Examination in Object Oriented Programming

University of Applied Science Ravensburg-Weingarten
Prof. Dr. M. Zeller

Date, time	3 February 2018, 10:30 – 12:00 (90 min)
Number of pages	13 pages (including title)
Resources	All accepted resources

Study Program	Room
EI	C004
AI, MD	C109

Name: ________________________ Matriculation number: ____________

Reminder:

- Please note name and matriculation number on each sheet.
- If you use additional sheets do not forget to note name and matriculation number on them too.

---

leave blank, please:

Part	1	2	3	4	5	Sum
max.	5	19	9	20	5	58
Points						

Part 1

1.1 (5 Points) Exception Handling

Analyse the program given in section **Exception** of the handout "Programs and JDK-Documentation".

The program defines some exception classes and a main class. Method `foo()` may throw any of the exceptions defined in the program a well as an ArithmeticException an another RunTimeException.

The program should catch any exception. Each exception defined in the program should be caught separately; the ArithmeticException should also be caught separately.

What is a proper sequence of catch clauses?

```
catch ( . . . . .   ex) {
        System.out.println(ex);
```

The catch-clauses should just print out the caught exception. Fill in the catch clauses in a proper sequence:

```
try{
    foo(i);
  }
```

. . . .

. . . .

. . . .

. . . .

. . . .

. . . .

. . . .

. . . .

. . . .

. . . .

Part 2

2.1 (15 Points)

Analyse the program given in section **Constructors** of the handout "Programs and JDK-Documentation". The class `Breeder` is defined in the package `persons`. All other classes of this program are defined in the package `exoop`.

When answering the questions please keep in mind that there might be more or less dotted lines than actually needed (this holds for all questions).

```
package exoop;
import persons.Breeder;
public class InheritanceTest {
    void test() {
        System.out.println("———— 1 ————");
        Plant aPlant = new Plant();
        System.out.println("———— 2 ————");
        aPlant.grow();
        System.out.println("———— 3 ————");
        Flower aFlower = new Flower();
        System.out.println("———— 4 ————");
        aFlower.grow();
        System.out.println("———— 5 ————");
        aFlower.bloom();
        System.out.println("———— 6 ————");
        Rose aRose = new Rose();
        System.out.println("———— 7 ————");
        aRose.grow();
        System.out.println("———— 8 ————");
        aRose.bloom();
        System.out.println("———— 9 ————");
        Breeder aBreeder = new Breeder();
        System.out.println("——— 10 ————");
        Plant rPlant = new Rose();
        System.out.println("——— 11 ————");
        // rPlant.grow();
        // rPlant.bloom();
        // aBreeder.name = "Evers";
        // aBreeder.country = "France";
    }
}
```

What is the output of the program? Fill in the output step by step.

What is the output of the program when line Plant aPlant = **new** Plant(); is executed?

```
------- 1 --------
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

What is the output of the program when line 8 aPlant.grow(); is executed?

```
------- 2 --------
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

What is the output of the program when line 10 Flower aFlower = **new** Flower(); is executed?

```
------- 3 --------
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

What is the output of the program when line 12 aFlower.grow(); is executed?

```
 ------- 4 --------
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

What is the output of the program when line 14 aFlower.bloom(); is executed?

```
------- 5 --------
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

What is the output of the program when line 16 Rose aRose = **new** Rose(); is executed?

```
------- 6 --------
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

What is the output of the program when line 18 aRose.grow(); is executed?

```
------- 7 --------
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

What is the output of the program when line 20 aRose.bloom(); is executed?

```
------- 8 --------
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

What is the output of the program when line 22 Breeder aBreeder = **new** Breeder(); is executed?

```
------- 9 --------
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

What is the output of the program when line 24 Plant rPlant = **new** Rose(); is executed?

```
------ 10 --------
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

2.2 (4 Points)

Now consider the lines 26 – 29. In the following always one of these lines is activated – the others remain commented out.

```
 4  void test() {
 5      System.out.println("——— 1 ———");
 6                  :
 7                  :

26          // rPlant.grow();
27          // rPlant.bloom();
28          // aBreeder.name = "Evers";
29          // aBreeder.country = "France";
30  }
```

Some of these lines may not compile. Indicate which lines are correct and which will be rejected by the compiler. Note: You receive a point for each correct indication; for each wrong indication one point is deducted. If you ommit an indication in a line no point is added or deducted. The minimum score is zero points even if you marked less correct answers than wrong ones.

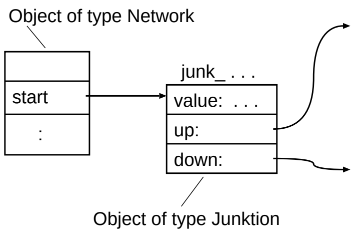
Activated line	correct	Compiler error
rPlant.grow();		
rPlant.bloom();		
aBreeder.name = "Evers";		
aBreeder.country = "France";		

Part 3

Analyse the program given in section **Network** of the handout "Programs and JDK-Documentation".

3.1 (5 Points)

What data structure results in the program after after line 21 junk_1.up=junk_2; is executed? Complete the given sketch by drawing the missing objects, the missing references and the values stored in the member-variable `value`.



