PRG – ETSInf – THEORY – Academic year 2017-18
Retake second mid term exam – 19 June 2018 – Duration: 2 hours

Note: Maximum mark for this exam is 10 points, but its weight in the final grade of PRG is **3 points**.

1. 2.5 points **What to do:** Write an static method for loading integer values from a text file. The filename is provided as a parameter for the method. The method must store the loaded values into an object of the class `StackIntLinked`. The method must return (the reference to) the object of the previously mentioned class. The first valid integer read from the file must be at the bottom of the stack.

   Details to be taken into account:

   - The file can contain tokens which are not integer values, so exceptions of the class `InputMismatchException` can be thrown and must be appropriately caught for allowing the method to continue loading correct integer values that appear in the text file after invalid tokens.
   - If the file is empty or contains no correct integers the returned object will be an empty stack.
   - If the file doesn't exist an exception of a well known class can be thrown. Such exceptions must be caught within the method and the message ``File not found'' should be shown on screen.
   - If the object of the class `Scanner` was successfully created, then it must be closed independently of the exceptions thrown during the process of loading/reading the integer values from the file.

Solution:

```
public static StackIntLinked fileToStack(String name) {
    Scanner s = null;
    StackIntLinked stack = new StackIntLinked();
    try {
        s = new Scanner(new File(name));
        while (s.hasNextLine()) {
            try {
                stack.push(s.nextInt());
            } catch (InputMismatchException n) {
                s.nextLine();
            }
        }
    } catch (FileNotFoundException e) {
        System.out.println("File not found!");
    } finally {
        if (s != null) { s.close(); }
    }
    return stack;
}
```

2. 2.5 points **What to do:** Write an static and iterative method for copying the elements stored in an object `l` of the class `ListPIIntLinked` given as parameter into a new list. The method must return the new list. The values stored in the new list must be the result of subtracting the minimum in the original list.

   For instance, if the original list `l` contains `[12] 50 10 120`, the minimum is `10` and the new list to be returned must be `2 40 0 110 [ ]`. `[ ]` refers to the position of the cursor.

   Additionally, the following details must be taken into account:

   - If `l` is empty the method must return `null`.
   - The original values stored in `l` must remain untouched. Only the cursor of `l` can be changed.
   - **REQUIREMENT:** The method you have to write is a method that will be in a class different from the class `ListPIIntLinked`, so **you can only use** the public methods of the class `ListPIIntLinked`.
   - In the implementation used in the ARA group the name of the class with interest point was `ListIntLinked`, so in your solution you can use indistinctly `ListPIIntLinked` or `ListIntLinked`.

Solution:

```
public static ListPIIntLinked subtractMinimumToList(ListPIIntLinked l) {
    if (l.empty()) { return null; }
    ListPIIntLinked res = new ListPIIntLinked();
    l.begin();
    int min = l.get();
    while (!l.isEnd()) {
        if (l.get() < min) { min = l.get(); }
        l.next();
    }
    for (l.begin(); !l.isEnd(); l.next()) {
        res.insert(l.get() - min);
    }
    return res;
}
```

3. 2.5 points **What to do:** Write an static method that given a linked sequence of objects of the class `NodeInt` returns another linked sequence of objects of the same class with the even numbers. For instance, if the provided linked sequence is: `4 7 2 8 9 3 6` the returned sequence must be `4 2 8 6`. If the linked sequence provided is `null` or does not contain even numbers, then the method must return `null`. You can provide a solution either with single linked sequences or double linked sequences.

Solution:

```
public static NodeInt evenSubsequence(NodeInt seq) {
    NodeInt first = null;
    NodeInt last = null;
    while (seq != null) {
        if (seq.data % 2 == 0) {
            if (first == null) {
                last = new NodeInt(seq.data);
                first = last;
            }
            else {
                last.next = new NodeInt(seq.data);
                last = last.next;
            }
        }
        seq = seq.next;
    }
    return first;
}
```

4. 2.5 points **What to do:** Write a non-static method in the class `QueueIntLinked` that splits the queue into two halves. The profile of the method must be `public QueueIntLinked splitQueue()`

   The following details must be taken into account:

   - Precondition: the initial queue (`this`) has two elements at least, i.e. `this.size() >= 2`.
   - The partition must be done in the following way: the original queue must contain the first half and the new queue to be returned should contain the remaining elements.
   - The order of the elements in both queues after the execution of the method must be the same order in the original queue.
   - If the size of the original queue is an odd number, the returned queue should be the longest.

   Some examples:

Initial queue	Initial queue modified	Returned queue
1 2 2 4 3 1	1 2 2	4 3 1
1 2 2 4 3 1 1	1 2 2	4 3 1 1

REQUIREMENT: In the method to be implemented only the attributes of the class `QueueIntLinked` can be used, the public methods can not be used, except the constructor. References to objects of the class `NodeInt` and the methods of this class can be used.

Solution:

```
    public QueueIntLinked splitQueue() {
        QueueIntLinked nq = new QueueIntLinked();
        int middle = size / 2;
        NodeInt aux = this.first;
        for (int i = 0; i < middle - 1; i++) {
            aux = aux.next;
        }
        nq.first = aux.next;
        nq.last = this.last;
        aux.next = null;
        this.last = aux;
        nq.size = this.size - middle;
        this.size = middle;
        return nq;
    }
```

ANNEX

Methods of the classes `StackIntLinked` and `ListPIIntLinked`, plus the attributes of the class `QueueIntLinked`.

```
public class StackIntLinked {
        ...
    public StackIntLinked() { ... }
    public boolean empty() { ... }
    public int size() { ... }
    public void push(int x) { ... }
    public int pop() { ... }
    public int peek() { ... }
    public boolean equals(Object o) { ... }
    public String toString() { ... }
}

public class ListPIIntLinked {
    ...
    public ListPIIntLinked() { ... }
    public boolean empty() { ... }
    public int size() { ... }
    public boolean isEnd() { ... }
    public void begin() { ... }
    public void next() { ... }
    public void insert(int x) { ... }
    public int remove() { ... }
    public int get() { ... }
    public boolean equals(Object o) { ... }
    public String toString() { ... }
}

public class QueueIntLinked {
    private NodeInt first;
    private NodeInt last;
    private int size;
    public QueueIntLinked() { }
    ...
}
```