Second mid-term lab exam – Computer Programming 11/06/2015 Duration: 1h

(Notice: the maximum score of this exam is 10, but the total weight of this exam in the final grade is 1,20)

Do not fill.

Write your complete name and the code of your lab group:

Last name, first name:		Group:	

First question. **(3 points)**

In the class BankManager from lab practice 4 we implemented the following method:

```
public static double getBalanceAfterWithdrawal( Account c, double amount )
{
      c.withdraw(amount);
      return c.getBalance();
}
```

But the compiler shows the following error message:

(unreported exception NoMoneyException; must be caught or declared to be thrown)

<u>You have to</u> modify the method getBalanceAfterWithdrawal() for adding the proper instructions for handling the abovementioned exception and checking if the account is null. In both cases the method should show an error message indicating why and returning 0 as the result.

```
   public static double getBalanceAfterWithdrawal( Account c, double amount
) {
      try {
         c.withdraw(amount);
         return c.getBalance();
      } catch( NullPointerException e ) {
         System.out.println( "The account does not exist!" );
      } catch( NoMoneyException e ) {
         System.out.println( "Insufficient balance!" );
      }
      return 0;
   }
```

Second question. **(2 points)**

For the lab practice 4, it is being implemented a new method for reading from a binary file the accounts with their balance and to write into another binary file those accounts whose balance is greater than one million euros. The name of the input binary file is in the parameter **a** and the name of the output binary file is in the parameter **b** of the method.

```
public void createFileForMillionaire( String a, String b ) {
   ObjectInputStream f = new ObjectInputStream(new FileInputStream(a));
   ObjectOutputStream g = new ObjectOutputStream(new FileOutputStream(b));
   try {
      while( true ) {
         // loop: read from file a and writes to file b
         // infinite loop: ends when an exception of the class
         //  EOFException has been thrown
      }
   } catch( EOFException e ) {
      f.close(); g.close();
   }
}
```

The method should be completed taking into account the following conditions:

- It is known that the input file only contains objects of the class Account, the accounts are well defined and you don't need to check it.
- The number of objects in the input file is unknown, the method should read one object of the class Account at a time. If the loaded account fulfils the condition of millionaire then it should be written to the output file.
- The method should propagate exceptions of the classes **IOException** and **ClassNotFoundException**.

You have to tell which code must be added and where in order to achieve the functionality specified by the above conditions.

- 1°) Add in the profile of the method the throws clause as follows:

```
     throws IOException, ClassNotFoundException
```

- 2°) Put the following code in the body of the loop:

```
     Account c = (Account) f.readObject();
     if ( c.getBalance() > 1e+6 )
          g.writeObject(c);
```

Do not fill.

Third question. **(2 points)**

Here you have a wrong implementation of the method **`insertInOrder`**(String, int) of the class **`Concordance`** that presents some execution errors:

```
 0   private void insertInOrder( String word, int numLine ) {
 1      NodeCnc temp = prim, prev = null;
 2      while( temp != null && temp.getWord().compareTo(word) > 0 ) {
 3         temp = temp.getNext();
 4         prev = temp;
 5      }
 6      if ( temp != null && temp.getWord().compareTo(word) == 0 )
 7         temp.numLines.enqueue(numLine);
 8      else {
 9         NodeCnc newNode = new NodeCnc( word, numLine, temp );
10         size++;
11         prev.setNext( newNode );
12      }
13   }
```

The data structures **`Concordance`** and **`NodeCnc`** have the attributes that are shown in the fourth question (see next page), and the methods used during the lab sessions.

<u>**You have to**</u> identify each error and write the code with the solution.

- 1°) In line 2, the comparison operator must be <

```
2       while( temp != null && temp.getWord().compareTo(word) < 0) {
```

- 2°) Lines 3 and 4 must be interchanged:

```
3       prev = temp;
4       temp = temp.getNext();
```

- 3°) Line 11 should be substituted by the following code:

```
11        if ( temp == null)
             first = newNode;
          else
             prev.setNext( newNode );
```

Do not fill.

Fourth question. **(3 points)**

In the application of the lab practice 5, we need the possibility of removing all the information related to a particular line in the text.

Suppose it is available in the class **`QueueIntLinked`** a method with the following profile: **`public void remove( int n )`**, that removes all the appearances of **`n`** in the queue.

The attributes of classes **`Concordance`** and **`NodeCnc`** are the following:

```
Concordance                        NodeCnc
private NodeCnc first;             String word;
private int size;                  QueueIntLinked numLines;
private boolean isOrd;             NodeCnc next;
private String delimiters;
```

You have to: implement in the class **`Concordance`** a new method with an integer **`n`** as parameter for removing the appearances of **`n`** from the queue **`numLines`** of each node in the list. Then, all the nodes whose queue **`numLines`** gets empty should be removed from the list in the concordance. The attributes of the class **`Concordance`** must be updated properly.

```
public void remove( int n )
  {
   NodeCnc temp = first, prev = null;
   while( temp != null ) {
      temp.numLines.remove( n );
      if ( temp.numLines.size() == 0 ) {
         if ( prev != null ) prev.setNext( temp.getNext );
         else first = first.getNext();
         size--;
      } else prev = temp;
      temp = temp.getNext();
   }
}
```

Do not fill.