress(){
26          return address;
27      }
28
29      public String getCity(){
30          return city;
31      }
32  }
```

5.4 Team-Exercise

5.4.1 Exercise 1

Create a class ListIteratorApplication with the attribute of type ArrayList for Strings. Fill the ArrayList with the words of the sentence "With the iterator it is possible to tavers Lists back and forth".

1. *Program a method iterateDown() that is iterating through the ArrayList. Use a Iterator for this method.*
2. *Program a method iterateUp() that is iterating reverse through the ArrayList. Use a for-loop and the get() method form ArrayList.*
3. *Program a method iterateBothWays() that is iterating through the ArrayList and then backwards. Use the ListIterator for this method and take a look at the API documentation of it.*
4. *Optional Ecerxice: Use the StringTokenizer to fill the ArrayList. An example is given by the OOP5 presentation on page 10. Take also a look at the API documentation.*

../workspace/ListApp/src/list/Main.java

```
1  package list;
2
3  public class Main
4  {
5      public static void main(String[] args)
6      {
7          final String input = "With the ListIterator it is possible to "
8                  + "travers Lists back and forth";
9
10         ListIteratorApplication lia = new ListIteratorApplication(input);
```

```
11
12          lia.iterateDown();
13          lia.iterateUp();
14          lia.iterateBothWays();
15      }
16  }
```

../workspace/ListApp/src/list/ListIteratorApplication.java

```
1  package list;
2
3  import java.util.ArrayList;
4  import java.util.Iterator;
5  import java.util.ListIterator;
6  import java.util.StringTokenizer;
7
8  public class ListIteratorApplication
9  {
10     ArrayList<String> sentence = new ArrayList<String>();
11
12     public ListIteratorApplication(String input)
13     {
14         snipString(input);
15
16         /* this would be needed without the snipString() method
17         sentence.add("With");
18         sentence.add("the");
19         sentence.add("ListIterator");
20         sentence.add("it");
21         sentence.add("is");
22         sentence.add("possible");
23         sentence.add("to");
24         sentence.add("travers");
25         sentence.add("Lists");
26         sentence.add("back");
27         sentence.add("and");
28         sentence.add("forth");
29         */
30     }
31
32     public void iterateDown(){
33         Iterator<String> it = sentence.iterator();
34         while(it.hasNext()){
35             System.out.print(it.next() + " ");
36         }
37         System.out.println("");
38     }
39
40     public void iterateUp(){
41         Iterator<String> it = sentence.iterator();
42         for(int i = sentence.size(); i > 0; i--){
43             System.out.print(sentence.get(i-1) + " ");
44         }
45         System.out.println("");
46     }
```

```
47
48      public void iterateBothWays(){
49          ListIterator<String> it = sentence.listIterator();
50          while(it.hasNext()){
51              System.out.print(it.next() + " ");
52          }
53          System.out.println("");
54
55          while(it.hasPrevious()){
56              System.out.print(it.previous() + " ");
57          }
58          System.out.println("");
59      }
60
61      public void snipString(String input){
62          StringTokenizer st = new StringTokenizer(input);
63          while(st.hasMoreElements()){
64              sentence.add(st.nextToken());
65          }
66      }
67  }
```

5.4.2 Exercise 2

A sequence of integers is arranged in an array. A subsequence of a sequence is of arbitrary length but is combined by trailed parts of it. See the following example.

```
int[] series = {5, -8, 3, 4, -5, 7, -2, -7, 3, 5}
int[] subseries1 = {5, -8, 3}
int[] subseries2 = {-5, 7, -2, -7, 3}
```

Of course an empty sequence is also a valid subsequence. A subtotal is the sum of all entries of a subsequence. The subtotal of subseries1 form the example is 0, the subtotal of subseries2 is -4. An empty subsequence has the subtotal 0. The following pseudocode describes a intuitiv algorithm.

```
1  For all start values form 0 up to the length of the sequence
2      For all end values from start value up to the length of the sequence
3          Calculate the sum of the subsequence form start value up to the end
               value
4          Is it the maximum sum?
```

1. *What order has the algorithm described above?*
2. *Create a method public int maxSubtotal(int[] sequence) in class Series, which implements the algorithm above.*
3. *Test your implementation with a testmethod, which generates different sequences and calls the method maxSubtotal().*

5.5 Selfstudy-Questions OOP7

5.5.1 Chapter 5.6 - Using maps for associations

Exercise 1

Solve the exercises 5.24 to 5.30

5.24 HashMap is a parameterized class and these are the mehtods that depend on the type.
5.25 If we want to know how many entries are contained in a map we can use the `size()` method.
5.26 A very simple phone book implementation with HashMap.

../workspace/Snippets/src/phoneBook/Main.java

```
1  package phoneBook;
2
3  public class Main
4  {
5      public static void main(String[] args)
6      {
7          // create a new phoneBook
8          MapTester myContacts = new MapTester();
9
10         // add some entries
11         myContacts.enterNumber("Richard Stallman", "1234567890");
12         myContacts.enterNumber("Donald Ervin Knuth", "1123581321");
13
14         // lookup for the numbers
15         System.out.println(myContacts.lookupNumber("Richard Stallman"));
16         System.out.println(myContacts.lookupNumber("Donald Ervin Knuth"));
17
18         if(myContacts.checkForKey("Richard Stallman")){
19             System.out.println("Yep, he's in your phonebook!");
20         } else{
21             System.out.println("Nope, he's not in your phonebook!");
22         }
23     }
24
25 }
```

../workspace/Snippets/src/phoneBook/MapTester.java

