PRG – ETSInf – Academic year 2012/2013 – Theory – Partial exam 2
June 7th, 2013 – Duration 2 hours

1. [2 points] Given the class `StackEmptyException` derived from the class `Exception`, already implemented and available to be used (**you don't have to implement it**). It is possible to define the class `StackIntLinked` in such a way that their methods can thrown an exception of the class `StackEmptyException` if they are invoked in relation with an empty stack, instead of assuming as a precondition they can't be executed if the stack is empty.

   To be done:

   Write the methods of the class `StackIntLinked` that they should never be executed with respect to an empty stack. Your version of these methods must throw an exception of the class `StackEmptyException` in the case they are invoked in relation with an empty stack.

   Solution:

```
/** Removes and returns the value on the top of the stack */
public int pop()
    throws StackEmptyException
{
    if ( 0 == size ) throw new StackEmptyException( "Stack overflow!" );

    int value = this.top.datum;
    this.top = this.top.next;
    this.size--;
    return x;
}

/** Returns the value on the top of the stack */
public int top()
    throws StackEmptyException
{
    if ( 0 == size ) throw new StackEmptyException( "Stack overflow!" );

    return this.top.datum;
}
```

---

2. [2.5 points] Given a linked sequence of integer numbers `seq` and an integer value `x`, you have to write a method with the following profile:

```
public static int lastAppearanceOf( NodeInt seq, int x )
```

   The method should return the position of the last appearance of `x` in the sequence, or `-1` if `x` is not in the sequence. Zero is the position of the first value in the sequence.

   Solution:

```
public static int lastAppearanceOf( NodeInt seq, int x )
{
    NodeInt current = seq;
    int pos = 0, lastAppearance = -1;
    while( current != null ) {
        if ( current.datum == x ) lastAppearance = pos;
        pos++;
        current = current.next;
    }
    return lastAppearance;
}
```

---

3. 2.5 points Given a non-empty stack of integers, you have to write a recursive method with the following profile:

```
public static void removeBottom( StackIntLinked s )
```

The method must remove the oldest value, the one which is at the bottom of the stack, and leave the remaining values in the same order.

Solution:

```
/** s is a non-empty stack */
public static void removeBottom( StackIntLinked s )
{
    if ( s.size() > 1) {
        int x = s.pop();
        removeBottom(s);
        s.push(x);
    } else
        s.pop();
}
```

---

4. 3 points Given two lists of integers with interest point, `list1` and `list2`, both sorted in strictly increasing order, you have to write a method with the following profile:

```
// list1 and list2 are in strictly ascending order
public static ListIPIntLinked union( ListIPIntLinked list1, ListIPIntLinked list2 )
```

The output should be a new list with the union of both lists with all the values in strictly increasing order. Remember, the union has no repeated values.

Example:

Let `list1` be a list with the values: { 1, 3, 5, 7, 9, 11, 13, 15, 17, 19 }
Let `list2` be a list with the values: { 1, 4, 7, 10, 13, 16, 19, 21, 27 }

The result of `union( list1, list2 )` must be the following list:

{ 1, 3, 4, 5, 7, 9, 10, 11, 13, 15, 16, 17, 19, 21, 27 }

Solution:

```
/** list1, list2 are in strictly increasing order */
public static ListIPIntLinked union( ListIPIntLinked list1, ListIPIntLinked list2 )
{
    ListIPIntLinked li = new ListIPIntLinked();

    list1.begin();
    list2.begin();

    while( !list1.atTheEnd() && !list2.atTheEnd() ) {

        int i = list1.get(), j = list2.get();

        if ( i < j ) {
            li.insert(i);
            list1.next();
        } else if ( i > j ) {
            li.insert(j);
            list2.next();
        } else {
            li.insert(i);
            list1.next();
            list2.next();
        }
    }

    while( !list1.atTheEnd() ) {
        li.insert( list1.get() );
        list1.next();
    }

    while( !list2.atTheEnd() ) {
        li.insert( list2.get() );
        list2.next();
    }
    return li;
}
```