PRG - ETSInf. THEORY. Academic Year 2014-15. Second mid-term exam
11th of june, 2015 – Duration: 2 hours

1. 2.5 points Write a static method according to the following rules:

   - It should have as a parameter an object of the class `String` that it is supposed contains the absolute path of a text file.
   - The method should work fine for all the lines in the file, the number of lines is unknown and can vary depending on the file.
   - The contents of each line could be a valid integer number or not.
   - It should propagate exceptions of the class `FileNotFoundException` if the given file cannot be opened.
   - The method should return the sum of all the valid integers found in the file.
   - When a valid integer is found its value should be added to the sum.
   - Otherwise the method should catch the exception of the class `InputMismatchException` that could be thrown, and show on screen a message including the name of the exception and the token that is not a valid integer, i.e., the contents of the line.
   - Each time an exception of the class `InputMismatchException` has been caught the method should continue reading lines till the end of the file is reached.

Solution:

```
public static int sum( String f ) throws FileNotFoundException {
    int sum = 0;
    Scanner sf = new Scanner( new File(f) );
    while( sf.hasNextLine() ) {
        try {
            sum += sf.nextInt();
        }
        catch ( InputMismatchException e ) {
            System.err.println(  e+ "::" + sf.nextLine() );
        }
    }
    sf.close();

    return sum;
}
```

2. 3 points Add a new method to the class `QueueIntLinked` with the following profile:

```
public void move_to_the_first_position( int x )
```

that searches the first element in the queue equal to `x`, if it exists then it sould be moved to the first position of the queue, otherwise the queue should not be altered.

NOTICE: This method should work with the internal attributes of the class `QueueIntLinked`. The use of its methods is not allowed.

Solution:

```
/** If x exists in the queue it is moved to the first position. */
public void move_to_the_first_position(int x)
{
```

```
    NodeInt temp = first, prev = null;
    while( temp != null && temp.value != x ) {
       prev = temp;
       temp = temp.next;
    }

    if ( temp != null && temp != first ) {
       prev.next = temp.next;
       temp.next = first;
       first = temp;

       if ( temp == last ) last = prev;
    }
}
```

3. 2 points Taking into account the class `ListIntLinked` with all the methods we developed in class for it and the following method also included:

```
/** Returns true if 'n' is in the list, false otherwise.
  */
public boolean contains( int n )
```

You should implement an static method in a class different from `ListIntLinked` for the following task:

- The method should have as parameters two objects of the class `ListIntLinked`, suppose they are named `listA` and `listB`.
- The method should insert into `listA` those values in `listB` that do not exist in `listA`.
- The insertion in `listA` should be done before the interest point, i.e., before the position of the cursor. The position of the cursor should remain in the same position.
- The method you have to implement **must** use the method `contains()`.
- The position of the cursor in `listB` can be changed.

Solution:

```
public static void union_into_a( ListIntLinked listA, ListIntLinked listB )
{
    listB.begin();
    while(  listB.isValid() ) {
       int i = listB.get();
       if ( ! listA.contains(i) ) listA.insert(i);
       /* for 2016/2017 version
       if ( ! listA.contains(i) ) {
          listA.insert(i);
          listA.next();
          if ( ! listA.isValid() ) listA.end();
       }
       */
       listB.next();
    }
}
```

4. 2.5 points Given an object of the class `StackIntLinked s` and an integer `x`, you have to write an static method in a class different from `StackIntLinked` for doing the following task:

- To compute and return the number of appearances of `x` in `s`,
- but leaving the stack `s` as it was before calling this method.

Solution:

```
public static int countAppearances( StackIntLinked s, int x )
{
   int n = 0;
   if ( ! s.isEmpty() ) {
        int temp = s.pop();
        n = countAppearances( s, x );
        if ( temp == x ) n++;
        s.push(x);
   }
   return n;
}
```

Other version:

```
public static int countAppearances( StackIntLinked s, int x )
{
    StackIntLinked temp = new StackIntLinked();
    int n=0;
    while( ! s.isEmpty() ) {
        if ( s.top() == x ) n++;
        temp.push( s.pop() );
    }
    while( ! temp.isEmpty() ) s.push( temp.pop() );

    return n;
}
```