```
1  package phoneBook;
2
3  import java.util.HashMap;
4
5  public class MapTester
6  {
7      private HashMap<String, String> myMap = new HashMap<String, String>();
8
9      public MapTester()
10     {
11
12     }
13
14     public void enterNumber(String name, String number)
15     {
```

```
16          myMap.put(name, number);
17      }
18
19      public String lookupNumber(String name)
20      {
21          return myMap.get(name);
22      }
23
24      public boolean checkForKey(String key)
25      {
26          return myMap.containsKey(key);
27      }
28  }
```

5.27 If a map is holding a key and a new pair is added with the same key, the previous value is overwritten by the new value. See the follwing example.

```
1  // add a pair for the key "A" with value "a"
2  myMap.put("A", "a");
3
4  // add a pair for the key "A" with value "b"
5  myMap.put("A", "b");
6
7  // lookup the value for the key "A"
8  System.out.println(myMap.get("A"))
```

The output will be `"b"` and there is only one pair for the key `"A"`.

5.28 If you put two different keys to a map there will be two keys in this map. That's it, it is the usual usage.

5.29 If I want to know if there is already a key stored contained in a map I have to use the `containsKey(Object key)` method.

```
1  // add a pair for the key "A" with value "a"
2  myMap.put("A", "a");
3
4  // add a pair for the key "A" with value "b"
5  myMap.put("A", "b");
6
7  // check for the key "A"
8  if(myMap.contains("A")){
9      System.out.println("Key is already there!");
10 }
11 else{
12     System.out.println("Key isn't set yet!");
13 }
```

5.30 I don't get the question.

Exercise 2

For what kind of lookups are maps the ideal collections?
Maps are ideal for paired information like a phonebook or similar.

5.5.2 Chapter 5.8 - Dividing Strings

Exercise 3

Solve the exercises 5.35 and 5.37

5.35 The method `split()` is used to separate a string into different strings. The method is taking one parameter which is defining the separator. See the following example.

```
String myString = "Hello. My Name is Foo. I live in Bar."

String[] mySubStrings = myString.split(".");
```

This example above will create a new array of String with size 2 and the strings `"Hello"`, `"My Name is Foo"`, `"I live in Bar"`.

5.36 If you want to split a string at ":" then give that as parameter to the split method.

5.37 If you split a string in pieces and return them to an HashSet, then you will have every string only once in that HashSet. If you return it to an ArrayList you'll get the whole string there and it's indexed, so you can only search by iteration.

5.5.3 Chapter 5.10 - Writing class documentation

Exercise 4

Solve the exercises 5.46 to 5.48

5.46 To generate documentation for Java code you can use the command line tool `javadoc`. If we want to document a single file (class), we can run the following command

```
$ cd /path/to/source
$ javadoc -d myDocumentation File.java
```

This command will make a new directory named myDocumentation and there it places a bunch of HTML files including the documentation for the given file. If you want to document all Java files in that directory, you can change the command to

```
$ javadoc -d myDocumentation *.java
```

5.47 Some javodoc key symbols from the TechSupport project are

- @author
- @version
- @return

Not all of these key symbols do influence the formatting of the documentation but `@return` does (see the documentation for details).

5.48 There is a nice table for all tags (key symbols) on wikipedia, see `http://de.wikipedia.org/w/index.php?title=Javadoc&oldid=124140284`

Exercise 5

Comment one of your own projects with javadoc.

../workspace/Snippets/src/recursions/Main.java

```java
package recursions;

/**
 * This is a test project to write a recursive method.
 * @author ninux
 * @version 1.0
 */
public class Main
{
    /**
     * The main method is defining a fixed array of type integer
     * with the values {1,2,3,4,5}. This array will be printed
     * reverse with the method reverse
     * @param args
     */
    public static void main(String[] args)
    {
        int[] myRow = new int[]{1,2,3,4,5};
        reverse(myRow, (myRow.length-1));
    }

    /**
     * This method is taking an array and prints it in reverse order.
     * The index specifies the first element of the array that shall
     * be printed. The common value for the index is (array.length - 1).
     * @param array
     * @param index
     */
    public static void reverse(int[] array, int index){
        System.out.println(array[index]);
        if(index > 0){
            reverse(array, index-1);
        }
    }
}
```

5.5.4 Chapter 5.13 - Class variables and constants

Exercise 6

Solve the exercise 5.68

- A public variable that is used to measure tolerance, with the value 0.001.

```java
        public static final double TOLERANCE = 0.001;
```

- A private variable that is used to indicate a pass mark, with the integer value of 40.

```
1        private static final int PASSMARK = 40;
```

- A public character variable that is used to indicate that the help command is 'h'.

```
1        public static final char HELP = 'h';
```

5.6 Selfstudy-Questions ALG2

Exercise 7

The following method shall be called with a actual parameter. Durign the execution the maximum height of the call-stack is 5. So how many times was the method coolMethod() called?

