Retake Second Lab Partial Exam – PRG – ETSINF – Academic year 2013/14
June 23^{rd}, 2014 – Duration: 50 minutes

1. 2.5 points The method `readWordFrom()`, whose code is below, reads a `String` and an integer from an object of the class `Scanner` and returns a substring of the read `String` from the position specified by the read integer. When the method `substring(int)` is executed an exception of the class `IndexOutOfBoundsException` could be thrown if the integer is negative or greater than the length of the read `String`.

```
public static String readWordFrom( Scanner sf )
{
    System.out.print( "Enter a word: " );
    String line = sf.nextLine();
    System.out.println( "Enter a position: " );
    int pos = sf.nextInt();
    sf.nextLine();
    return line.substring( pos );
}

public static void main( String[] args )
{
    Scanner input = new Scanner( System.in );
    String word = readWordFrom( input );
    System.out.println( "The read word is: " + word );
}
```

To be done: Modify the method `main()` in order to **catch** exceptions of the class `IndexOutOfBoundsException` and asking for the data again. In such a way that the method `readWordFrom(Scanner)` should be executed as many times as needed until no exception is thrown.

Solution:

```
public static void main( String[] args )
{
    Scanner input = new Scanner( System.in );
    boolean error = true;
    do {
        try {
            String word = readWordFrom( input );
            System.out.println( "The read word is: " + word );
            error = false;
        }
        catch( IndexOutOfBoundsException e ) {
            System.out.println( "Be careful and enter a correct position!" );
        }
    } while( error );
}
```

2. 2.5 points Write a method with two parameters, a `double` and an object of the class `Scanner` that references a text file with the accounts of a bank. The method has to read the accounts from the given file and write to a new file named `"accounts.txt"` the id of the accounts with a balance greater than the `double` value passed as parameter.

   Each line in the input file contains two values, the id of an account (an `int`) and the balance of that account (a `double`). The header of the method is

```
public static void selectAccounts( double b, Scanner in ) throws Exception
```

   No exceptions must be caught because all possible exceptions are propagated.

Solution:

```
public static void selectAccounts( double b, Scanner in ) throws Exception
{
    PrintWriter out = new PrintWriter( "accounts.txt" );
    while( in.hasNext() ) {
        int accountId = in.nextInt();
        double balance = in.nextDouble();
        if ( balance > b ) out.println( accountId );
    }
    out.close();
}
```

3. 2.5 points Given the data structures `Concordance` and `NodeCnc` as they were studied in lab practises and with the following attributes:

```
Concordance                          NodeCnc
------------                         -------
private NodeCnc first;               String word;
private NodeCnc last;                QueueIntLinked lineNumbers;
private int size;                    NodeCnc next;
private boolean isSorted;            NodeCnc previous;
private String delimiters;
```

Write a method in the class `Concordance` with the profile:

```
// PRECONDITION: n >= 1
public int numberOfWordsAppearingMoreThanNTimes( int n )
```

that returns how many words appear more than `n` times in the text used for building the `Concordance`.

Solution:

```
// PRECONDITION: n >= 1
public int numberOfWordsAppearingMoreThanNTimes( int n )
{
    int counter = 0;
    for( NodeCnc temp = first; temp != null; temp = temp.next )
        if ( temp.lineNumbers.size() > n ) counter++;
    return counter;
}
```

4. 2.5 points The following code is incomplete and corresponds to one of the possible implementations of the method `insertNotInOrder( String, int )` of the class `Concordance`.

```
private void insertNotInOrder( String word, int lineNumber )
{
    NodeCnc temp = first, previous = null;

    while( /* CONDITION */ ) { previous = temp;  temp = temp.next; }

    if ( /* CONDITION */ )  // word already exists.
        /* ONE INSTRUCTION */
    else {  // The new NodeCnc with the word should be the last one in the list
            // and it should be inserted after the node referenced by previous
            NodeCnc newNode = new NodeCnc( word, lineNumber );
            size++;
            if ( previous == null ) // The new node is the first one and the last one.
                /* ONE INSTRUCTION */
            else { // The new node is the last one
                /* INSTRUCTIONS */
            }
        }
}
```

To be done: complete the code with:

a) [0.6 points]: the condition of the while loop.

b) [0.4 points]: the condition of the first if.

c) [0.7 points]: the missing instruction when the word already exists in the `Concordance`.

d) [0.4 points]: the missing instruction when the word should be the first one and the last one in the `Concordance`.

e) [0.4 points]: the missing instructions when the word should be the last one in the `Concordance`.

Solution:

```
private void insertNotInOrder( String word, int lineNumber )
{
    NodeCnc temp = first, previous = null;
    while( temp != null && !temp.word.equals( word ) ) {  // a)
        previous = temp;  temp = temp.next;
    }

    if ( temp != null )                                   // b)
        temp.lineNumbers.enqueue( lineNumber );           // c)
    else {
        NodeCnc newNode = new NodeCnc( word, lineNumber );
        size++;
        if ( previous == null )
            first = last = newNode;                       // d)
        else {
            last.next = newNode;                          // e)
            last = newNode;                               // e)
        }
    }
}
```