The Ohio State University

PROJECT 4: SET ON BINARY SEARCH TREES

Daniil Gofman

Ansh Pachauri

SW 2: Dev & Dsgn

Paolo Bucci

Yiyang Chen

Shivam Gupta

September 29, 2023

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

import org.junit.Test;

import components.set.Set;

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

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

    /**
     * Invokes the appropriate {@code Set} constructor for the reference
     * implementation 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 for empty constructor.
     */
    @Test
    public final void testConstructorEmpty() {
        /*
         * Set up variables
         */
        Set<String> s = this.constructorTest();
        Set<String> sExpected = this.constructorRef();
        /*
         * Assert that values of variables match expectations
         */
        assertEquals(sExpected, s);
    }

    /**
     * Test for non-empty constructor.
     */
    @Test
    public final void testConstructorNonEmpty() {
        /*
         * Set up variables
         */
        Set<String> s = this.createFromArgsTest("Apple", "Banana");
        Set<String> sExpected = this.createFromArgsRef("Apple", "Banana");
        /*
         * Assert that values of variables match expectations
         */
        assertEquals(sExpected, s);
    }

    /**
     * Test add boundary test case.
     */
    @Test
    public final void testAddEmpty1() {
        /*
         * Set up variables
         */
        Set<String> s = this.createFromArgsTest();
        Set<String> sExpected = this.createFromArgsRef("Apple");
        /*
         * Call methods under the test.
         */
        s.add("Apple");
        /*
         * Assert that values of variables match expectations
         */
        assertEquals(sExpected, s);
    }

    /**
     * Test add boundary test case.
     */
    @Test
    public final void testAddNonEmpty1() {
        /*
         * Set up variables
         */
        Set<String> s = this.createFromArgsTest("Apple");
        Set<String> sExpected = this.createFromArgsRef("Apple", "Banana");
        /*
         * Call methods under the test.
         */
        s.add("Banana");
        /*
         * Assert that values of variables match expectations
         */
        assertEquals(sExpected, s);
    }

    /**
```

```
     * Test add routine test case.
     */
    @Test
    public final void testAddNonEmpty2() {
        /*
         * Set up variables
         */
        Set<String> s = this.createFromArgsTest("Apple", "Banana");
        Set<String> sExpected = this.createFromArgsRef("Apple", "Banana",
                "Orange");
        /*
         * Call methods under the test.
         */
        s.add("Orange");
        /*
         * Assert that values of variables match expectations
         */
        assertEquals(sExpected, s);
    }

    /**
     * Test add routine test case.
     */
    @Test
    public final void testAddNonEmpty3() {
        /*
         * Set up variables
         */
        Set<String> s = this.createFromArgsTest("Apple", "Banana",
"Orange",
                "Watermelon", "Kiwi");
        Set<String> sExpected = this.createFromArgsRef("Apple", "Banana",
                "Orange", "Watermelon", "Kiwi", "Avocado");
        /*
         * Call methods under the test.
         */
        s.add("Avocado");
        /*
         * Assert that values of variables match expectations
         */
        assertEquals(sExpected, s);
    }

    /**
     * Test remove boundary test case.
     */
    @Test
    public final void testRemoveEmpty1() {
        /*
         * Set up variables
```

```java
         */
        Set<String> s = this.createFromArgsTest("Apple");
        Set<String> sExpected = this.createFromArgsRef();
        /*
         * Call methods under the test.
         */
        String temp = s.remove("Apple");
        /*
         * Assert that values of variables match expectations
         */
        assertEquals(sExpected, s);
        assertEquals("Apple", temp);
    }

    /**
     * Test remove routine test case.
     */
    @Test
    public final void testRemoveNonEmpty1() {
        /*
         * Set up variables
         */
        Set<String> s = this.createFromArgsTest("Apple", "Banana");
        Set<String> sExpected = this.createFromArgsRef("Apple");
        /*
         * Call methods under the test.
         */
        String temp2 = s.remove("Banana");
        /*
         * Assert that values of variables match expectations
         */
        assertEquals(sExpected, s);
        assertEquals("Banana", temp2);
    }