```
1 public int coolMethod(int n)
2 {
3     if(n >= 2){
4         return (coolMethod(n-1) + coolMethod(n-2));
5     }
6     else{
7         return n;
8     }
9 }
```

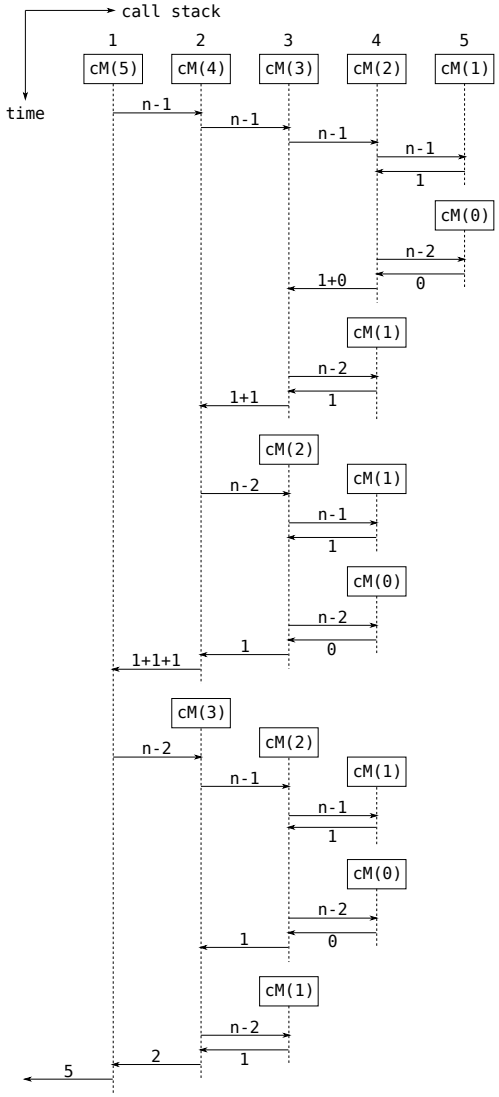

Figure 2: Graphical explanation of call stack rising.

Exercise 8

Implement a recursive algorithm for the calculation of the faculty (see presentation ALG2, page 9). Debug you method for the faculty of 4.

../workspace/Snippets/src/faculty/Main.java

```
1  package faculty;
2
3  public class Main
4  {
5      public static void main(String[] args)
6      {
7          System.out.println(fac(6));
8      }
9
10     public static int fac(int n)
```

```
11      {
12          if(n <=1){
13              return 1;
14          }
15          else{
16              return (n*fac(n-1));
17          }
18      }
19  }
```

Exercise 9

Implement a recursive method to print an array forwards and backwards (see presentation ALG2, pages 29 and 30).

../workspace/Snippets/src/reverse/Main.java

```
1  package reverse;
2
3  public class Main
4  {
5      public static void main(String[] args)
6      {
7          int[] myArray = new int[]{0,1,2,3,4,5,6,7,8,9};
8          printBackAndForth(myArray, myArray.length, 0);
9      }
10
11     public static void printBackAndForth(int[] array, int length, int index)
12     {
13         System.out.println("Forward: ");
14         printForth(array, length, index);
15         System.out.println("\nBackwards: ");
16         printBack(array, length, index);
17     }
18
19     public static void printForth(int[] array, int lenght, int index)
20     {
21         System.out.print(array[index] + ", ");
22
23         if(index < (lenght-1)){
24             printForth(array, lenght, index+1);
25         }
26     }
27
28     public static void printBack(int[] array, int length, int index)
29     {
30         if(index == array.length){
31             return;
32         }
33         else{
34             printBack(array, length, index+1);
35             System.out.print(array[index] + ", ");
36         }
37     }
```

```
38 }
```

Exercise 10

Implement a recursive method that is calculating the sum of all integer values of an array. Make a documentation with javadoc.

../workspace/Snippets/src/sum/Main.java

```
1  package sum;
2
3  public class Main
4  {
5      public static void main(String[] args)
6      {
7          int[] myArray = new int[]{0,1,2,3,4,5,6,7,8,9};
8          System.out.println(sum(myArray, myArray.length, 0));
9      }
10
11     public static int sum(int[] array, int length, int index)
12     {
13
14         if(index < (length-1)){
15             return array[index] + sum(array, length, index+1);
16         }
17         else{
18             return array[index];
19         }
20     }
21 }
```

Additional Exercise

Think about a algorithm that can perform a permutation on an array. Lets say for the sequence 1,2,3,4.

The idea is that we have to use a tree. The tree shows every possible path that can be used for the rearrangement of the numbers. If we start with the number 1, what are the possible paths that we can go trough? See the graphical tree in picture 3 to get an idea of it.

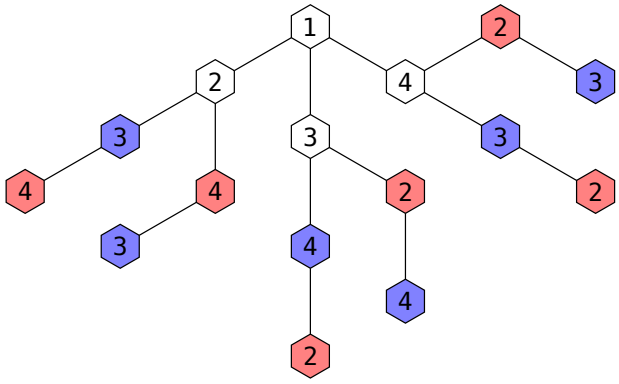

Figure 3: Tree for permutation of 1,2,3,4 beginning with 1.

So we see, that the tricky part is at the end because the last two segments are turned around while the other are just incremented.

5.7 Team-Exercise

5.7.1 Exercise 1

Solve the exercise 5.71

../workspace/StarWars/src/star/NameGenerator.java

