Object-Oriented Programming with Java - Advanced Course

Exercises

Prof. Dr. Robin Mueller-Bady

1 Introduction

1.1 Setup of the Environment

Setup your working environment. You will need the following software that you can find free available on the internet:

- Java Development Kit
- IDE of choice (e.g. eclipse)

The JDK is available for Microsoft Windows, MacOS and Linux. Most IDEs should also be available for those operating systems. For installation, follow the instruction of the specific products or use the pre-installed environment in the lab.

1.2 The first "Hello World" (IDE)

Aim of this exercise is to print the first "Hello World" text on the screen. Following steps are necessary:

1. Start the IDE and create a new Java project
2. Inside the new project create a new class and give it a name
3. Create an entry point for your program like shown in the lecture
4. Write the line that is necessary to print the "Hello World" text on the screen

1.3 The first "Hello World" (commandline)

To understand how Java works it is useful to work also without IDEs that do most of the organizational stuff. In this exercise you will create a Java program "by hand" from the commandline using the following steps:

1. Start the commandline (e.g. "terminal" or "konsole" in Linux)

2. Navigate to the directory where you want to create the Java program
3. Create a class that is equivalent to the one you created in the exercise before (*Important:* the filename and the class name must be identical, e.g. "Test.java" implies "public class Test")
4. Compile your program with the commandline tool "javac" that gets installed while installing the JDK
5. Run your program with the "java" command

In case javac or java programs are not found examine the environment variables. Google will help you to find matching configurations for your environment.

1.4 Experiment with the IDE

Aim of this exercise is to get familiar with your tools. Experiment with the IDE, toggle options in the "Preferences" window and adjust windows as you like. Under "Help" you will find most likely an offline manual to for most IDEs.

2 Syntax

2.1 Identifier

Create a program that defines several different types of variables, e.g. int, float, etc. Try different identifiers for these variables and see how the compiler reacts.

2.2 Default Value

Create a program (or use the program from exercise 2.1) and leave some variables uninitialized. What happens ? Are you able to print those variables ?

2.3 Operators I

Change the operators in the following program such that the *shortcut version* of the operation is used. Note the last 2 outputs. Explain the difference and the output of line 18 and 19.

```
1  class ArithmeticDemo {
2      public static void main (String[] args){
3          int result = 76 + 3;
4          System.out.println(result);
5
6          result = result - 3;
7          System.out.println(result);
8
9          result = result * 3;
10         System.out.println(result);
11
12         result = result / 3;
13         System.out.println(result);
```

```
14
15          result = result + 1337;
16          result = result % 42;
17          System.out.println(result);
18          System.out.println(++result);
19          System.out.println(result++);
20      }
21  }
```

2.4 Flow Control with if/else

Look at the following program and decide (without executing it) what will be printed on the screen using the following values for the variable "number"

a) 3

b) 0

c) -1

```
1  if (number >= 0)
2      if (number == 0)
3      System.out.println("Cow");
4  else System.out.println("Donkey");
5  System.out.println("Cat");
```

2.5 Flow Control with switch/case

Write a program that has exact the same functionality than the program seen in exercise 2.4 with help of switch/case. Are there any problems?

2.6 Loops

Write a program that counts from 0 to 999 and outputs its status each 50 steps, e.g. "... 100, 150, 200, ..." Use the following loops to accomplish this task:

a) For loop

b) While loop

c) Do-While loop

Hint: Look at the "modulo" operator!

3 Object-Oriented Programming

3.1 Class- vs. Instance Variable

Look at the following class and decide which is the class- and which is the instance variable. When are those variables available for the programmer?

```
1 public class SomeClass {
2     public static int number1 = 1;
3     public int number2 = 2;
4 }
```

3.2 Accessing Class- and Instance Variables

Use the program from exercise 3.1. Create two instances of this class (objects). On one object set all variables to "1", on the other object set all variables to "99".
Output all variables. What do you recognize? Explain!

3.3 Development of a Card Deck

Aim of this exercise is to create a card game. Several classes will be necessary to do that.

3.3.1 Development of Class "Card"

Write a class that symbolizes a playing card in a card deck. Remember that each card has a different suit (spades, hearts, diamonds, clubs) and value (2 to ace).

3.3.2 Development of Class "CardDeck"

Now create a class that aggregates all playing cards in a card deck. Each card should be available exactly once (king of hearts, king of spades, king of diamonds, ...)

