This second Java programming work consists of the partial implementation of a parametric list using recursive algorithms to traverse the list. And a unitary test program to validate the correctness of methods already implemented or to be implemented.

Start by reading the description of the ***IntId*** class (**IntId.html** document). Under no circumstances should this data type be changed since it is used to test the algorithms of the linked list.

We intend to implement the ***RecLinkedList*** data type using defensive programming with exceptions thrown in case of abnormal data situations that violate the preconditions and/or postconditions of the methods. For this purpose, an exception class called *EmptyListException* is provided to indicate when a list is empty. The other used exception classes *IndexOutOfBoundsException* and *NoSuchElementException* already exist in the Java language.

1. Start by reading the description of the ***RecLinkedList*** class (**RecLinkedList.html** document). Then implement the private helper functions that effectively do the recursive traversal of the list: *insertPos*; *copy*; *reverse*; *searchDown*; *searchUp*; *removeAllElements*; *removeFirstElement*; *removeLastElement*; *firstIndexOf*; and *lastIndexOf*. Some of which you may have already implemented last year in Programação II. Under no circumstances should you change the other methods.
2. Complete the **TestRecList.java** testing program that tests some of the methods in the recursive list. Add tests using the *assert* statement to validate the correction of missing or partially tested methods. Also don't forget to test all methods for an empty list, for example in a separate simulation program, to ensure that the methods are implemented robustly.

Very important remark: bear in mind that the private recursive functions ***remove***, in addition to disconnecting the desired element from the list, bring its reference so that the invoker method can return the element removed from the list to the program. In order to be able to simultaneously return the reference of the element and the reference of the next element in the list to re-link the list, it is necessary to use an internal data type (**private class DelNode**) that stores two references. To understand how a function can be implemented in this way, the ***removePosElement*** method is used as a solved example. This is the only possible way to implement the ***removeLastElement*** method.

The work must be delivered by June 5, 2022. Only one of the students in the group must send by email to adrego@ua.pt a zip file containing only the final files of the class and the unitary test program: **RecLinkedList.java** and **TestRecList.java**.

For any Java and linked lists doubts you can and should contact me by email with questions about the work.