```
1  package star;
2
3  public class NameGenerator
4  {
5
6      public NameGenerator()
7      {
8
9      }
10
11     /**
12      * This method returns a name that is created by the
13      * Star Wars algorithm.
14      * @param yourLastName
15      * @param yourFirstName
16      * @param yourMothersMaidenName
17      * @param yourHomeTownName
18      * @return
19      */
20     public String generateStarWarsName( String yourLastName,
21                                         String yourFirstName,
22                                         String yourMothersMaidenName,
23                                         String yourHomeTownName)
24     {
25         String newFirstName;
26         String newLastName;
27
28         /**
29          * The new first name is built by the first three letters
30          * of the last name and the first two letters of the
31          * first name.
32          */
33         newFirstName = yourLastName.substring(0,3) +
34                 yourFirstName.substring(0,2);
35
36         /**
37          * The new last name is built by the first two letters of
38          * the mothers maiden name and the first three letters of
39          * the home town.
40          */
41         newLastName = yourMothersMaidenName.substring(0, 2) +
42                 yourHomeTownName.substring(0,3);
43
44         /**
```

```
45            * Return the new names as full name (first and last name)
46            * separated by a whitespace.
47            */
48           return newFirstName + " " + newLastName;
49       }
50
51   }
```

../workspace/StarWars/src/star/Main.java

```
1  package star;
2
3  public class Main
4  {
5      public static void main(String[] args)
6      {
7          NameGenerator ng = new NameGenerator();
8
9          System.out.println(ng.generateStarWarsName("Nino", "Ninux",
               "Fooman", "Bartown"));
10     }
11 }
```

6 Well-behaved objects

6.1 Selfstudy-Questions OOP8

6.1.1 Chapter 7.1 to 7.3 - Unit testing with BlueJ

Exercise 1

Solve the exercises 7.1 to 7.11

7.1 - 7.9 Solved by testing the source code.

7.10 The source has 2 bugs. These are fixed with the following code:

Bugfix 1 in SalesItem.java

```
1     private boolean ratingInvalid(int rating)
2     {
3         return rating <= 0 || rating > 5;
4     }
```

Bugfix 2 in SalesItem.java

```
1     public Comment findMostHelpfulComment()
2     {
3         Iterator<Comment> it = comments.iterator();
4         if(comments.size() > 0){
5             Comment best = it.next();
6             while(it.hasNext())
7             {
8                 Comment current = it.next();
9                 if(current.getVoteCount() > best.getVoteCount()) {
10                    best = current;
11                }
12            }
13            return best;
14        }else{
15            return new Comment("System", "no comment yet", 3);
16        }
17    }
```

6.1.2 Chapter 7.4 - Test automation

Exercise 2

Solve the exercises 7.12 to 7.19

7.15

../workspace/OnlineShopJUnit/src/shop/SalesItemTest.java

```
1     /**
2      * Test that the same user can't give multiple comments on the same
          product.
3      */
4     @Test
5     public void testDoubleComment()
```

```
6       {
7           SalesItem salesItem1 = new SalesItem("test name", 1000);
8           assertEquals(true, salesItem1.addComment("Max Power", "a", 1));
9           assertEquals(false, salesItem1.addComment("Max Power", "b", 5));
10      }
```

7.16

../workspace/OnlineShopJUnit/src/shop/SalesItemTest.java

```
1       /**
2        * Test that a rating out of the specified range isn't possible.
3        */
4       @Test
5       public void testOutOfBoundaryRating()
6       {
7           SalesItem salesItem1 = new SalesItem("test name", 1000);
8           assertEquals(false, salesItem1.addComment("Max Power", "a", 0));
9           assertEquals(false, salesItem1.addComment("Homer Jay", "b", 6));
10      }
```

7.18

../workspace/OnlineShopJUnit/src/shop/CommentTest.java

```
1       /**
2        * Test that the given information on a comment is stored
3        * correctly.
4        */
5       @Test
6       public void testCommentCreation()
7       {
8           Comment comment1 = new Comment("Max Power", "a", 1);
9           assertEquals("Max Power", comment1.getAuthor());
10          assertEquals(1, comment1.getRating());
11      }
```

6.1.3 Chapter 7.5 to 7.6 - Commenting and style

Exercise 3

Solve the exercises 7.21 to 7.23

6.1.4 Chapter 7.7 - Manual walkthroughs

Exercise 4

Solve the exercises 7.24 to 7.27

6.1.5 Chapter 7.8 - Print statements

Exercise 5

Solve the exercises 7.30 to 7.33

6.1.6 Chapter 7.9 - Debuggers

Exercise 6

Solve the exercise 7.34

6.1.7 Chapter 7.10 - Choosing a debugging strategy

Exercise 7

Solve the exercise 7.36

6.1.8 Chapter 7.11 - Putting the techniques into practice

Exercise 8 (optional)

Solve the exercise 7.37

Exercise 9

Write a test specification for a program that calculates the volume V of a cuboid with the sides a,b,c. Use the following table.

Case	Case No.	Input	Output	Passed
normal	1	$0 < a, b, c <$ `INT_MAX_VALUE`	> 0	yes
normal	2	$0 \leq a, b, c <$ `INT_MAX_VALUE`	≥ 0	yes
normal	3			
acceptable	4			
acceptable	5			
acceptable	6			
illegal	7			
illegal	8			
illegal	9			

Table 1: Test specification for volume calculation program.

6.2 Selfstudy-Questions ALG3

Exercise 10

Implement a pseudo code for the "towers of hanoi" in Java.

../workspace/HanoiTowers/src/towers/Main.java