3.3.3 Testing your Program

To test the functionality of your card deck you can create a test program. Develop such a test program that creates a card deck and outputs each card inside this deck.

3.4 Find the Error!

Find the error in the following program and explain it. After that find a solution such that the program will run again.

```
1 public class FindTheError {
2     public static void main(String [] args) {
3         Bicycle bike;
4         bike.speedUp(100);
5         bike.applyBrakes(100);
6     }
7 }
```

3.5 Interfaces

Write a class "CeasarCypher,java" that implements the interface "java.lang.CharSequence" and its methods. This class receives a java.lang.String in its constructor and an integer parameter. The integer parameter shifts the input string by the denoted number to the right, such that a String *AB25* results in the String *BC36* for the integer parameter value 1. Whitespaces and symbols must not be changed during the conversion process. An overflow for *Z* or *9* is handled by starting at *A* or *0* respectively.
You can use the following program to test your code but keep in mind that you have to rename the used class to your class name!

```
1 public class CaesarCypherCharSequenceTester {
2     public static void main(String [] args) {
3         CharSequence cs = new SomeNumberCharSequence("Test 26", 1);
4         System.out.println(cs.charAt(1)); // Result: f
5         System.out.println(cs.subSequence(0,6)) // Result: Uftu 37
6     }
7 }
```

3.6 Inheritance

Create a class "Bicycle" with the following attributes and the appropriate getter/setter methods:

- cadence
- speed
- gear

Inherit a class "Tandem" from the created class "Bicycle" and add an attribute that defines the number of seats. Also create the getter/setter methods for the new attribute. Write a constructor that requires the following values and forwards the necessary ones to the super class.

- cadence
- speed
- gear
- seats

Create a test program or an entry point to test your program.

3.7 java.lang.Object

Take a look on the program from exercise 3.6. Which methods from java.lang.Object can be overwritten? Overwrite at least one method of your choice.

3.8 Card Deck with Enumerations

Modify your program from exercise 3.3 such that values and suits are now represented with the help of Java enums.

4 Strings and Collections

4.1 Process Program Parameters

Write a program which takes an arbitrary number of integer values as parameter, sum them up and output them. The program call could, for example, look like follows (from the commandline)

```
$> java program 1 2 3 4 5 // Result -> 15
$> java program 2 2 2 2 2 2 2 2 2 2 980 // Result -> 1000
```

Hint: If you use an IDE, you can configure program parameters that are used for each execution of the program (e.g. in eclipse: "Runtime Configurations" → "Arguments" → "Program Arguments").

4.2 Process Strings

Given a String test

```
String test = "That's it! You people have stood in my way long enough. I'm going
    to clown college!";
```

What is returned by the following function calls of the class java.lang.String? Think about it first, then implement a small test and check it.

a) test.length()

b) test.charAt(5)

c) test.contains("way")

d) test.indexOf("You")

e) test.equals("That's it! You people have stood in my way long enough. I'm going to clown college!")

f) (test == new String("That's it! You people have stood in my way long enough. I'm going to clown college!"))

4.3 Strings Concatenation

Write a program which concatenates the given Strings to one big String including white spaces in between.

- Homer

- no
- function
- beer
- well
- without

4.4 Anagram

Develop a (commandline) program which takes 2 Strings as parameter (method or program). Test if the given String 1 is an anagram of String 2.
Explanation: "An anagram is the result of rearranging the letters of a word or phrase to produce a new word or phrase, using all the original letters exactly once"[1]
Example:

```
# java anagramm Replays Parsley
Yes, it is an anagram
(or Boolean value True)
# java anagramm Replays AnyOtherWord
No, it is not an anagram
(or Boolean value False)
```

Hint: An anagram is not a palindrome!

4.5 Collections - Comparison

Take a deeper look at the 4 core interfaces of the Java Collection classes: List, Queue, Map and Set. See the list of scenarios below. For each scenario, decide which Collection class fits best for the purpose.

- A company saves the names of all employees. Every month an employee will be picked at random for getting a salary increase.
- The same company as above wants to name their products with the help of their employees names. Therefore, a list of all first names is required where each first name occurs exactly once.
- After quite a while, the company decides to take only the most popular first names in the company into account.
- At Christmas time, the company receives a couple of free passes for a local football game from the city council. For distributing them, they require a waiting list.

