CSE 143 Assignment 2 (HTMLManager) Score Sheet

Student(s): akshit <akshit@uw.edu>

Graded by: Melissa Medsker <medskm@cs.washington.edu>

28 / 30 : Total Score

28 / 30 : Correctness

- 2 / 2 : constructor
- **3 / 3 : removeAll**
  - 1 / 1 : attempt
  - 2 / 2 : correct
- 1 / 1 : add
- **2 / 2 : getTags**
  - 1 / 1 : attempt
  - 1 / 1 : correct (returns a copy)
- **6 / 6 : fixHTML**
  - 1 / 1 : self-closing
  - 1 / 1 : valid HTML
  - 1 / 1 : extra closing tag
  - 1 / 1 : extra opening tag
  - 1 / 1 : resetting field
  - 1 / 1 : passes all provided test cases
- 2 / 2 : HTMLManagerTest
  - *Excellent testing program!*
- **1 / 2 : constructor, add, and remove throw proper exceptions**
  - 1 / 1 : attempt: at least 1 out of the 3 correctly throws the exceptions
  - 0 / 1 : correct: all 3 throw the proper exceptions
    - *-1: See blue*
- **3 / 3 : fixHTML is clear and concise and works in most cases**
  - 1 / 1 : Creates correct structures
  - 2 / 2 : Uses control flow well
- 2 / 2 : interfaces and generics
- **2 / 3 : comments**
  - 1 / 1 : attempt to comment
  - 1 / 2 : well documented code
    - *-1: See yellow*
- 4 / 4 : otherwise good style
  - *-0: See green and orange*
  - *-0: Lines should be no more than 80 characters long*

Lateness and Other Deductions

	Thu 2016/10/13 11:30pm	Due
	Thu 2016/10/13 07:50pm	Submitted (on time)
0	Late days used on this assignment	
0	Lateness deduction	
	Other deductions	

Overall comments:

Great work Akshit! Overall, you did a great job handing different edge cases for your HTMLManager and providing clear and descriptive documentation. Make sure to implem

Annotations: HTMLManager.java

```java
/**
 * @author Akshit Patel
 * @Date 10/12/2016
 * CSE 143D DC
 * TA: Melissa Medsker
 * HW #2 File #1 HTMLManager
 */

import java.util.*; // Queues & Lists.

/**
 * This class manages the HTMLTags by providing useful methods like adding the
 * tags, removing all specific HTMLTags, get the tags and a method that fixes
 * potential errors in the HTML.
 *
 */
public class HTMLManager {

    /**
     * This field stores the HTMLTags to be processed or managed.
     */
    private Queue<HTMLTag> tagStorage;

    /**
     * This constructor takes in HTMLTags that make up an HTML page.
     *
     * @param page Queue of HTMLTags to be processed for using other methods.
     * @throws IllegalArgumentException if the Queue passed is null.
     *
     * PostCondition: The Queue of HTMLTags passed remains in its original
     * state.
     */
    public HTMLManager(Queue<HTMLTag> page) {
        if (page.equals(null)) {
            throw new IllegalArgumentException("The HTMLTags can't be null!");
        }
        this.tagStorage = new LinkedList<HTMLTag>();// initialize the field.
        int size = page.size();
        for (int i = 0; i < size; i++) {
            this.tagStorage.add(page.peek());// add the tag.
            page.add(page.remove());// update the queue to get next tag.
        }
    }

    /**
     * This method adds the given HTMLTag to the end of the HTMLTags being
     * managed.
     *
     * @param tag HTMLTag that needs to be added to the already present
     * HTMLtags.
     * @throws IllegalArgumentException if the HTMLTag passed is null.
     */
    public void add(HTMLTag tag) {
        if (tag == null) {
            throw new IllegalArgumentException();
        }
        this.tagStorage.add(tag);
    }

    /**
     * This method removes all occurrences of the given HTMLTag of specific type
     * like opening or closing "b" from the already present HTMLtags.
     *
     * @param tag HTMLTag that needs to be removed from the HTMLTags.
     * @throws IllegalArgumentException if the HTMLTag passed is null.
     *
     * PostCondition: The order of HTMLTags that are managed is not changed,
     * only the unwanted tags are removed and there place is taken by next
     * useful tag.
     */
    public void removeAll(HTMLTag tag) {
        if (tag == null) {
            throw new IllegalArgumentException();
        }
        int size = this.tagStorage.size();
        for (int i = 0; i < size; i++) {
            // if statement to check if the current tag equals the one to
            // remove.
            if (this.tagStorage.peek().equals(tag)) {
                this.tagStorage.remove();// remove the tag.
            } else {
                // since the match is not found, add the tag back to preserve
                // order.
                this.tagStorage.add(this.tagStorage.remove());
            }
        }
    }

    /**
     * This method helps to get HTMLTags being managed as an ArrayList of
     * HTMLTags.
     *
     * @return ArrayList of HTMLTags used to manage or that have been processed.
     */
    public List<HTMLTag> getTags() {
        int resultSize = this.tagStorage.size();
        List<HTMLTag> resultList = new ArrayList<HTMLTag>();
        // For loop to add the contents to the list.
        for (int i = 0; i < resultSize; i++) {
            resultList.add(i, this.tagStorage.peek());
            this.tagStorage.add(this.tagStorage.remove());// restore the order.
        }
        return resultList;// return the List processed.
    }

    /**
     * This method helps to fix the HTMLTags used in HTML if there were any
     * missing or extra tags. The opening tags will be closed and self closing
     * tags will be added. However, if there is an closing tag then the method
     * will fix the HTML until there is a matching opening tag else if not found
     * the closing tag will be discarded.
     *
     * PostCondition: The intended order and format of the HTML is preserved.
     */
    public void fixHTML() {
        Queue<HTMLTag> output = new LinkedList<HTMLTag>();// stores the output.
        Stack<HTMLTag> oTags = new Stack<HTMLTag>();// keeps track of open tags.
        // while loop to fix HTML until no every tag is checked.
        while (!this.tagStorage.isEmpty()) {
            // if statement to check for opening tag.
            if (this.tagStorage.peek().isOpening()) {
                oTags.push(this.tagStorage.peek()); // store the tag for later.
                output.add(this.tagStorage.remove());// add it to result.
            } else if (this.tagStorage.peek().isSelfClosing()) {
                output.add(this.tagStorage.remove());// add to the result.
            } else if (this.tagStorage.peek().isClosing()) {
                // if the closing tag matches the opening then add it to the
                // correct result.
                if (!oTags.isEmpty()
                        && oTags.peek().matches(this.tagStorage.peek())) {
                    output.add(this.tagStorage.remove());
                    oTags.pop();
                } else {
                    // if the matching is not found then add the matching from
                    // the storage till the matching is found.
                    while (!oTags.isEmpty()
                            && !oTags.peek().matches(this.tagStorage.peek())) {
```

