Freie Universität Bozen
Libera Università di Bolzano
Università Liedia de Bulsan

Advanced Programming

Written Examination

April 5th 2015

FIRST NAME		**LAST NAME**	
STUDENT NUMBER		**SIGNATURE**	

Instructions for students:

Write First Name, Last Name, Student Number and Signature where indicated. If not, the examination can not be marked.

Do not speak to any other student during the examination. If you speak to another student, your examination will be cancelled.

Use a pen, not a pencil.

Write neatly and clearly.

1. Exercise (RegEx) (max 8 points)

The input text contains, in each line, the birthdays of people, their surnames and names in the american date format YYYY-DD-MM (year-day-month). The three fields are separated using commas in the CSV (Comma-Separated Value) format:

```
1965-18-03,Baggio,Roberto
1949-22-12,Agostini,Giacomo
```

Write a program (fill in the code below - each of the correctly solved segments awards you 3, 2 and 3 points, respectively) that uses regular expressions to search in these lines and print on the console each line in the following format: Name Surname (DD.MM.YYYY), for example,

```
Roberto Baggio (18.03.1965)
Giacomo Agostini (22.12.1949)
```

```
public static void main(String[] args) {
    String inputText = "1965-18-03,Baggio,Roberto\n1949-22-12,Agostini,Giacomo";

    // (1) write the regular expression in the regexp variable (3 points)
    String regexp = "";

    String regexp = "([0-9]+)-([0-9]+)-([0-9]+),(\\w+),(\\w+)";


    // (2) use Pattern and Matcher correctly (2 points)


    Pattern p = Pattern.compile(regexp);
    Matcher m = p.matcher(inputText);


    // (3) iterate through the Matcher to find and print to System.out (3 points)
    while(m.find()){
        System.out.println(m.group(5) + " " +m.group(4) + " (" +m.group(2) + "." +m.group(3) +
"." +m.group(1) + ")");


    }
}
```

2. Exercise (XML) (4 points)

For the XML file below draw the XML tree.

```
<?xml version="1.0" encoding="UTF-8" ?>
<records>
    <record type="vinyl">
        <title>Kind of Blue</title>
        <artist name="Miles Davis"/>
        <artist name="John Coltrane"/>
        <artist name="Bill Evans"/>
    </record>
</records>
```

3. Exercise (serialisation) (4 points)

What are the values of the object variables after deserialisation? Write in the box below the code.

```
[Person.java]
public class Person implements java.io.Serializable {
    public String name;
    public String address;
    public int SSN;
    public transient int age;
}

[Program.java]
...
public static void main(String[] args) {
    Person e = new Person();
    e.name = "John Serebeton";
    e.address = "Viale Druso, 17, Bolzano";
    e.SSN = 6357894;
    e.age = 32;

    String filename = "user.ser";
    // SERIALIZE
    try
    {
        ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream(filename));
        out.writeObject(e);
        out.close();
    }catch(IOException i)
    {
        i.printStackTrace();
    }

    e.age=34;
    e.name = "Claes Mitch";

    // DESERIALIZE
    try
    {
        FileInputStream fileIn = new FileInputStream(filename);
        ObjectInputStream in = new ObjectInputStream(fileIn);
        e = (Person) in.readObject();
        in.close();
        fileIn.close();
    }
    catch(IOException i){
        System.out.println("IOException found!");
    }
    catch(ClassNotFoundException c){
        System.out.println("ClassNotFoundException found!");
    }
}
```

```
Name: John Serebeton

Address: Viale Druso, 17, Bolzano

SSN: 6357894

Age: 0
```

4. Exercise (IO Streams) (max 4 points)

We want to write a java program that writes 2 long variables (var1 and var2) into the file export.dat. Write the code that does that (fill in the code below - 3 points). You don't need to optimize for speed. Work with binary streams.

```
public static void main(String[] args) {
    String fileName = "export.dat" ;

    long var1 = 123131;
    long var2 = 234234;

    try
    {
        // create the stream object (name it "out") correctly (2 points)


       DataOutputStream out = new DataOutputStream( new FileOutputStream( fileName  ) );

       OR

       DataOutputStream out = new DataOutputStream( new BufferedOutputStream( new
FileOutputStream( fileName )));

        // write the variables var1 and var2 to the stream (1 point)

        out.writeLong(var1);
        out.writeLong(var2);

        out.close();
    }
    catch ( IOException iox )
    {
        System.out.println("Problem writing " + fileName );
    }
 }
```

How many bytes will the export.dat file have? **(1 point)**

16

5. Exercise (Theory) (max 5 points)

Describe the difference between stack and heap:

(2 points) Describe the role of stack and heap in the execution of a program.

The stack models the memory responsible for any computation. It is filled by blocks that are popped in or out according to the LIFO principle. The memory stored in the stack is defined static as it refers to computation. The heap models the dynamic memory, i.e., the objects and the area of statics or interfaces.

(1 point) Describe the different types of memory blocks you have in both and the information each block contains (e.g., what is an AR and which kind of information does it contain?) (e.g., what does SL indicate in the SAR of a "for loop").

In the stack we have Activation Records (AR) and Scope Activation Records (SAR). The former contains information to perform the computation in a method (SP, SL, RV, RA, local variables of method, reference to objects that eventually called the method)), the latter contains information to perform computation in any non-method code block (e.g., loops). In particular, an AR is a SAR and contains the information about the scope of variables in the block.

(2 points) Describe each type of information in each block and its role in the execution of a program (e.g., what SL stands for and what is it useful for?)

Stack
AR:
RA: the line of code in which the method returns
RV: the value returned by the method
SL: Static link: the outer code block containing the one of the current method
SP: Stack pointer: address in the stack of the memory of the code that called the current method
local variables: reference variables (containing the address of their objects in the heap), primitive variables
this: only used in instance calls. It is the address in the heap of the object with which we invoke the method
SAR:
SL: Static link: the outer code block containing the one of the current method

local variables: reference variables (containing the address of their objects in the heap), primitive variables

Heap:
Objects of type : memory block of the object of a given type. It contains object attributes. If they are reference variables they contain the address of the referring objects in the heap
Area of static of a type: memory allocated to the static attributes of a type

6. Exercise (Memory Model) (max 5 points)

Draw the memory models of the following piece of code:

```
public class Producer {
    Product myProduct;

    Producer(){
        myProduct=new Product();
    }
}

public class Product {
    Producer myProducer;
    private static int ID = 0;

    Product(){
        myProducer=new Producer();
        ID++;
    }
}

public class Factory {
    public static void main(String[] args) {
        Producer myProducer =new Producer();
        myProducer = null;
    }
}
```

This is stackoverflow (the stack fills with constructors); the lines ID++; ad myProducer = null; are never reached. The area of static of Product in the heap contains ID=0 that never increments; use "this" in the constructors to point to their objects in the heap; the constructors do not have any local variable; the objects of type Producers in the heap have a local variable myProduct that points to an object in the heap of type Product and vice-versa.

7. Extra Exercise (max 3 points)

Assume you have the bytecode (.class file) of the following source code (that you do not have!)

```
[1]  class foo{
[2]      void for99(){
[3]          for(int i=0; i<99; i++){}
[4]      }
[5]      void while99(){
[6]          int i =0;
[7]          while(i<99){
[8]              i++;
[9]          }
[10]      }
[11]  }
```

How can you prove that the source code contains two methods that do exactly the same thing?

Using java disassembler verbose javap -c foo.class and comparing the output