[1] Source: Wikipedia; http://en.wikipedia.org/wiki/Anagram

4.6 Collections - Operations

a) Write a method that has a List of Strings as parameter. This method appends a unique number to all given Strings in the List and returns it. For Example, a List of "Hello" and "World" is given. The method returns "Hello1" and "World2".

b) Write a program which tests your written method from a). Output the List before and after the method call.

c) Take the written method and write a new one that changes the List "on the fly" but returns nothing (void). How does the List look like before and after the method call? Explain!

Hint: Be careful with concurrent modification of lists.

5 Error Handling

5.1 Try and Finally

Will the following code compile?

```
try {
    // Any code
} finally {
    // More code
}
```

5.2 Exception Types

Which types of Exceptions will the following code catch and which not? Explain.

```
try {
    // Any Code
} catch (Exception e) {
    // More Code
}
```

5.3 Exceptions

What happens if you compile the following code?

```
public class SomeName {
        public static void main(String[] args) {
        String[] quote;
        quote[0] = "Beer: ";
        quote[1] = "The cause of, and solution to, all of life's problems.";
        System.out.println(quote[0] + " " + quote[1]);
    }
}
```

5.4 Writing own Exceptions

Write an own exception (not just catch an existing!). For that use the program from exercise "Strings and Collections" where you summed up the program parameters. Throw your new exception in case the program got less than 10 parameters. Your new Exception should provide a message in which you explain what went wrong so that your user knows how to use your program.

5.5 Catch your own Exception

Improve your program from exercise 5.4 such that it catches your new exception. In case there occurs an exception, print an error message and fill the array with '0' up to a number of 10 numbers.

6 Streams and Files

6.1 Reading from Keyboard

Develop a program which takes input from a keyboard and prints it on *standard out (stdout)*.
Hint: Use the standard input stream *System.in* either with a *InputStreamReader* or take a deeper look at the class *java.util.Scanner*.

6.2 Input of Files

Write a program which reads in files. This program has to count all occurrences of "y" (no matter if upper- or lowercase) inside the file. For testing purposes, one could use e.g. *Faust - Eine Tragödie*[2]

6.3 Output of Objects

Write a program which writes a List of Cards (e.g. from the CardDeck exercise) into a file. In case you do not have the CardDeck exercise, use any other List of Objects or create a new one. After writing into a file, read it in again and check your result.

6.4 Java NIO 2

Traverse your local file system and count all files that have a suffix of .pdf. Output also the name of the file to standard out or alternatively a file. **Warning:** You need at least Java SE 7 for using Java NIO 2!

[2] http://www.gutenberg.org/files/21000/21000-0.txt

7 Threads

7.1 Threads and Processes

Explain the difference between a process and a (Java) thread

7.2 Create Threads

Create a thread which prints "Hello World" 5 times and ends. Start this thread in a main methods and wait for its completion. Afterwards, print a String that the thread finished.

7.3 Create Threads (again)

There exists 2 approaches to create a thread. Create the thread from exercise 7.2 with the opposite method that you did not use in your program yet. (**Hint**: implementation of an Interface or extending a class)

7.4 Concurrent Threads

Given 2 threads and a main method

```
public class Thread1 extends Thread {
    @Override
    public void run() {
        System.out.println("I am thread 1");
    }
}
```

```
public class Thread2 extends Thread {
    @Override
    public void run() {
        System.out.println("I am thread 2");
    }
}
```

```
public static void main(String[] args) {
    Thread t1 = new Thread1();
    Thread t2 = new Thread2();
    t1.start();
    t2.start();
}
```

What does this program output and why?

7.5 Concurrent access of Variables

Write a class (Counter) with an Integer member variable accessed with getter/setter methods (POJO). Write also an increment/decrement method, which increments and decrements the Integer variable by 1. Additionally, create a thread that fulfills following tasks

- It maintains a reference to one object of the class "Counter"

- At thread start, the thread checks 1.000 times if the counter is odd or even:
  - In case it is odd, increment the number
  - In case it is even, decrement the number

Write a main method which starts 3 of the self written threads above. Wait for their "natural ends" and print out the Integer value in "Counter". Start the program multiple times and explain the result!

7.6 Concurrent access of Variables with Synchronized

Extend your program from exercise 7.5 by adding the synchronized keyword to every access of the Integer variable. Experiment with different synchronization methods form the lectures. Which method(s) solve(s) the concurrency problems and why?