Name:	Number:

Object Oriented Programming 2015/16

Final Exam
June 17, 2016

Directions (read carefully):

- CLEARLY print your name and ID on every page.
- The exam contains 8 pages divided into 4 parts. Make sure you have a complete exam.
- The point value of each problem is indicated next to the problem and below.
- You have two hours.
- It is wise to skim all the problems before beginning, to best plan your time.
- This is a closed book exam. No notes of any kind are allowed. Do all work in the space provided, using the backs of sheets if necessary.
- **Over the table it should ONLY be this exam, a pen and an ID.**
- Turn off the mobile phone. The use of a mobile phone annuls the exam.

Part	Problem	Description	Page	Marks
I	1 a) b)	UML	2	1.5
II	2 a) b) c) d) e)	Java development	3	3.0
	3 a) b) c) d) e)	Java multiple choice	6	2.5
	4	Java miscellaneous	7	1.0
III	5	XML	8	1.0
IV	6	Swing	8	1.0
Total				**10.0**

Name:	Number:

Part I -- UML (1.5 marks)

1 – Companies consist of departments and offices. Each company has a name. Departments are located in one or more offices. Offices contain an address and one or more departments. One office in a company acts as headquarter. Each department has a name and a manager who is recruited from the set of employees. The employees contain a name and a title, which can only be accessed/updated by getters/setters. The employees are associated with the department they work in. Note: If the company ceases to exist then the offices and departments should also be erased.

a) **[1.0 mark]** Define the UML class diagram for the presented problem.

Name:	Number:

b) [0.5 marks] Define an UML object diagram considering a company with an office and a headquarter. Set all needed attributes/associations with some dummy values.

Part II -- Java (6.5 marks)

2 – Consider a class `Pipe` with two private fields, `diameter` (of type double) and `number` (of type integer), and a public constant field named `TOLERANCE`, defined as `10E-2`. Provide an implementation of class `Pipe`, with two different ways to compare pipes: one compares pipes by their `diameter`, up to the `TOLERANCE` value, and the other compares pipes by the `number` of pipes. Note: The only types and methods you can use from `java` package are:

- `java.lang.Comparable` with method `public int compareTo(T obj)`;
- `java.util.Comparator` with method `public int compare(T obj1, T obj2)`; and
- `java.util.Arrays` with methods:
  - `public static void sort(Object[] a)`; and
  - `static <T> void sort(T[] a, Comparator<? super T> c)`.

a) **[0.5 marks]** Provide an implementation of the `Pipe` class.

b) **[0.75 marks]** Provide the pipe comparison by their `diameter`.

c) **[0.75 marks]** Provide the pipe comparison by `number` of pipes.

d) **[0.5 marks]** Provide a `main` method in a `Main` class where five pipes are built and stored in an array of pipes.

e) **[0.5 marks]** Within the `main` method sort the array of pipes according to the two criteria previously implemented and print the result to the terminal.

Name: Number:

```
public class Pipe implements Comparable<Pipe>{

      private double diameter;
      private int numberOfPipes;
      public static final double TOLERANCE= 10E-15;

      public Pipe(double diameter, int numberOfPipes) {
        this.diameter = diameter;
        this.numberOfPipes = numberOfPipes;
      }

      public double getDiameter() { return diameter; }
      public int getNumberOfPipes() { return numberOfPipes; }

      @Override
      public int compareTo(Pipe other) {
        if (Math.abs(diameter - other.diameter) < TOLERANCE) return 0;
        if (diameter < other.diameter) return -1;
        return 1;
      }

      public String toString() {
        return("Diameter: " + diameter + " number: " + numberOfPipes);
      }
}

public class PipeComparator implements Comparator<Pipe> {

     @Override
     public int compare(Pipe o1, Pipe o2) {
       return o2.getNumberOfPipes()-o1.getNumberOfPipes();
     }

}

public class Main {

     public static void main(String[] args) {

       PipeComparator comp = new PipeComparator();

       Pipe[] pipes = new Pipe[5];
       pipes[0] = new Pipe(0.27, 1);
       pipes[1] = new Pipe(0.15, 3);
       pipes[2] = new Pipe(0.25, 7);
       pipes[3] = new Pipe(0.26, 16);
       pipes[4] = new Pipe(0.23, 4);
       //Sorting pipes in an array (with the Comparable)
       Arrays.sort(pipes);
       System.out.println("*** Array - Sorting with the Comparable");
       for (Pipe p: pipes)
             System.out.println(p);
       System.out.println();
       //Sorting pipes in an array with a Comparator
       Arrays.sort(pipes, comp);
       System.out.println("*** Array - Sorting with the Comparator");
       for (Pipe p: pipes)
             System.out.println(p);

}
```

Name:	Number:

Name:	Number:

3 – Fill the answers of multiple-choice questions in the following table (use only capital letters). If you want to correct your answer scratch out and write the correct answer. Each correct question is marked 0.5 points. A question not answered is marked 0 points, whereas a wrong answer discounts 0.2 points. If you think NONE of the options are correct, write NONE.