-1: Remember that in order to check if an object is null, you should use "==" - if you dereference a null object with "." to access a method or field, you will get a NullPointerException otherwise

Great work restoring the Queue parameter here!

-0: You should store this next tag as a variable from page.remove() since you will remove each tag for each iteration of the loop

-0: This exception check for null tags is used in multiple places throughout the file and should be factored out to reduce redundancy

Excellent comment!

-0: You should store the result of tagStorage.remove() as a variable to avoid an extra call to tagStorage.peek()

-1: This is an implementation detail - the client only knows that a List object is being returned, but shouldn't depend on an ArrayList being returned

Great fixHTML comment!

It's not clear what this "no" means here

-0: This variable name should be more descriptive - what does "oTags" refer to?

-0: You should store your tags as variables in this method rather than calling tagStorage.peek() repeatedly to get the same value

Excellent use of inline comments!

```
138                             // add to the result & update the storage.
139                             output.add(oTags.pop().getMatching());
140                         }
141                         // if the storage is empty then no opening found.
142                         if (oTags.isEmpty()) {
143                             this.tagStorage.remove();// remove the unwanted.
144                         }
145                     }
146                 }
147             }
148             // if there are opening tags remaining in the storage then add the
149             // matching closing tag.
150             while (!oTags.isEmpty()) {
151                 output.add(oTags.pop().getMatching());
152             }
153             this.tagStorage = output;
154         }
155 
156     }
157 
158 
```

-0: Extra blank line (155)

