Joanna Klukowska
joannakl@cs.nyu.edu

Project 2: HTML/CSS Colors - Revisited

Due date: October 8, 11:55PM EST.

You may discuss any of the assignments with your classmates and tutors (or anyone else) but all work for all assignments must be entirely your own . Any sharing or copying of assignments will be considered cheating (this includes posting of partial or complete solutions on Piazza, GitHub or any other public forum). If you get significant help from anyone, you should acknowledge it in your submission (and your grade will be proportional to the part that you completed on your own). You are responsible for every line in your program: you need to know what it does and why. You should not use any data structures and features of Java that have not been covered in class (or the prerequisite class). If you have doubts whether or not you are allowed to use certain structures, just ask your instructor.

In this project you will provide an alternative implementaiton of the program that you implemented for project 1. The major difference in the program is going to be in the data structure used for the `ColorList` class. In this project, you will need to implement and use a sorted linked list instead of the `ArrayList<E>` class in the first project.

You should use the specification of project 1 for the description of design and implementation of the `Color` and `ColorConverter` classes. If your project 1 program did not implement those classes correctly, you should fix as many problems with the implementation as you can.

Objectives The goal of this programming project is for you to master (or at least get practice on) the following tasks:

- working with multi-file programs
- reading data from input files
- using and understanding command line arguments
- working with existing code (written by you, in this case)
- implementing provided interfaces

Start early! This project may not seem like much coding, but debugging always takes time and you will be working with a new data structure.

For the purpose of grading, your project should be in the package called `project2`. This means that each of your submitted source code files should start with a line:

```
package project2;
```

Dataset

See project 1 specification.

User Interface

See project 1 specification.

Data Storage and Organization

Your need to provide an implementation of several classes that store the data and compute the results when the program is executed.

In particular, your program must implement and use the following classes. You may implement additional classes as well, if you wish.

As you are working on your classes, keep in mind that they should be (and will be) tested separately from the rest of your program.

Joanna Klukowska
joannakl@cs.nyu.edu

`Color` Class

See project 1 specification.

`ColorList` Class

The `ColorList` class should be used to store all the `Color` objects whose hexadecimal value and name are provided in the input file. You can use it in a different context in your assignment as well, but you are required to use it to store the data from the input file.[1]

The class should inherit from `OrderedLinkedList<Color>` class. (In project 1, this class was inheriting from the `ArrayList<Color>` class. See below for the details of the `OrderedLinkedList<E>` class.

The rest of the specification for the `ColorList` class remains the same.

The class needs to provide the default constructor that creates an empty list.

```
public ColorList ( )
```

The class should implement

- `public Color getColorByName ( String colorName )`

  method that returns the `Color` object in the list whose color name matches the `colorName` specified by the parameter. This method should be case insensitive. If the method is called with a non-existent color name, the return value should be `null`.

- `public Color getColorByHexValue ( String colorHexValue )`

  method that returns the `Color` object in the list whose hexadecimal value matches the `colorHexValue` specified by the parameter. This method should be case insensitive. If the method is called with a non-existent hexadeciamal value, the return value should be `null`.

You may implement other methods, if you wish.

`OrderedLinkedList<E>` Class

The `OrderedLinkedList<E>` class provides a singly linked list implementation of `OrderedList<E>` interface. For your reference, the interface is included at the end of this specification. You can also download the file from the course website.

The implementation of the `OrderedLinkedList` should guarantee that the elements are stored in a sorted order. It has to be a singly linked list. You can base your design on the `SinglyLinkedList<E>` implementaiton in the textbook (section 3.2.1), but keep in mind that the classs specified in the book does not store elements in a sorted order.

Your `Node<E>` class should be an internal class of the `OrderedLinkedList<E>` class.

HINT: Section 3.5 in the book discusses the ideas of equivalence testing of lists. The code in that section may be very handy for the implementation of the `equals()` method. Make sure to give proper credit to your sources, if you decide to base your code on the code in the textbook.

You may implement other methods, if you wish.

`ColorConverter` Class

See project 1 specification.