```
1  package towers;
2
3  public class Main{
4      public static void main(String[] args){
5          moveDisk("A", "B", "C", 3);
6      }
7
8      public static void moveDisk(String from, String via, String to, int n){
9          if(n==1){
10             System.out.println("Move Disk from " + from + " to " + to);
11         }else{
12             moveDisk(from, to, via, n-1);
```

```
13              moveDisk(from, "", to, 1);
14              moveDisk(via, from, to, n-1);
15          }
16      }
17  }
```

Exercise 11

Observe the open methods that are frozen on the stack. Use the Call Graph from the BlueJ Debugger for observation.

7 Designing classes

7.1 Selfstudy-Questions OOP9

7.1.1 Chapter 6.1 to 6.4 - Code duplication

Exercise 1

Solve the exercises 6.1, 6.2, 6.4 and 6.5

Exercise 2

What is coupling and how should it be?
Coupling discribes iterconnectedness of classes. This means, that the coupling is a description for the interface of classes or the connection between classes. A good source code has a loose or weak coupling which means that the implementation of a class can be changed easily without effecting the application.

Exercise 3

What is cohesion and how should it be?
Cohesion is a term that describes how well a unit (package, class, method) maps to a logical task or entity. Ideally, one unit of code should be responsible for one cohesive task (one code – one job).

Exercise 4

What's the problem with code duplication?
The problem at code duplication is that the maintenance of an application is going to be ineffective and buggy. Especially the danger of changing not all of the duplicates can lead to painful bugs that are hard to find. After all code duplication is also just a sign for bad design, so it has to be avoided.

Exercise 5

Code duplication is a symptom for what? Bad coupling or bad cohesion?
Code duplication is usually a sign of bad cohesion because one job should be done by one code.

7.1.2 Chapter 6.5 to 6.11 - Cohesion

Exercise 6

Solve the exercises 6.6 to 6.8, 6.11, 6.14, 6.16 and 6.17

Exercise 7

The Room class is now managing neighboring rooms with a HashMap. To do so the program had to be changed at a lot of different places. Is this an evidence of strong or weak coupling?
This is a clear evidence of tight or strong coupling.

Exercise 8

What is information hiding or encapsulation and what's the effect of this fundamental principle?
Information hiding or encapsulation is a fundamental priciple of good class design. In short it says that a user of a class should only know that much that is needed to use it. This means a class should only provide information about what it does and not how it does that. The effect of this priciple is that we can change the implementation of something without effecting the rest of the application, so it supports automatically a weak or loose coupling.

Exercise 9

What's the meaning of the concept "localizing change"?
The concept of "localizing change" says that it should be clear where to make changes by improving the code. Ideally, only a single class needs to be changed to make a modification. This is primarily achieved by following the general design rules such as responsibility-driven design, loose coupling and strong cohesion.

Exercise 10

What's the difference between explicit and implicit coupling?
Explicit coupling is a coupling that is obvious, not just for a human reader of the code but also for the compiler. Implicit coupling is the worst kind of a strong coupling because it is not obvious, typically neither for a human reader nor the compiler. This is often a symptom of not following the rule of responsibility-driven design.

Exercise 11

Do cohesive methods make also sense?
Of course they make sense. It's just an other level of code base but the rule is the same either it's a method, class, package or application. The cohesion should be as strong as possible in the following manner:

- method – very strong
- class – less stronger than a method
- package – less stronger than a class
- application – less stronger than a package

Exercise 12

What are the two most powerful benefits of strong cohesion?
The two most powerful benefits of strong cohesion are

- readability – easy to understand and to maintain
- reuse – one particular job can be used more than once

7.1.3 Chapter 6.12 to 6.14 - Design guidelines

Exercise 13

Solve the exercise 6.27

Exercise 14

What's the reason for refactoring?
The reasons for refactoring are mostly changed functionality and requirements. The refactoring itself is a process of rethinking and redesigning existing classes and methods.

Exercise 15

Describe the method or the steps at a refactoring.
The refactoring follows usually the following steps:

1. Change the internal structure but don't add new functionality to the code. After the restructuration everything should work as before. Therefore the previous test should be ran again.
2. Since the restructuration is done and the tests gone well, the new functionalities can be implemented. Of course the tests should be ran again to prove the operation of the new developed code.

Exercise 16

At which point is a method too long?
A method is too long if the cohesion is not strong. In such a case we should consider to make helper methods to outsource details from a method. An other unwritten rule is to check the indentations. If the code has an indentation degree far more than three, you should probably change your code.

Simplified one could say "a method is too long if it does more than one logical task".

Exercise 17

At which point is a class too complex?
A class is to complex if there are things that don't belong there following the responsibility-driven design rule. This leads to a creation of a new class or making the changes into an other class.

Simplified one could say "a class is too complex if it represents more than one logical entity".

7.2 Selfstudy-Questions ALG4

Exercise 18

Illustrate the difference between satble and instabe sorting with a simple example.
Stable sorting means that the order of same keys is preserved in the result and vice versa. See the following example:

