Name:	Number:

Object Oriented Programming 2014/15

Final Exam
July 3, 2015

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**

Version A

Name:	Number:

Part I -- UML (1.5 marks)

1 – Consider an Emergency Room (ER) of a Hospital. It is organized in several sections. There are observation sections (where patients are observed) and surgery sections (where patients undergo a surgery). Each observation section has a number of beds. Surgery sections have one or two surgery rooms. Each surgery room has one or two beds. Each section maintains a list of the staff on duty; staff members are identified by their name and id number. The staff member positions are: Auxiliary, Doctors and Nurses. Doctors and Nurses are also identified by they medical sub-specialty; in addition Doctors also have their prescriber ID. When patients arrive to the ER they are placed in a queue for triage. After triage patients go directly to a section queue. If a bed is available, the patient is removed from the queue and assigned to the free bed.

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 one observation section with two beds, a doctor and a nurse, with one patient waiting in the section queue. Set all needed attributes/associations with some dummy values.

Part II -- Java (6.5 marks)

2 – A ternary min-heap is an ordered tree-based data structure with the following properties: (i) every node has a value; (ii) every node's value is less than all the values in the descendent nodes; (iii) all nodes have at most three descendants (left, middle, and right sub-heaps) that also fulfill the heap conditions. Provide an implementation of a ternary min-heap, named `TriHeap`. Usually heaps are implemented taking profit of arrays; here, for the sake of simplicity, implement it as a tree-based data structure. The `TriHeap` class should be generic with type parameter `V` (for the value). In addition, the type parameter `V` must implement the `Comparable` interface. The `Comparable` is the only type you can use from `java.util` package.

a) **[0.5 marks]** Provide the skeleton of the necessary classes with fields and constructors. For the `TriHeap` class only a no-arg constructor is needed.
b) **[1.0 marks]** Provide an `insert` method that receives a `value` of type `V` and inserts the `value` in the heap; it returns `void`. If the `value` being inserted already exists in the heap then the new `value` is not inserted. When the `value` is placed in a node it must preserve the heap structure. In addition, the depth of the sub-heaps should be balanced, as much as possible.
c) **[0.5 marks]** Provide a `depth` method that returns the depth of the heap (the length of the largest branch).
d) **[0.5 marks]** Provide a `delete` method that receives a `value` of type `V` and removes from the heap that `value`.
e) **[0.5 marks]** Provide a `search` method receives a `value` of type `V` and returns `true` if the `value` belongs to the heap and `false` otherwise. Note that the heap structure should be taken into account to optimize the method.

Name: Number:

```
public class TriHeap<V extends Comparable<V>> {

    Node<V> root;
    public TriHeap() {}

    /* INSERT: */
    public void insert(V value){
    if (root==null) root=new Node<V>(value);
    else {
            V aux;
            if (root.value.compareTo(value)>0){
                aux = root.value;
                root.value=value;
            } else
                aux = value;
                int lNbNodes=root.left.nbNodes(), mNbNodes=root.middle.nbNodes(), rNbNodes=root.right.nbNodes();
                if (lNbNodes<=rNbNodes && lNbNodes<=mNbNodes) root.left.insert(aux);
                else if (mNbNodes<=rNbNodes) root.middle.insert(aux);
                else root.right.insert(aux);
            }
    }

    private int nbNodes(){
            if (root==null) return 0;
            return 1+root.left.nbNodes()+root.middle.nbNodes()+root.right.nbNodes();
    }

    /* DEPTH: */
    public int depth(){
            if (root==null) return 0;
            int lDepth=root.left.depth();
            int mDepth=root.middle.depth();
            int rDepth=root.right.depth();
            return (lDepth>mDepth && lDepth>rDepth ? lDepth+1 : mDepth>rDepth ? mDepth+1 : rDepth+1);
    }

    /* DELETE: delete need not to produce a balanced tree! */
    public void delete(V value){
            if (root==null) return;
            if (root.value.equals(value)){
                if (depth()==1) root=null;
                else {
                        TriHeap<V> lth=root.left, mth=root.middle, rth=root.right;
                        root=null;
                        join(lth);
                        join(mth);
                        join(rth);
                }
            } else if (root.value.compareTo(value)<0) {
                root.left.delete(value);
                root.middle.delete(value);
                root.right.delete(value);
            }
    }
```

