COMPUTER PROGRAMMING RETAKE LAB EXAM 23/06/2015 Duration: 1 Hour

To be filled out by teacher

Last name, first name		Group:	

First question. **(2 points)**

You have to complete the implementation of a recursive method named **`isBeginAndEnd()`**. This method has two arguments, **a** and **b**, objects of the class **`String`** and checks if **a** is prefix and inverse suffix of **b**, i.e., the symbols of **a** can be read from left to right at the beginning of **b** and from right to left and the end of **b**. Here you have some examples about how the method runs.

```
boolean x = isBeginAndEnd("lasa","lasaladelasal");   // x ← true
boolean y = isBeginAndEnd("des","desiertodelased");  // y ← true
boolean z = isBeginAndEnd("los","losdiasdesol");     // z ← true
boolean v = isBeginAndEnd("susi","susaludasusta");   // v ← false
boolean w = isBeginAndEnd("dos","dosdiasmalos");     // w ← false
```

The partial implementation of the method you have to complete is:

```
0     /** precondition: 2 * a.length() <= b.length() */
1     public static boolean isBeginAndEnd( String a, String b ) {
2       if ( a.length() == 0 ) return true;
4       else return a.charAt(0) == ...
5            && a.charAt(0) == ...
6            && isBeginAndEnd( ...
7     }
```

You have to complete it by adding the missing code in the general case, i.e. complete the lines ended by dots.

```
public static boolean isBeginAndEnd( String a, String b ) {
   if ( a.length() == 0 ) return true;
   else return a.charAt(0) == b.charAt(0)
        && a.charAt(0) == b.charAt(b.length()-1)
        && isBeginAndEnd(a.substring(1), b.substring(1,b.length()-1));
}
```

Second question. **(2 points)**

Given the following table with the running times in milliseconds of two algorithms which solve the same problem:

# Size n	Algorithm A TA(n)	Algorithm B TB(n)
1000	10148.00	397.81
2000	19753.00	1501.83
3000	25327.00	3312.83
4000	34552.00	5820.27
5000	44145.00	9029.67
6000	51902.00	12936.26
7000	61002.00	17543.27
8000	69827.00	22835.75
9000	77687.00	30385.04
10000	86409.00	36873.96
11000	94799.00	43157.29
12000	103808.00	51282.19
13000	112750.00	59839.54
14000	124301.00	69399.81
15000	131841.00	81848.08
16000	143048.00	90444.16
17000	154218.00	102002.64
18000	166435.00	116535.31
19000	173328.00	127247.52
20000	182737.00	143252.33

You have to give an answer to the following questions. All answer should be justified.

a) What is the asymptotic behaviour that best matches with the data on column **Alg. A**?

b) What is the asymptotic behaviour that best matches with the data on column **Alg. B**?

c) Which algorithm has a better asymptotic behaviour?

a) The asymptotic cost that best matches the data on column **Alg. A** is linear, because as the input size is multiplied by 2 the temporal cost of the algorithm is also multiplied by 2. If f(2n)/f(n) = 2n/n = 2, then f(n) is linear.

b) The asymptotic cost that best matches the data on column **Alg. B** is squared, because the quotient f(2n)/f(n) gives a value around 4. So, if a function f(n) is squared then the condition f(2n) = 4 f(n) is fulfilled. If the input size is multiplied by 2 then the value of the function is multiplied by 4. Considere f(x)=x*x as an example, it can be observed that f(2*x) = 4*f(x), because f(2*x) = (2*x)*(2*x) = 2*2*x*x = 4*x*x = 4 * f(x).

c) Obviously, the asymptotic behaviour of algorithm A is better than the one of algorihtm B, despite the fact the running times of algorithm A are worse than the running times of algorithm B for small input sizes.

To be filled out by teacher

Third question. **(3 points)**

To be filled out by teacher

Taking as starting point the project of lab practice number 4, which has been properly modified in order to allow accounts with deficit, i.e. the balance of the account could be negative.

For allowing this feature, we added a new **boolean** attribute in the class **Account** named **deficitAllowed**. When its value is true the program should allow negative balances. We also added a static attribute named **MAX_DEFICIT** whose value is the maximum amount of money allowed as deficit in an account. Now, the attributes of the class are:

```
private static final double MAX_DEFICIT = 1000.0;
private double balance;
private int accountId;
private boolean deficitAllowed;
```

Additionally, we have implemented a class named **DeficitExceededExcpetion**, derived from class **Exception**, for managing unexpected situations caused by a negative balance that can exceed the maximum allowed deficit.

You have to implement a method in the class **Account** named **payReceipt()**, such that given a float number (the import of the receipt) makes the payment by subtracting it from the balance of the account except if one of the following conditions are true:

- The account doesn't admit deficit and the import of the receipt is greater than the existing balance in the account, then an exception of the class **NoMoneyException** with the appropriated message. Exceptions of this class should be propagated.

- The account does admit deficit and the import of the receipt is greater than the sum of the existing balance in the account plus the maximum allowed deficit. In this case an exception of the class **DeficitExceededException** should be thrown with the appropriated message. Exceptions of this class also should be propagated.

```
public void payReceipt( double amount )
   throws NoMoneyException, DeficitExceededException {
   if ( !deficitAllowed && amount > balance )
      throw new NoMoneyException(
      "Impossible to pay the receipt. No enough money in the account!");
   if ( deficitAllowed && amount > balance + MAX_DEFICIT )
      throw new DeficitExceededException(
      "Impossible to pay the receipt. Maximum deficit allowed exceeded!");
   balance -= amount;
}
```

Fourth question. **(3 points)**

To be filled out by teacher

In the project of lab practice number 5, we need to obtain from the concordance all the words that appear in a given line of the text.

We will suppose that in the class **`QueueIntLinked`** there is a new method with the following profile **`public boolean contains( int n )`**, that returns true if the integer **`n`** is in the queue in relation to the method is invoked, otherwise the method returns false.

Remember that the data structures of classes **`Concordance`** and **`NodeCnc`** have the following attributes:

Se recuerda que las estructuras de datos **`Concordancia`** y **`NodoCnc`**, vistas en prácticas, tienen los atributos siguientes:

Concordance

```
private NodeCnc first, last;
private int size;
private boolean isOrd;
private String delimiters;
```

NodeCnc

```
String word;
QueueIntLinked lineNumbers;
NodeCnc next, previous;
```

You have to implement in the class **`Concordance`** a method with an integer **`n`** as parameter representing a line number. The method should return an object of class **`String`** with all the words that appear in the line **`n`** of the text. Words should be delimited by comas. The method should operate independently whether the concordance is sorted or not. The order in that the words appear in the resulting **`String`** is not relevant.

Example: given the concordance that appear in the box of the right, if the method is invoked with respect to this object of the class **`Concordance`** passing the value 2 as parameter, then it should return an object of the class String with the following contents: **"More, tall, state, "**; and if it is invoked with the value 4 then **"the, yes, and, "** should be returned.

```
More (2):     1  2
blue (2):     1  1
than (2):     1  3
the (4):      1  3  4  4
tall (1):     2
state (1):    2
love (1):     3
loving (1):   3
yes (3):      3  4  4
and (2):      4  4
```

```
public String wordsInTheLine( int n ) {
   String s = "";
   for( NodeCnc p = first; p != null; p = p.next )
      if ( p.getLineNumbers().contains(n) ) s += p.getWord() + ", ";
   return s;
}
```