array	$[r_0]$	$[r_1]$	$[r_2]$	$[r_3]$	$[r_4]$
unsorted	1	12_1	4	12_2	3
stable sorted	1	3	4	12_1	12_2
instable sorted	1	3	4	12_2	12_1

Table 2: Comparison of stable and unstable sorting.

Exercise 19

What's the effort for simple or direct sorting algorithms?
Simple or direct sorting algorithms have an effort of $O(n^2)$

Exercise 20

What's the effort for higher sorting algorithms?
Higher sorting algorithms have an effort of $O(n \cdot log(n))$

Exercise 21

Enumerate three simple or direct sorting algorithms.

Name	type	O
Insertion sort	stable	$O(n^2)$
Selection sort	instable	$O(n^2)$
Shellsort	instable	$O(n^2)$
Bubble sort	stable	$O(n^2)$

Table 3: Basic sorting algorithms

Exercise 22

Which of the sorting algorithms that we talked about in class are instable?
See the table 3.

Exercise 23

At the analysis of srting algorithms it is important to know how to calculate the following sum. Solve the problem for x.

$$1 + 2 + 3 + \cdots + n = \sum_{i=1}^{n} i = x$$

This is a limit of sequence problem which has the form

$$S_n = \sum_{k=1}^{n} a_k = a_1 + a_2 + a_3 + \cdots + a_n$$

This can be either geometrical or arithmetical. In this case it's an arithmetical one. So it has the rule

$$S_n = n \cdot \left(a_1 + d \cdot \frac{n-1}{2} \right) = n \cdot \frac{a_1 + a_n}{2}$$

Substituting with the given values we get the solution

$$S_n = n \cdot \frac{1+n}{2}$$

The following examples shall give the prove

$$\begin{array}{ll} \sum_{i=1}^{n=1} & = 1 \\ \sum_{i=1}^{n=2} & = 1 + 2 = 3 \\ \sum_{i=1}^{n=3} & = 1 + 2 + 3 = 6 \\ \sum_{i=1}^{n=4} & = 1 + 2 + 3 + 4 = 10 \end{array}$$

So the solution of this problem is

$$\sum_{i=1}^{n} i = \lim_{n \to k} n \cdot \frac{1+n}{2} = \frac{1}{2} n(1+n)$$

7.2.1 Optional Exercises about chapter 6.13 (p. 226-231)

Exercise 24

What's the difference between a class type (with the keyword class) and a enumeration type (with the keyword enum)?

Exercise 25

Every enumeration type knows a method values(). What's the return value of that method?

8 Improving structure with inheritance

8.1 Selfstudy-Questions OOP10

8.1.1 Chapter 8.1

Exercise 1

Solve the exercise 8.1

8.1.2 Chapter 8.3

Exercise 2

Solve the exercise 8.3

8.1.3 Chapter 8.4

Exercise 3

Solve the exercises 8.4 and 8.7

8.1.4 Chapter 8.5

Exercise 4

Solve the exercise 8.8

8.1.5 Chapter 8.6

Exercise 5

Solve the exercise 8.9

8.1.6 Chapter 8.7

Exercise 6

Solve the exercises 8.12 and 8.14

8.1.7 Chapter 8.11

Exercise 7

Solve the exercise 8.18

8.2 Selfstudy-Questions ALG5

Exercise 8

Sort the following sequence with the classic Quicksort algorithm (according to the implementation in the class Quicksort.java on ILIAS). Write down the steps and results in between.

Step	$[r_0]$	$[r_1]$	$[r_2]$	$[r_3]$	$[r_4]$	$[r_5]$	$[r_6]$	$[r_7]$	$[r_8]$
Initial sequence	8	2	4	9	5	3	1	7	6

Table 4: Quicksort example

Exercise 9

Check your results form exercise 8 by implementing a console output for the given algorithm.

Exercise 10

When would you prefer an other sorting algorithm over the Quicksort algorithm? Define two different scenarios and explain your answer.

Exercise 11

Sort the following sequence with the Mergesort algorithm (according to the implementation in Java, see Page 36). Write down the steps and results in between.

Step	$[r_0]$	$[r_1]$	$[r_2]$	$[r_3]$	$[r_4]$	$[r_5]$	$[r_6]$	$[r_7]$	$[r_8]$
Initial sequence	8	2	4	9	5	3	1	7	6

Table 5: Mergesort example

Glossary

A

abstraction

Abstraction describes the ability to ignore details and focus attention on a higher level of a problem. As an example think about an car as a Parking-Boy. You would ignore how many seats the cas has, but not how big it is, because it's relevant for your task. 75

access modifier

An access modifier defines the visibility of the declared field, constructor or method. There are only four of them in Java: `private` (visible only from inside the same class), nothing specified aka. default (visible inside the same package), `protected` (visible form inside the same package and subclasses) and `public` (visible from everywhere). 75, 77

accessor

An accessor or accessor method is a method that provides access to information about an object's state (get-methods). 75

agile

Agile or agile software development is a methodology in software development. In short it says that only the very next step in the project is planned. 75

array

An array is a special type of collection that can store a fixed number of items. These items have to be of the same data type. 75

assignment

An assignment (statement) is a directive to assign a value into a variable, for example `speed = newSpeed;` is an assignment. 75

B

blackbox testing

Blackbox testing is a testmethod in software development. Using this method, the test is only observing the in- and output of the software that is been tested. The inner occurances of the tested software are not observed (see whitebox testing for inner obervation). 75, 79

body

A body is a part of an method. It is the part that is bordered by the curly braces. The whole content between these braces is called body (see header for contrast). 75

boolean expressions

A boolean expression is an expression that has only two possible values: `true` or `false`. They are often controlling conditional statements. For example an `if(a<b)` can only return a `true` or `false`. 75

C

class

A class describes the kind of an object. This is done by giving instance variables and methods. The objects represents individual instatiations of the class. 75

class variable

In Java a class can also have a field in contrast to fields that belong to each and every object that is created. Such a field is called class variable or static variable and exists only once for all instances of that class. 75, 79

code coverage

Code coverage is a testmethod in software development. Using this method the testspecification is made such that every possible path in the code is taken at least once. 75

code duplication

If the same or logically similar code is placed on differents places in a source code, we say that there is code duplication. It is a sign of bad design (especially for bad cohesion) and has to be avoided since there are also issues on maintenance and bug appearances. 75

