# Examination in Object Oriented Programming

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

| | |
|---|---|
| Date, time | July 8th, 2014, 10:30 – 12:00 (90 min) |
| Number of pages | 9 pages (including title) |
| Resources | All accepted resources |

| Study Progam | Exam. No. | Room |
|---|---|---|
| EI | 2451 | H061 |
| EI | 2608 | H061 |

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 | Sum |
|---|---|---|---|---|---|
| max. | 5 | 15 | 17 | 13 | 50 |
| Points | | | | | |

# Part 1

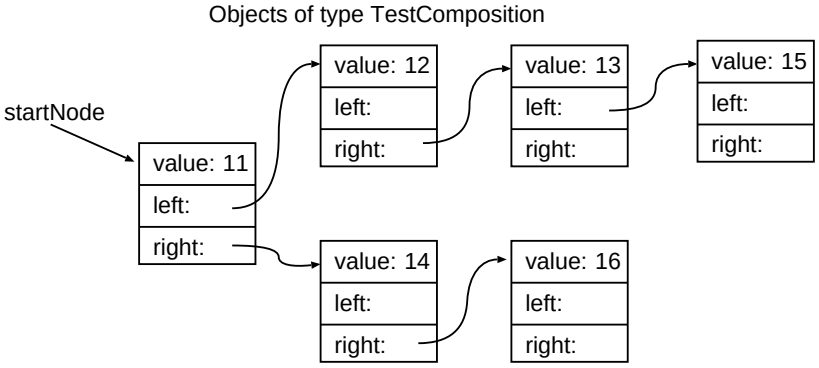
## 1.1 (5 points)

Given the following program, which data structure results in the main-method afte line 18 is executed?

Complete the given sketch by drawing the missing objects, the missing references and the values stored in the member-variable `value`.

```
1  public class TestComposition {
2
3      int value;
4      TestComposition left;
5      TestComposition right;
6
7      TestComposition(int value_) {
8          value = value_;
9      }
10
11     public static void main(String[] args) {
12         TestComposition startNode = new TestComposition(11);
13         startNode.left = new TestComposition(12);
14         TestComposition tmpComp = startNode.left;
15         tmpComp.right = new TestComposition(13);
16         startNode.right = new TestComposition(14);
17         tmpComp = startNode.left.right;
18         tmpComp.left = new TestComposition(15);
19         startNode.right.right = new TestComposition(16);
20         System.out.println(". . . done");
21         return;
22     }
23 }
```

Solution:



Note: The given programs contain some methods which you do not need to analyse. Namely the methods `createRandomXXX()`.

# Part 2

## 2.1 (9 Points)

Analyse the program given in section "Constructors". What is the output of the program?

```
DataObject - id: defaultId, count:  80 , value: 0
AltDataObject - id: defaultId, tmp: 90
DataObject - id: Bob, count:  80 , value: 22
DataObject - id: aSubDO, count:  55 , value: 77 LastDataObject bye bye . . .
```

## 2.2 (6 Points)

The classes `PlainDataObject`, `SubDataObject`, `AltDataObject` and `lastDataObject` belong to the package `polymorphism`. Now we add the class `VeryLastDataObject` given below.

```
1  package collection_io;
2  import polymorphism.SubDataObject;
3
4  public class VeryLastDataObject extends SubDataObject{
5
6      VeryLastDataObject(String name, String msg) {
7          super(33);
8          textId = name;
9          count = -1;
10         value = 11;
11         System.out.print(textId);
12     }
13
14
15 }
```

Does this class definition compile? Indicate which line is legal, which line causes a compile-time-error?

| line | legal | error |
|---|---|---|
| **super**(33); | X | |
| textId = name; | X | |
| count = −1; | | X |
| value = 11; | | X |
| System.out.print(textId); | X | |

# Part 3

This part refers to section "Collection and IO" of the program handout. A program stores objects in an ArrayList, writes these objects to a file and reads the objects from the file.

## 3.1 (2 Points)

Define a member `dataList` in the class `CollectionIO`. This member should be typed as a reference to an `ArrayList` that holds objects of type `SubDataObject` (s. section Program Constructors).

Further complete the Constructor of the class `CollectionIO` so that the member `dataList` actually holds a reference to an `ArrayList`. The `ArrayList` should have an initial capacity of 30.

Complete the method program fragment at the ellipsis.

```
1 public class CollectionIO {
2
3     ArrayList<AltDataObject> doList;
4     int elemCount;
5
6     public CollectionIO(int elemCount_) {
7         doList = new ArrayList<AltDataObject>(30);
8         elemCount = elemCount_;
9     }
```

### 3.2 (3 Points)

The method call `cio.initializeDataList()` creates random objects an stores them in the `ArrayList dataList`. The output listing below shows the values of these objects at a certan run i. e. the output of the first call to `cio.printDOs();`.