[1] "Why should I store the data from the input file in memory rather than reading it from the input file whenever I need to?", you might ask. Well, reading from files stored on disks is MUCH slower than reading from program's memory. How much slower? Take a look at these numbers *Latency Numbers Every Programmer Should know*, https://gist.github.com/jboner/2841832.

Joanna Klukowska
joannakl@cs.nyu.edu

Programming Rules

You should follow the rules outlined in the document *Code conventions* posted on the course website at https://joannakl.github.io/cs102_f17/notes/CodeConventions.pdf.

The data file should be read only once! Your program needs to store the data in memory resident data structures.

You may use any exception-related classes.

You may use any classes to handle the file I/O, but probably the simplest ones are `File` and `Scanner` classes. You are responsible for knowing how to use the classes that you select.

Working on This Assignment

You should start right away!

You should modularize your design so that you can test it regularly.

Make sure that at all times you have a working program. You can implement methods that perform one task at a time. This way, if you run out of time, at least parts of your program will be functioning properly.

You should make sure that your program's results are consistent with what is described in this specification by running the program on carefully designed test inputs and examining the outputs produced to make sure they are correct. The goal in doing this is to try to find the mistakes you have most likely made in your code.

You should backup your code after each time you spend some time working on it. Save it to a flash drive, email it to yourself, upload it to your Google drive, do anything that gives you a second (or maybe third copy). Computers tend to break just a few days or even a few hours before the due dates - make sure that you have working code if that happens. (A second copy of the files on the same computer is a good idea to keep multiple versions, but it is NOT a good backup since you do not have access to it if there are problems with your computer.)

Grading

If your program does not compile or if it crashes (almost) every time it is run, you will get a zero on the assignment.

If the program does not adhere to the specification, the grade will be low and will depend on how easy it is to figure out what the program is doing.

50 points program and class correctness: the correct values and format of output, correct behavior of methods, handling of invalid arguments

25 points design and implementation of the required classes and any additional classes

25 points proper documentation, program style and format of submission

How and What to Submit

For the purpose of grading, your project should be in the package called `project2`. This means that each of your submitted source code files should start with a line:

```
package project2;
```

Your should submit all your source code files (the ones with .java extensions only) in a single **zip** file to Gradescope. The files that should be submitted are `Color.java`, `ColorList.java`, `OrderedList.java`, `OrderedLinkedList.java`, `ColorConverter.java`. If you implement any other classes, they should be included as well.

You can produce a zip file directly from Eclipse:

- right click on the name of the package (inside the `src` folder) and select Export...
- under General pick Archive File and click Next

Joanna Klukowska
joannakl@cs.nyu.edu

- in the window that opens select appropriate files and settings:
  - in the right pane pick ONLY the files that are actually part of the project, but make sure that you select all files that are needed
  - in the left pane, make sure that no other directories are selected
  - click Browse and navigate to a location that you can easily find on your system (Desktop or folder with the course materials or ...)
  - in Options select "Save in zip format", "Compress the contents of the file" and "Create only selected directories"
- click Finish

Appendix

```
1  package project2;
2
3  /**
4   * A sorted collection.  The element in this list are stored in order determined
5   * by the natural ordering of those elements (i.e., the {@code compareTo()} method
6   * defined in the elements' class).
7   * The user can access elements by their integer index (position in
8   * the list), and search for elements in the list.<p>
9   *
10  * The lists implementing this interface allow duplicate elements.
11  * They do not allow null elements.<p>
12  *
13  * @param <E> the type of elements in this list
14  *
15  * @author  Joanna Klukowska
16  */
17
18 public interface OrderedList<E extends Comparable<E>> extends Cloneable {
19
20          /**
21       * Adds the specified element to  this list in a sorted order.
22       *
23       * <p>The element added must implement Comparable<E> interface. This list
24       * does not permit null elements.
25       *
26       * @param e element to be appended to this list
27       * @return <tt>true</tt> if this collection changed as a result of the call
28       * @throws ClassCastException if the class of the specified element
29       *          does not implement Comparable<E>
30       * @throws NullPointerException if the specified element is null
31       */
32          public boolean add(E e);
33
34      /**
35       * Removes all of the elements from this list.
36       * The list will be empty after this call returns.
37       */
38          public void clear();
39
40      /**
41       * Returns a shallow copy of this list. (The elements
42       * themselves are not cloned.)
43       *
```