code review

Code review or code inspection is a systematic examination of source code. Reviews are done in various forms such as pair programming, informal walkthroughs, and formal inspections. Usually a code review is done by a person that hadn't developed the code. This is rising the chance to find overlooked mistakes in the development phase of the code. 75

cohesion

Cohesion describes the internal mapping as a logical entity (or task). If a class for example

is called highly cohesive, this means that the internals are responsible for a well defined task or entity (one code – one job). 75

collection

A collection can store an arbitrary number of other objects. Common variants for collections in Java are the `ArrayList`-Objects and arrays. 75

conditional statement

A conditional statement takes one of two possible actions based upon the result of a test. For example `if(a<b)... else ...` is a typical conditional statement. 75

constructor

A constructor is a special method in a class which is responsible to initialize objects properly. In difference to usual methods it has no return value and is only used once. 75

coupling

Coupling describes the interconnectedness of classes. Classes that have a lot of dependencies to other classes are strong coupled. Good classes have a weak or loose coupling, which means that they have no or few dependencies to other classes. Classes with a weak coupling for example do communicated only over well defined and small interfaces.. 75

D

debugger

A debugger is a software tool used by programmers to do a examination of their own code. It allows usually a step by step examination with and without breakpoints and supports the view on variables and other data during the procedure. Debuggers can be used standalone or inplemneted in a so called IDE (Itegrated Development Environment) like eclipse, NetBeans or BlueJ. 75

documentation

Documentation in programming is very important and nobody can achieve success in programming without a good use of documentation. In Java there are some conventions on documentation but in general the following rule is the most important: Write documentation (comments) so that you explain what something does but not how it does it. This is an important aspect of desing pattern and strategy (see information hiding). Java has a tool named `javadoc` that is converting source comments into a HTML formatted documentation. 75

E

encapsulation

Encapsulation is used as a synonym for information hiding. Its guidline says that only information about what a class can do should be visible to the outside, not about how it does it. This leads to a better design and reduces the coupling. 75, 77

error

An error in software can be of different kind, in general there are three types: Syntax error (spelling), semantic error (meaning), logic error (operational). An error is often called a bug. 75

F

field

Fields store data for an object. Fields are also known as instance or member variables. 75

H

header

A header is a part of a method. It is the part that is not only including the signature but the whole definition. Example: `public int getAge(String name)` is the header whereas `getAge(String )` is the signature. 75

I

immutability

In Java the expression "immutable" is mostly used to describe an object. It sais that the objects state or contents cannot be changed once it's created. A common example in Java are String object which are always immutable. So in short this means that a immutable object is a unchangable or fixed object. 75

implementation

An implementation is a source code that is defining something like a class or method.

Takling about methods you could say something like "this is the implementation of the method XY" pointing to a source code. The implementation is a recipe how something is actually done (recipe is a good analogy to source code). 75

information hiding

Information hiding is a principle that states that internal details of an implementation should be hidden from the user of it. This shall ensure better modularization and support abstraction. In Java this is done with the use of an access modifier like `private` or `protected`. A synonym for information hiding is encapsulation. 75, 76

instance

An instance is a realisation of a class to a real object, so instance is a synonym to object. 75

integration test

The integration test is a testmethod in software development. Using this testmethod multiple software components are tested together (for example multiple calsses form a package). Other testmethods are the unit test and the system test. 75, 79

iterface

An interface in Java is used to agree or declare signatures of methods that are shared over different classes. The interface is declaring which methods exist or have to exist. It can be used without showing the implementation. 75

iterator

An iterator is an object that provides functionality to iterate over all elements of a collection. 75

L

library

In Java a libraray is a importable collection of classes. These classes are arranged in packages. A known library is the so called Java standard class library, which contains very useful classes for almost all Java programs. 75

library documentation

The Java class library documentation shows details about all classes in the library. Using this documentation is essential in order to make good use of the library classes. 75

lifetime

The lifetime of a variable describes how long the variable continues to exist before it is destroyed. 75

local variable

A local variable is a variable declared and used within a single method. Its scope and lifetime are limited to that specific method they're defined in. A special variant of local variables are actual parameters. 75

loop

A loop is a functionality that is given by the elementary functions of a programming language, like in Java. They are used to repeat a sequence of expressions (a body) for a number of times, coupled to one or more conditions. In Java there are three essential types of loops: The `while`, `do while` and `for` loop. There are also other types of loops like the "foreach" loop. 75