The program runs up to line `// - - - 1 - - - .`

Complete the output of the program so far (Note: There might be more doted lines than you will need).

```
 --- List of stored objects ---
AltDataObject - id: Ellen, tmp: 141
AltDataObject - id: Bill, tmp: 130
AltDataObject - id: Charles, tmp: 129
AltDataObject - id: Dan, tmp: 135
 --- List of stored objects ---
AltDataObject - id: First, tmp: 203
AltDataObject - id: Ellen, tmp: 141
AltDataObject - id: Bill, tmp: 130
AltDataObject - id: Charles, tmp: 129
AltDataObject - id: Dan, tmp: 135
 --- List of stored objects ---
AltDataObject - id: First, tmp: 203
AltDataObject - id: Bill, tmp: 130
AltDataObject - id: Charles, tmp: 129
AltDataObject - id: Dan, tmp: 135
```

### 3.3 (1 Point)

What happens during the call `cio.insertDOs(10)` in line 24?

The method call `dataList.add(pos, newDo);` inside of `cio.insertDOs()` throws an exception since it is not allowed to insert an element at a position greater than `dataList.size()`.

### 3.4 (5 Points)

The progam iterates through the `dataList` and saves some data of each object through an `DataOutputStream` to a file (method call: cio.saveDOsToFile("doStore.dat");). The method should work for an arbitry number of objects inside the `dataList`.

It first saves the number of objects to write. Inside the loop it writes the value of the member `value` and the value of the member `textId` to the output-stream. Complete the method `saveDOsToFile` at the ellipsis.

```
1  public void saveDOsToFile(String dataFileName) {
2      DataOutputStream dos = null;
3      try {
4          BufferedOutputStream bos = getBufferedOutputStream(dataFileName);
5          dos = new DataOutputStream(bos);
6          dos.writeInt(doList.size());
7          for (int i = 0; i < doList.size(); i++) {
8              AltDataObject edo = doList.get(i);
9              dos.writeInt(edo.value);
10             dos.writeUTF(edo.textId);
11         }
12     } catch (IOException ex) {
13         ex.printStackTrace();
14     }
15     try {
16         dos.close();
17     } catch (IOException ex) {
18         ex.printStackTrace();
19     }
20     return;
21 }
```

### 3.5 (6 Points)

Later the progam reads data via a `DataInputStream` from the file and constructs new objects from these data.

First it reads the number of data sets stored in the file. For each data set it reads an integer value and a string. It then creates a new objct of type `AltDataObject` providing the string as parameter to the constructor. It sets the value of member `value` and adds the object to the ArrayList doList.

Complete the method `readDOsFromFile` at the ellipsis.

```
1  public void readDOsFromFile(String dataFileName) {
2          DataInputStream dis = null;
3          try {
4              BufferedInputStream bis = getBufferedInputStream(dataFileName);
5              dis = new DataInputStream(bis);
6              int adoCount = dis.readInt();
7              for (int i = 0; i < adoCount; i++) {
8                  int value_ = dis.readInt();
9                  String textID = dis.readUTF();
10                 AltDataObject ado = new AltDataObject(textID);
11                 ado.value = value_;
12                 doList.add(ado);
13             }
14         } catch (IOException ex) {
15             ex.printStackTrace();
16         }
17         try {
18             dis.close();
19         } catch (IOException ex) {
20             ex.printStackTrace();
21         }
22         return;
23     }
```

# Part 4

## 4.1 (4 points)

Given the program presented in section "Program Array". The program creates objects of type DataObject with random values for the different members and stores these objects in an array . It also fills an integer-array with random values (method call `anArrayTest.fillArrays();` in line 73).

In the lines 77 to 80 the program catches all possible types of exceptions separatly. Note: A built-in `RuntimeException` might be thrown in the program.

Each catch-clause indicates the type of the cought exception:
`System.out.println("cought . . . Exception");`

Provide the four necessary catch-clauses in a sensible squence.

```
catch (ExtendedException eex) {
    System.out.println("cought ExtendedException");
} catch (RareException rex) {
    System.out.println("cought RareException");
} catch (BasicException bex) {
    System.out.println("cought BasicException");
} catch (Exception ex) {
    System.out.println("cought Exception");
}
```

## 4.2 (3 Points)

The lines 69, 70 and 71 produce some output: Complete the output at the ellipsis.

```
testNum[0]: 77
DataObject: id: 999, data:  1,23, name: ModifiedTestDto
myNum:321
```

### 4.3 (3 Points)

Method `void printNumberPairs()` iterates through all values of array `numbers`. If two adjacent values in the array are equal, the method reports the value and the two positions where this value is found in the array.

e.g. `numbers: [9, 14, 12, 12, 1, 7, 19, 6 . . . ]`

results in the following output: `Number: 12 found at: 2 and 3`

Complete the following two methods at the ellipsis:

```
public void printNumberPairs() {
    for (int i = 0; i < numbers.length - 1; i++) {
        if (numbers[i] == numbers[i + 1]) {
            System.out.print("Number: " + numbers[i]);
            System.out.print(" found at: " + i);
            System.out.println(" and " + (i + 1));
        }
    }
    return;
}
```

### 4.4 (3 Points)

The method `print_DOs_IdGreaterValue()` iterates through all objects in the array `dtObjects`. It calls the method `print()` of each object, where the value in member `dataValue` is greater than the value in member `idNumber`.

Complete the following method at the ellipsis.

```
public void print_DOs_IdGreaterValue() {
    for (int i = 0; i < dtObjects.length; i++) {
        if (dtObjects[i].idNumber < dtObjects[i].dataValue) {
            dtObjects[i].print();
        }
    }
    return;
}
```