Joanna Klukowska
joannakl@cs.nyu.edu

```
44      * @return a shallow copy of this list instance
45      */
46         public Object clone();
47
48     /**
49      * Returns <tt>true</tt> if this list contains the specified element.
50      *
51      * @param o element whose presence in this list is to be tested
52      * @return <tt>true</tt> if this list contains the specified element
53      * @throws ClassCastException if the type of the specified element
54      *         is incompatible with this list
55      * @throws NullPointerException if the specified element is null
56      */
57         public boolean contains(Object o);
58
59     /**
60      * Compares the specified object with this list for equality.  Returns
61      * {@code true} if and only if the specified object is also a list, both
62      * lists have the same size, and all corresponding pairs of elements in
63      * the two lists are <i>equal</i>.  (Two elements {@code e1} and
64      * {@code e2} are <i>equal</i> if {@code e1.equals(e2)}.)
65      * In other words, two lists are defined to be
66      * equal if they contain the same elements in the same order.<p>
67      *
68      * @param o the object to be compared for equality with this list
69      * @return {@code true} if the specified object is equal to this list
70      */
71         public boolean equals(Object o);
72
73     /**
74      * Returns the element at the specified position in this list.
75      *
76      * @param index index of the element to return
77      * @return the element at the specified position in this list
78      * @throws IndexOutOfBoundsException if the index is out of range
79      * <tt>(index < 0 || index >= size())</tt>
80      */
81     public E get(int index);
82
83     /**
84      * Returns the index of the first occurrence of the specified element
85      * in this list, or -1 if this list does not contain the element.
86      *
87      * @param o element to search for
88      * @return the index of the first occurrence of the specified element in
89      *         this list, or -1 if this list does not contain the element
90      */
91         public int indexOf (Object o);
92
93     /**
94      * Removes the element at the specified position in this list.  Shifts any
95      * subsequent elements to the left (subtracts one from their indices if such exist).
96      * Returns the element that was removed from the list.
97      *
98      * @param index the index of the element to be removed
```

Joanna Klukowska
joannakl@cs.nyu.edu

```
99       * @return the element previously at the specified position
100      * @throws IndexOutOfBoundsException  if the index is out of range
101      * <tt>(index < 0 || index >= size())</tt>
102      */
103
104        public E remove (int index);
105
106     /**
107      * Removes the first occurrence of the specified element from this list,
108      * if it is present.  If this list does not contain the element, it is
109      * unchanged.  More formally, removes the element with the lowest index
110      * {@code i} such that
111      * <tt>(o.equals(get(i))</tt>
112      * (if such an element exists).  Returns {@code true} if this list
113      * contained the specified element (or equivalently, if this list
114      * changed as a result of the call).
115      *
116      * @param o element to be removed from this list, if present
117      * @return {@code true} if this list contained the specified element
118      * @throws ClassCastException if the type of the specified element
119      *         is incompatible with this list
120      * @throws NullPointerException if the specified element is null and this
121      *         list does not permit null elements
122      */
123        public boolean remove (Object o);
124
125     /**
126      * Returns the number of elements in this list.
127      *
128      * @return the number of elements in this list
129      */
130        public int size();
131
132     /**
133      * Returns a string representation of this list.  The string
134      * representation consists of a list of the list's elements in the
135      * order they are stored in this list, enclosed in square brackets
136      * (<tt>"[]"</tt>).  Adjacent elements are separated by the characters
137      * <tt>", "</tt> (comma and space).  Elements are converted to strings
138      * by the {@code toString()} method of those elements.
139      *
140      * @return a string representation of this list
141      */
142        public String toString();
143
144 }
145
```