Question	a)	b)	c)	d)	e)
Answer	D	A	B	D	D

a) **[0.5 marks]** Let `int x=8`, what is the output of `System.out.println(++x * 3 + "," + x)`?

A. 24,8
B. 24,9
C. 27,8
D. 27,9
E. It does not compile

b) **[0.5 marks]** Which of these is necessary to specify at time of matrix-array initialization?

A. Row
B. Column
C. Both row and column
D. None of the mentioned
E. It depends on the operating system

c) **[0.5 marks]** What is the size of a `char` variable in Java?

A. 1 byte
B. 2 bytes
C. 4 bytes
D. It depends on the compiler settings
E. It depends on the operating system

d) **[0.5 marks]** Let `lli1` and `lli2` be both a `LinkedList<Integer>` with values {0,1,2} and {3,4,5}, respectively. What is the output from the following code?

A. [0,1,2][3,5]
B. [0,1,2][4,5]
C. [0,1][3,5]
D. [0,1][4,5]
E. The code throws an exception

```
Iterator<Integer> it1 = lli1.iterator();
Iterator<Integer> it2 = lli2.iterator();
it1.next(); it1.next(); it1.next();
it2.next();
it1.remove();
it2.remove();
System.out.print(lli1);
System.out.println(lli2);
```

e) **[0.5 marks]** What is the value of x after two objects of type Xpto are built?

A. 0
B. 1
C. 2
D. 3
E. None of the previous

```
public Xpto {
   public static int x=0;
   static { x=1;}
   public Xpto() {x++;}
}
```

4 – [1.0 marks] What are the differences between checked and unchecked exceptions in java?

Checked exceptions (CE):

- In this case the compiler checks the exceptions a method declares to throw. So, a method is obliged to establish a policy for all checked exceptions thrown by its implementation, either by passing the checked exception further up the stack, or by handling it somehow.

- CEs are subclasses of Exception.

- CEs represent conditions that are reasonably expected to occur, typically representing invalid conditions in areas outside the immediate control of the program (invalid user input, etc).

Unchecked exceptions (UE):

- The compiler does not check for these exceptions, and so a method is not obliged to establish a policy for the unchecked exceptions thrown by its implementation (and they almost always do not do so).

- UEs are subclasses of RuntimeException.

- UEs represent conditions that generally reflect errors in the program's logic and cannot be reasonably recovered from at run time (e.g. NullPointerException, ArrayIndexOutOfBound).

Name: Number:

Part III -- XML (1 mark)

5 – [1.0 marks] Consider the problem presented in Part I-1.a). Present a DTD and XML document, with XML elements and attributes, to store all information needed. Use dummy values in the XML.

```
<?xml version="1.0"?>
<!DOCTYPE Company [
<!ELEMENT Company (Office+, Headquarter, Department+)>
<!ATTLIST Company cname NMTOKEN #REQUIRED>
<!ELEMENT Office (Address, Departments)>
<!ELEMENT Headquarter (Address, Departments)>
<!ELEMENT Department (Employee+)>
<!ATTLIST Department name ID #REQUIRED>
<!ELEMENT Address (#PCDATA)>
<!ELEMENT Departments EMPTY>
<!ATTLIST Departments deps IDREFS #IMPLIED>
<!ELEMENT Employee (#PCDATA)>
<!ATTLIST Employee ename NMTOKEN #REQUIRED>
<!ATTLIST Employee manager (true|false) "false">
]>
<Company cname="Galp">
    <Office>
          <Address>Porto</Address>
          <Departments deps="D2"/>
    </Office>
    <Headquarter>
          <Address>Lisboa</Address>
          <Departments deps="D1 D2"/>
    </Headquarter>
    <Department name="D1">
        <Employee ename="Joao_Silva" manager="true">CEO</Employee>
        <Employee ename="Ana_Silva">CTO</Employee>
    </Department>
    <Department name="D2">
        <Employee ename="Joao_Sales" manager="true">RD</Employee>
        <Employee ename="Ana_Sales">RD</Employee>
    </Department>
</Company>
```

Part IV -- Swing (1 mark)

6 – [1.0 marks] How can a GUI component handle its own events?

```
A component can handle its own events by implementing the required event-
listener interface and adding itself as its own event listener.
```