Name: Number:

```
    private void join(TriHeap<V> th){
        if (th.root==null) return;
        insert(th.root.value);
        join(th.root.left);
        join(th.root.middle);
        join(th.root.right);
    }

    /* SEARCH: */
    public boolean search(V value){
        if (root==null) return false;
        if (root.value.equals(value)) return true;
        if (root.value.compareTo(value)>0) return false;
        return root.left.search(value) || root.middle.search(value) || root.right.search(value);
    }
}

public class Node<V extends Comparable<V>> {

    TriHeap<V> left;
    TriHeap<V> middle;
    TriHeap<V> right;
    V value;

    public Node(V v) {
        value=v;
        left=new TriHeap<V>();
        middle=new TriHeap<V>();
        right=new TriHeap<V>();
    }
}
```

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	C	A	C	E	D

a) **[0.5 marks]** What is the value of the class field x after building two objects of type X with the no-arg constructor?

```
class X {
      static int x=0;
      static { x+=1; }
      int y=-1;
      { y+=2; }
      public X() { this(2); }
      public X(int y) { x=x+y; }
}
```

A. 3
B. 4
C. 5
D. 6
E. The code does not compile

b) **[0.5 marks]** Consider the following classes and identify, method by method of class Y, which are overload, override or compile-time errors. The methods are ordered in the alternatives (A, B, C, D and E) as q;r;v;t.

```
class X {
      int q(){…}
      Number r(int x) {…}
      Integer v() {…}
      <T extends Number> T t(){…}
}
```

```
class Y extends X {
      long q(){…}
      Long r(int y) {…}
      <T extends Number> T v(){…}
      Integer t() {…}
}
```

A. compile-time error; override; compile-time error; override
B. override; override; override; override
C. override; override; overload; overload
D. compile-time error; override; overload; override
E. compile-time error; override; overload; overload

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

A. 2 bytes
B. 4 bytes
C. 8 bytes
D. It depends on the compiler settings
E. It depends on the operating system

Name: Number:

d) **[0.5 marks]** What is printed to the terminal?

```
class X{
      private int xpto(){return 5;}
      public static int foo() {return 15;}
      public static void main(String[] args){
            X x = new Y();
            Y y = (Y)x;
            System.out.print(x.xpto());
            System.out.print(y.foo());
            System.out.print(((X)y).xpto());
      }
}
class Y extends X{
      public int xpto(){return 10;}
      public static int foo() {return 20;}
}
```

A. `5 20 10`
B. `10 15 10`
C. `5 15 5`
D. `10 20 10`
E. `5 20 5`

e) **[0.5 marks]** What is the output of the following code (if any)?

A. `Finished`
B. `Exception`
C. `Arithmethic Exception`
D. None, compilation fails
E. Nothing is printed

```
try {
    int x = 0;
    int y = 5 / x;
} catch (Exception e) {
    System.out.println("Exception");
} catch (ArithmeticException ae) {
    System.out.println(" Arithmetic Exception");
}
System.out.println("Finished");
```

4 – [1.0 marks] Explain the open-closed principle of OOP.

In OOP, the open-closed principle states that software entities (classes, interfaces, etc.) should be open for extension, but closed for modification.

Name:	Number:

Part III -- XML (1 mark)

5 – [1.0 marks] Consider the problem presented in Part I and, particularly, problem 1b), that is, consider one observation section with two beds, a doctor and a nurse, with one patient waiting in the section queue. Informally, present an XML document, with XML elements and attributes, to store all information needed, as well as the corresponding DTD.

Part IV -- Swing (1 mark)

6 – [1.0 marks] What is a layout manager?

A layout manager is an object that implements the LayoutManager interface; it determines the size and position of the components within a container.