Homework 8: Set Implementation on Queue

- **Name**: Saahib Mohammed
- **Dot Number**: Mohammed.206
- **Due Date**: sept 7 1:50pm

Preparation

Previous students would have wanted you to know the following before you get started (based on 1 review):

- Estimated time to complete the assignment: 45 minutes
- Most common emotion before starting the assignment: ??
- Most common emotion while completing the assignment: ??
- Most common emotion after completing the assignment: ??

If the information above is incomplete, you can help by providing your own feedback after completing this assignment.

Problems

This homework is necessary preparation for the lab. Make sure you type your answers in files you bring to the lab so that you will not have to waste time entering your code during the lab.

Problem 1

> Implement the following method that, given a queue and an entry of type T, searches for the given entry in the given queue and, if it finds it, moves that entry to the front of the queue.

```
/**
 * Finds {@code x} in {@code q} and, if such exists, moves it to the front
 * of {@code q}.
 *
 * @param <T>
 *            type of {@code Queue} entries
 * @param q
 *            the {@code Queue} to be searched
 * @param x
 *            the entry to be searched for
 * @updates q
 * @ensures <pre>
 * perms(q, #q)  and
 * if <x> is substring of q
 *  then <x> is prefix of q
 * </pre>
 */
private static <T> void moveToFront(Queue<T> q, T x) {

    for(T y : q){
        if(y == x){
```

```
            q.enque(y)
            for(T v : q){
            if(v != x){
             q.enque(v)
            }

            }
        }
    }
}
```

> Note that moveToFront is a static, generic method: it is parameterized by the type T of the entries in the queue. You can use the type T wherever you need to declare a variable that refers to an object of type T.

> Pay attention to the contract. There is no requires clause. If you have trouble reading and understanding the *ensures* clause, be sure to ask for help.

Problem 2

> Develop a complete test plan for the Set constructor and kernel methods: add, remove, removeAny, contains, and size and enter them in SetTest. You can find some more information on how to effectively test removeAny here.

```
import static org.junit.Assert.assertEquals;

import org.junit.Test;

import components.set.Set;

/**
 * JUnit test fixture for {@code Set<String>}'s constructor and kernel methods.
 *
 * @author Saahib Mohammed
 *
 */
public abstract class SetTest {

    /**
     * Invokes the appropriate {@code Set} constructor and returns the result.
     *
     * @return the new set
     * @ensures constructorTest = {}
     */
    protected abstract Set<String> constructorTest();

    /**
     * Invokes the appropriate {@code Set} constructor and returns the result.
     *
     * @return the new set
     * @ensures constructorRef = {}
```

```
     */
    protected abstract Set<String> constructorRef();

    /**
     * Creates and returns a {@code Set<String>} of the implementation under
     * test type with the given entries.
     *
     * @param args
     *            the entries for the set
     * @return the constructed set
     * @requires [every entry in args is unique]
     * @ensures createFromArgsTest = [entries in args]
     */
    private Set<String> createFromArgsTest(String... args) {
        Set<String> set = this.constructorTest();
        for (String s : args) {
            assert !set.contains(s) : "Violation of: every entry in args is
unique";
            set.add(s);
        }
        return set;
    }

    /**
     * Creates and returns a {@code Set<String>} of the reference implementation
     * type with the given entries.
     *
     * @param args
     *            the entries for the set
     * @return the constructed set
     * @requires [every entry in args is unique]
     * @ensures createFromArgsRef = [entries in args]
     */
    private Set<String> createFromArgsRef(String... args) {
        Set<String> set = this.constructorRef();
        for (String s : args) {
            assert !set.contains(s) : "Violation of: every entry in args is
unique";
            set.add(s);
        }
        return set;
    }

  @Test
public void testAdd() {
Set<String> set = constructorTest();
set.add("A");
set.add("B");
set.add("C");
assertEquals(3, set.size());
}
@Test
public void testRemove() {
Set<String> set = createFromArgsTest("A", "B", "C");
```

```
set.remove("B");
assertEquals(2, set.size());
}
@Test
public void testRemoveAny() {
Set<String> set = createFromArgsTest("A", "B", "C");
String removed = set.removeAny();
assertEquals(true, set.contains(removed));
assertEquals(2, set.size());
}
@Test
public void testContains() {
Set<String> set = createFromArgsTest("A", "B", "C");
assertEquals(true, set.contains("A"));
assertEquals(true, set.contains("B"));
assertEquals(true, set.contains("C"));
assertEquals(false, set.contains("D"));
}
@Test
public void testSize() {
Set<String> set = createFromArgsTest("A", "B", "C");
assertEquals(3, set.size());
set.remove("B");
assertEquals(2, set.size());
set.removeAny();
assertEquals(1, set.size());
set.remove("A");
assertEquals(0, set.size());


}
```

Submission

If you have completed the assignment using this template, we recommend that you convert it to a PDF before submission. If you're not sure how, check out this Markdown to PDF guide.