M

map

A map is a collection type. It stores key-value pairs as entries and it stores each individual element (key) at most once. It does not maintain any specific order. A good use for maps is a phonebook, where every key (person, name) is only once in the book. 75

method

A method is a action (function) of a specific class that can be invoked on an object of the given class. Objects usually do something when a method is invoked, so a good keyword to it would be *what*, as most methods are named by a verb. The methods give the objects their own particular and characteristic behavior. 75

modularization

Modularization is the process of dividing a whole into well-defined parts that can be build and examined seperately and that interact in well defined ways. For example a car as a whole entitiy can be divided into modules such as the engine, seats, radio, wheels and so on. 75

mutator

A mutator or mutator method is a method that provides the ability to change fields of

an object. For example `changeSize(int newSize)` is a typical mutator method. 75

N

non-primitive types

Java has eight primitive types (**boolean**, **char**, **byte**, **short**, **int**, **long**, **float**, **double**) and gives the programmer the ability to define own types of a more complex manner. For example a class defines a new type with the name of the class. Variables that have a class as their type can store objects of that class. A popular example of such a type is `String` which in fact is a class. 75

null

null is a reserved word in Java (and many other programming languages) that indicates that a reference is not referencing to something, that it is showing to **null**. In Java it's used to mean "no object" because a refernece variable should point to a object, if it's not so it's containing the reference **null**. Also a filed that has not been explicitly set will contain **null** if it's not defined by an other default value (like `0` for variables of type **int**). 75

O

object

An object is a instance of a class. 75

object reference

Variables of an object type (non-primitive type) always store references to objects. 75

overloading

In Java sources, classes may contain multiple constructors, methods and variables (variable vs. parameter) with the same name. This is called overloading. In Java there is a keyword **this** to specify the variables so that the compiler can differ them plural. 75

P

parameter

Addition information (data) given to a method or object is called parameter. 75

primitive types

The primitive types in Java are the non-object types **boolean**, **char**, **byte**, **short**, **int**, **long**, **float**, **double**. An important characteristic to primitive-types is, that they don't have methods. 75

R

random number

Real or pure random numbers in computer programming are not that easy to implement, since computers operate in a well defined and deterministic way which makes them highly predictable and that way quite the opposite of everything random. A common but not equivalent alternative are so called pseudo-random numbers. These numbers are calculated by special algorithms that try to give random numbers. 75

refactoring

Refactoring is the activity of restructuring existing code to adapt it to changed functionality and requirements. Most important is that the design is improved and not what it does. 75

responsibility-driven design

Responsibility-driven design expresses the idea that each class should be responsible for handling its own data. 75

S

scope

The scope of a variable defines the section of source code from which the variable can be accessed. 75

set

A set is a collection type. It stores each individual element at most once. It does not maintain any specific order. A good use for a set is to collect all used words in a string (sentence, page, book), where you want to have every word only once. 75

signature

The signature of a method is the part that identifies it to the compiler. For example the signature of `public setSpeed(int newSpeed, int newTolerance)` is not the whole head of the method but the name `setSpeed` and the list of parameter-types `int ..., int ...`. 75

state

A object or its status is represented by his state. The state is represented by the values in the fields (instance variables). 75

static variable

See class variable. 75

system test

A system test is a testmethod in software development. Using this testmethod a complete system is tested with all components. Most of these tests are done by blackbox testing. 75, 77, 79

T

type

The type defines the kind of data or value (for example to a parameter, return value (see data types) or a variable. 75

U

unit test

Unit tests are a testmethod in software development. Using this test only a specific software part is tested completely isolated form other software components of the same project (for example a single class is tested). Other testmethods are the integration test and the system test. 75, 77

V

verification

Verification or formal verification is a act of proving or disproving an software with formal methods or mathematics. This is in general very complicated and therefore very uncommon for standard applications, since it is very expensive in all kind of resources.. 75

W

walkthrough

A walkthrough is a examination of souce code done by at least two personen where one is the developer of the source code and leads the examination and the other is a person that was not involved in the development. Usually the code isn't examined into details by a walkthrough. 75

waterfall

Waterfall or waterfall model is a sequential design process or methodology. In programming it is declaring that a project is planed from the very beginning up to the end in sequences. 75

whitebox testing

Whitebox testng is a testmethod in software development. Using this method, the test is observing the in- and output as well as the inner occurances of the software that is been tested (see blackbox testing for onyl in- and output oberservation). 75