Annotations: HTMLManagerTest.java

```java
/**
 * @author Akshit Patel
 * @Date 10/12/2016
 * CSE 143D DC
 * TA: Melissa Medsker
 * HW #2 File #2 HTMLManagerTest
 */

import java.util.*; // Queues & List.

/**
 * This program tests the removeAll() method of the HTMLManager class by
 * comparing the result with the correct output.
 */
public class HTMLManagerTest {

    public static void main(String[] args) {
        // Queue of tags to remove.
        Queue<HTMLTag> tags = new LinkedList<HTMLTag>();
        tags.add(new HTMLTag("ul", HTMLTagType.OPENING)); // <ul>
        tags.add(new HTMLTag("li", HTMLTagType.OPENING)); // <li>
        tags.add(new HTMLTag("br", HTMLTagType.SELF_CLOSING)); // <br/>
        tags.add(new HTMLTag("li", HTMLTagType.OPENING)); // <li>
        tags.add(new HTMLTag("br", HTMLTagType.SELF_CLOSING)); // <br/>
        tags.add(new HTMLTag("li", HTMLTagType.CLOSING)); // </li>
        tags.add(new HTMLTag("li", HTMLTagType.OPENING)); // <li>
        tags.add(new HTMLTag("li", HTMLTagType.CLOSING)); // </li>
        // give the queue to the HTMLManager.
        HTMLManager manager = new HTMLManager(tags);
        testOpening(manager);// test for opening tags.
        testClosing(manager); // test for closing tags
        testSelfClosing(manager);// test for self closing tags.
        testEmpty(manager);// test for empty situations.
    }

    /**
     * This method tests if the the removeAll() method can remove all the
     * opening tags of specific HTMLTag from the queue given.
     *
     * @param manager HTMLManager to access the removeAll() method and getTags()
     * method.
     */
    public static void testOpening(HTMLManager manager) {
        // List to store correct output.
        List<HTMLTag> correct = new ArrayList<HTMLTag>();
        correct.add(new HTMLTag("ul", HTMLTagType.OPENING)); // <ul>
        correct.add(new HTMLTag("br", HTMLTagType.SELF_CLOSING)); // <br/>
        correct.add(new HTMLTag("br", HTMLTagType.SELF_CLOSING)); // <br/>
        correct.add(new HTMLTag("li", HTMLTagType.CLOSING)); // </li>
        correct.add(new HTMLTag("li", HTMLTagType.CLOSING)); // </li>
        System.out.println("Test 1 initiated to remove <li>");
        // remove <li> from the user queue.
        manager.removeAll(new HTMLTag("li", HTMLTagType.OPENING));
        testAnalysis(1, correct, manager);// evaluate results.
    }

    /**
     * This method tests if the the removeAll() method can remove the closing
     * tags of specific HTMLTag from the queue given.
     *
     * @param manager HTMLManager to access the removeAll() method and getTags()
     * method.
     */
    public static void testClosing(HTMLManager manager) {
        List<HTMLTag> correct = new ArrayList<HTMLTag>();
        correct.add(new HTMLTag("ul", HTMLTagType.OPENING)); // <ul>
        correct.add(new HTMLTag("br", HTMLTagType.SELF_CLOSING)); // <br/>
        correct.add(new HTMLTag("br", HTMLTagType.SELF_CLOSING)); // <br/>
        System.out.println("Test 2 initiated to remove </li>");
        // remove </li> from the user queue.
        manager.removeAll(new HTMLTag("li", HTMLTagType.CLOSING));
        testAnalysis(2, correct, manager);// evaluate results.
    }

    /**
     * This method tests if the the removeAll() method can remove the
     * self-closing tags of specific HTMLTag from the queue given.
     *
     * @param manager HTMLManager to access the removeAll() method and getTags()
     * method.
     */
    public static void testSelfClosing(HTMLManager manager) {
        List<HTMLTag> correct = new ArrayList<HTMLTag>();
        correct.add(new HTMLTag("ul", HTMLTagType.OPENING)); // <ul>
        System.out.println("Test 3 initiated to remove <br/>");
        // remove <br/> from the user queue.
        manager.removeAll(new HTMLTag("br", HTMLTagType.SELF_CLOSING));
        testAnalysis(3, correct, manager);// evaluate results.
    }

    /**
     * This method tests if the the removeAll() method can remove the last
     * remaining tag of specific HTMLTag from the queue given.
     *
     * @param manager HTMLManager to access the removeAll() method and getTags()
     * method.
     */
    public static void testEmpty(HTMLManager manager) {
        List<HTMLTag> correct = new ArrayList<HTMLTag>();
        System.out.println("Test 4 initiated to remove <ul>");
        // remove <ul> from the user queue.
        manager.removeAll(new HTMLTag("ul", HTMLTagType.OPENING));
        testAnalysis(4, correct, manager);// evaluate results.
    }

    /**
     * This method evaluates the results of the tests done on the user queue by
     * comparing them to the correct result.
     *
     * @param num the int representation of the test done.
     * @param correct The correct List of HTMLTags after the removeAll() method.
     * @param manager HTMLManager to access the getTags() method.
     */
    private static void testAnalysis(int num, List<HTMLTag> correct,
        HTMLManager manager) {
        int error = 0;// error counter.
        List<HTMLTag> clientList = manager.getTags();// get the user result.
        if (clientList.size() == correct.size()) {
            // for statement to check for any potential errors.
            for (int i = 0; i < correct.size(); i++) {
                if (!clientList.get(i).equals(correct.get(i))) {
                    error++;
                }
            }
        }
        if (error > 0 || correct.size() != clientList.size()) {
            System.out.println("Your output: " + clientList.toString());
            System.out.println("Correct output: " + correct.toString());
            System.out.println("Test " + num + " Failed!");
            System.out.println();
        } else {
            System.out.println("Test " + num + " passed!");
            System.out.println();
        }
    }
}
```

Note that multi-line method headers should be broken apart such that the second line is indented to align with the first parameter after the "("