3.2 (4 Points)

The method `printNetwork()` of class `Network` traverses any data structure build of objects of type `Junktion`. It calls the method `print()` of each object of type `Junktion`. Complete the `printNetwork()` of class `Network`.

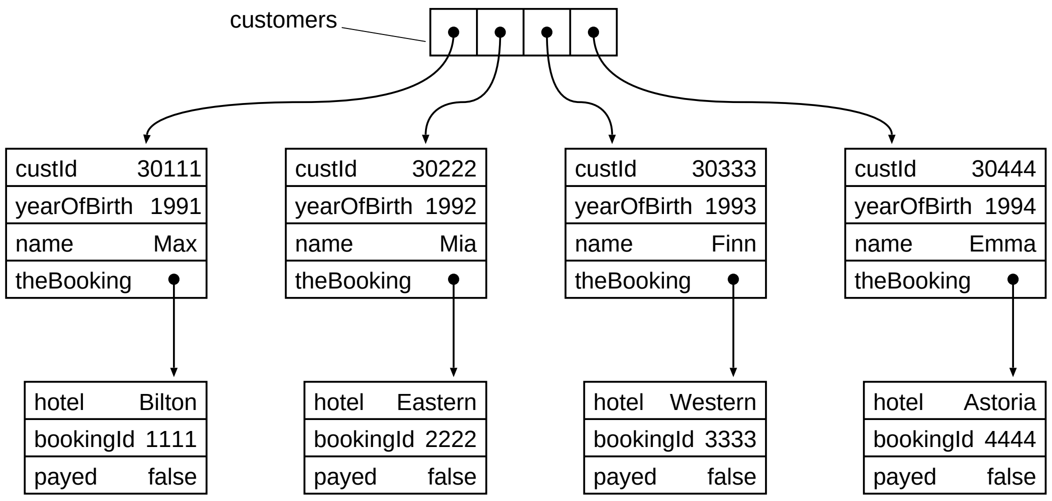
Hint: Implement the method recursively. It first checks if the value of the parameter is `null`. If so, it just returns. If the parameter references an object, it calls the method `print()` of the referenced object. Then it calls itself using the values of member `up` and `down` of the referenced object respectively.

```
void printNetwork(Junktion junk){
    . . . . . . . . . . . . . . .
    . . . . . . . . . . . . . . .
    . . . . . . . . . . . . . . .
    . . . . . . . . . . . . . . .
    . . . . . . . . . . . . . . .
    . . . . . . . . . . . . . . .
    . . . . . . . . . . . . . . .
    . . . . . . . . . . . . . . .
}
```

Part 4

This part refers to section **Collections and IO** of the handout "Programs and JDK-Documentation". A program creates objects of type `Customer`, stores them in an HashMap, prints them to the screen, writes them to a file and reads the data from the file. It uses Objects of type `Booking` as keys in the HashMap. Note: You can work on the sections 4.2 and 4.3 independently.

The method `initCustomers()` creates four objects of type `Customer`, four objects of type `Booking` and stores the customers in the array `cutomers`.



4.1 Print Customer (3 Points)

The class `Customer` is given on page 8, line 101 through 117 of the listing in section **Collections and IO**. The method `print()` of class `Customer` prints the data stored in an object of class `Customer` to the screen (see the given sample output below).

It first prints the member `name`, then `yearOfBirth` and lastly the member `hotel` of the associated Booking.

Two sample output lines of the method `print()` :

```
Customer: Mia, born: 1992, booked in: Eastern
Customer: Finn, born: 1993, booked in: Western
```

Complete the method print() at the ellipsis.

```
void print() {
    System.out.print("Customer: "  . . . . . . .
    . . . . . . . . . . . . . . . . . . . . . . .
    . . . . . . . . . . . . . . . . . . . . . . .
}
```

4.2 Storing Data in an HashMap

4.2.1 (3 Points)

In line 4 of the listing the class `ObjectStorage` defines a member `custMap`. It is a reference to an object of type `HashMap` for storing objects of type `Customer`. The type of the key is `Booking`. The class `Booking` overrides the methods `hashCode()` and `equals()` properly; i. e. the method regards all three members of the class.

Fill in the the proper data type for `customerMap` and provide the code to create the object of type `HashMap`. The initial capacity of the `HashMap` should be 20 .

```
    . . . . . . . . . . . . . . .  custMap =  . . . . . . . . . . . . . . .
```

4.2.2 (3 Points)

The method `storeInHashMap` iterates over the array `customer` and stores each object of type `Customer` in the HashMap. It uses the booking-object associated with the customer-object as key respectively. I. e. the customer-object {30111, 1991, "Max"} ist stored using the key {"Bilton", 1111}.

```
void storeInHashMap() {
    for (Customer aCustomer : customers) {
        Booking key = . . . . . . . . . . . . . .    // get booking-object
        . . . . . . . . . . . . . . . . . . . . .    // store customer ...
    }                                                // ... in HashMap
}
```

4.2.3 (5 Points)

The method `accessHashMap()` retrieves data from the HashMap, stores data and removes data.