    /**
     * Test remove routine test case.
     */
    @Test
    public final void testRemoveNonEmpty2() {
        /*
         * Set up variables
         */
        Set<String> s = this.createFromArgsTest("Apple", "Banana", "Orange",
                "Watermelon", "Kiwi", "Avocado");
        Set<String> sExpected = this.createFromArgsRef("Apple", "Orange",
                "Watermelon", "Kiwi", "Avocado");
        /*
         * Call methods under the test.
         */
        String temp = s.remove("Banana");
        /*
         * Assert that values of variables match expectations
         */
        assertEquals(sExpected, s);
        assertEquals("Banana", temp);
    }

    /**
     * Test size routine case.
     */
    @Test
    public final void testSize() {
        /*
         * Set up variables
         */
        Set<String> s = this.createFromArgsTest("Pizza", "Steak", "Sushi",
                "Salmon", "Grilled Pork");
        Set<String> sExpected = this.createFromArgsRef("Pizza", "Steak",
                "Sushi", "Salmon", "Grilled Pork");
        /*
         * Call method under test
         */
        int setSize = s.size();
        final int expectedSize = 5;
        /*
         * Assert that values of variables match expectations
         */
        assertEquals(sExpected, s);
        assertEquals(expectedSize, setSize);
    }

    /**
     * Test size routine case.
     */
    @Test
    public final void testSizeNonEmpty() {
        /*
         * Set up variables
         */
        Set<String> s = this.createFromArgsTest("Pizza", "Steak");
        Set<String> sExpected = this.createFromArgsRef("Pizza", "Steak");
        /*
         * Call method under test
         */
        int setSize = s.size();
        /*
         * Assert that values of variables match expectations
         */
```

```java
        assertEquals(2, setSize);
        assertEquals(sExpected, s);
    }

    /**
     * Test size boundary case.
     */
    @Test
    public final void testSizeEmpty() {
        /*
         * Set up variables
         */
        Set<String> s = this.constructorTest();
        Set<String> sExpected = this.constructorRef();
        /*
         * Call method under test
         */
        final int expectedSize = 0;
        int setSize = s.size();
        /*
         * Assert that values of variables match expectations
         */
        assertEquals(expectedSize, setSize);
        assertEquals(sExpected, s);
    }

    /**
     * Test contains routine case.
     */
    @Test
    public final void testContainsNonEmpty() {
        /*
         * Set up variables and call method under test
         */
        Set<String> s = this.createFromArgsTest("Pizza", "Steak",
                "Mac and Cheese");
        /*
         * Call methods under the test.
         */
        final boolean expectedValue = false;
        boolean isIcecream = s.contains("Ice cream");
        /*
         * Assert that values of variables match expectations
         */
        assertEquals(expectedValue, isIcecream);
    }

    /**
     * Test contains boundary case.
     */
    @Test
    public final void testContainsEmpty() {
        /*
         * Set up variables and call method under test
         */
        Set<String> s = this.createFromArgsTest();
        /*
         * Call methods under the test.
         */
        final boolean expectedValue = false;
        boolean isSteak = s.contains("Steak");
        /*
         * Assert that values of variables match expectations
         */
        assertEquals(expectedValue, isSteak);
    }

    /**
     * Test removeAny routine case.
     */
    @Test
    public final void testRemoveAnyNonEmpty() {
        /*
         * Set up variables and call method under test
         */
        Set<String> s = this.createFromArgsTest("Pizza", "Steak",
                "Mac and Cheese");
        Set<String> sExpected = this.createFromArgsRef("Pizza", "Steak",
                "Mac and Cheese");
        /*
         * Call methods under the test.
         */
        String removedValue = s.removeAny();
        String removedValueExpected = sExpected.remove(removedValue);

        /*
         * Assert that values of variables match expectations
         */
        assertEquals(removedValue, removedValueExpected);
        assertEquals(s, sExpected);
    }

    /**
     * Test removeAny boundary case.
     */
    @Test
    public final void testRemoveAnyLeavingEmpty() {
        /*
         * Set up variables and call method under test
         */
        Set<String> s = this.createFromArgsTest("Steak");
        Set<String> sExpected = this.createFromArgsRef("Steak");
        /*
         * Call methods under the test.
         */
        String removedValue = s.removeAny();
        assertTrue(sExpected.contains(removedValue));
        String removedValueExpected = sExpected.remove(removedValue);

        /*
         * Assert that values of variables match expectations
         */
        assertEquals(removedValue, removedValueExpected);
        assertEquals(s, sExpected);
    }

}

```