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
22      }
23  }
```

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?

Exercise 16

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

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 drwing/sektch. What does it look like, if it's storing an object?

4.1.5 Chapter 4.15 to 4.17 - Summary

Exercise 19

Solve the exercises 4.62 to 4.65

Exercise 20

Solve the exercises 4.66 to 4.68

Exercise 21

What are the pros and cons of Arrays?

Exercise 22

How do you get the length of an Array?

Exercise 23

Solve the exercises 4.69, 4.71, 4.73 and 4.74

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.. 39

accessor

A accessor or accessor method is a method that provides access to information about an object's state (get-methods). 39

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. 39

assignment

An assignment (statement) is a directive to assign a value into a variable, for example `speed = newSpeed;` is an assignment.. 39

B

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). 39

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`**. 39

C

class

A class describes the kind of an object. This is done by giving instance variables and methods. The objects represents individual instatioations of the class. 39

collection

A collection can store an arbitrary number of other objects. Common variants for collections in Java are the `ArrayList`-Objects and arrays. 39

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. 39

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. 39

F

field

Fields store data for an object to use. Fields are also known as instance variables.. 39

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. 39

I

instance

An instance is a realisation of a class to a real object, so instance is a synonym to object. 39

iterator

An iterator is an object that provides functionality to iterate over all elements of a collection. 39

L

lifetime

The lifetime of a variable describes how long the variable continues to exist before it is destroyed. 39

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. 39

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. 39

M

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. 39

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. 39

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. 39

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. 39

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`). 39

O

object

An object is a instance of a class. 39

object reference

Variables of an object type (non-primitive type) always store references to objects. 39

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. 39

P

parameter

Addition information (data) given to a method or object is called parameter. 39

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. 39

S

scope

The scope of a variable defines the section of source code from which the variable can be accessed. 39

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 ...`. 39

state

A object or its status is represented by his state. The state is represented by the values in the fields (instance variables). 39

T

type

The type defines the kind of data or value (for example to a parameter, return value (see data types) or a variable. 39