Firstly the method retrieves data from the HashMap using the object `aBooking` as key (line 49). Fill in the missing code in line 49.

In line 51 it creates an object of type `Customer`. In line 53 it stores this object using again the object `aBooking` as key. Fill in the missing code in line 53.

What ist the output created by the method call `aCustomer.print();` in line 55?

`Customer: . . . . . . . . . . . . . .`

If activate the lines 57 and 58 (i. e. we remove the comment marks //), what happens?

Does the program still compile? ...

Does it still run? ...

If it still runs, what is the output of line 58? ...

```
46 void accessHashMap() {
47     System.out.println(" ——— accessing HashMap ——— ");
48     Booking aBooking = new Booking("Eastern", 2222);

49     Customer aCustomer =  . . . . . . . . . . . . .
50     aCustomer.print();
51     aCustomer = new Customer(30555, (short) 1995, "Tom");
52     aCustomer.theBooking = aBooking;

53     . . . . . . . . . . . . . .
54     aCustomer = custMap.get(aBooking);
55     aCustomer.print();
56     custMap.remove(aBooking);
57 //  aCustomer = custMap.get(aBooking);
58 //  System.out.println("last access: " + aCustomer);
59 }
```

4.3 File IO

The program writes objects of type `Customer` to an `DataOutputStream` and reads data from an `DataInputStream` in order to restore the written objects.

The methods `getBufferedOutputStream()` and `getBufferedInputStream()` are given.

4.3.1 (3 Points)

The method **void** writeCustomer(String fileName) obtains a `BufferedOutputStream()` and creates a `DataOutputStream` based on this `BufferedOutputStream()`. Then it iterates over the array `customers`. It writes the values of the members `custId`, `yearOfBirth` and `name` of each object of type `Customer` to the `DataOutputStream` `datOS`.

if an `IOException` is thrown inside the try-block the method prints "IO-error while writing" to the screen. If another exception is thrown inside the try-block the method prints "general error while writing" to the screen.

Complete the method writeCustomer().

```
void writeCustomer(String fileName) {
    try (BufferedOutputStream bufs = getBufferedOutputStream(fileName);

            . . . . . . .   datOS =  . . . . . . . . . . . . . ) {
        for (int i = 0; i < customers.length; i = i + 1) {










        }
    } catch . . . . . . . . . . . .









}
```

4.3.2 (3 Points)

The method **void** readCustomer(String fileName) reads data from a `DataInputStream` and creates objects of type `Customer` based on these data. The data hav been written using the method writeCustomer().

The method obtains a `BufferedInputStream()` and creates a `DataInputStream` based on this `BufferedInputStream()`. Then it reads data from the `DataInputStream` in the given for-loop.

It reads the values for the members `custId`, `yearOfBirth` and `name` of objects of type `Customer`. After reading one set of values it creates an object of this type. The newly created object contains the values in its members. The method stores the newly created object in the array `customers`.

If an `IOException` is thrown inside the try-block the method prints "IO-error while reading" to the screen. If another exception is thrown inside the try-block the method prints "general error while reading" to the screen.

Complete the method readCustomer().

```
void readCustomer(String fileName) {
    try (BufferedInputStream bufs = getBufferedInputStream(fileName);
            . . . . . . . . . . . . datIS = . . . . . . . . . . . . . . . . .
        for (int i = 0; i < customers.length; i = i + 1) {









        }
    } catch (IOException ioex) {




}
```

Part 5

Analyse the program given in section **Class Design** of the handout "Programs and JDK-Documentation". All given classes are defined in the same package.

5.1 Class Design

Modify the program in order to simplify the class `TestClass`. The method `createItems()` should create the same number of objects of type `Desk`, `Printer` and `TableLamp` as before. For each of these objects the method `print()` should be called. The sequence of the output lines my be different.

The modified program should use less code (in the method `createItems()`) to produce this output. I can be written using less variables an less loops.

5.2 New class (2 Points)

If you introduce something new define here:

5.3 Changes to classes (2 Points)

How do you change the classes `Desk`, `Printer` and `TableLamp`? Just sketch the changes not the complete classes.

public class Desk

public class Printer

public class TableLamp

5.4 Changes to class TestClass (2 Points)

How do you change the classes `TestClass`? Just sketch the changes not the complete class.

Note: There are be more dotted lines than necessary.

```
public class TestClass {
    int itemCount = 4
    public void createItems() {
          . . . . . . . . . . . . . . . . . .
          . . . . . . . . . . . . . . . . . .
          . . . . . . . . . . . . . . . . . .

        for (int i = 0; i < itemCount; i++) {
            Desk aDesk = new Desk(3 * i);
            Printer aPrinter = new Printer(3 * i + 1);
            TableLamp aLamp = new TableLamp(3 * i + 2);
          . . . . . . . . . . . . . . . . . .
          . . . . . . . . . . . . . . . . . .
          . . . . . . . . . . . . . . . . . .
          . . . . . . . . . . . . . . . . . .
          . . . . . . . . . . . . . . . . . .
          . . . . . . . . . . . . . . . . . .
          . . . . . . . . . . . . . . . . . .


    }
}
```