CSE 2231 – Software 2: Software Development and Design

Professor: Rob LaTour

Project #4: Set on Binary Search Trees

The Ohio State University

College of Engineering

Columbus, Ohio

```
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 Danny Kan (kan.74@osu.edu)
 * @author Jatin Mamtani (mamtani.6@osu.edu)
 *
 */
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;
}

/*
 * Complete and Systematic Test Cases:
 */

/**
 * Testing the no-argument constructor.
 */
@Test
public final void testNoArgumentConstructor() {
    Set<String> sActual = this.constructorTest();
    Set<String> sExpected = this.constructorRef();
    assertEquals(sExpected, sActual);
}

/*
 * Testing .add() in this section:=
 */

/**
 * Testing .add() to empty {@code Set<String>}.
 */
@Test
public final void testAddToEmpty() {
    Set<String> sActual = this.createFromArgsTest();
    Set<String> sExpected = this.createFromArgsRef("x");
    sActual.add("x");
    assertEquals(sExpected, sActual);
}

/**
 * Testing .add() to non-empty {@code Set<String>} with three (3)
 * {@code String}.
 */
@Test
public final void testAddToNonEmptyV1() {
    // abcdefghijklmnopqrstuvwxyz.
    Set<String> sActual = this.createFromArgsTest("j", "i", "k");
    Set<String> sExpected = this.createFromArgsRef("j", "i", "k", "h");
    sActual.add("h");
    assertEquals(sExpected, sActual);
}

/**
 * Testing .add() to non-empty {@code Set<String>} with three (3)
 * {@code String}.
 */
@Test
public final void testAddToNonEmptyV2() {
    // abcdefghijklmnopqrstuvwxyz.
    Set<String> sActual = this.createFromArgsTest("j", "i", "k");
    Set<String> sExpected = this.createFromArgsRef("j", "i", "k", "l");
    sActual.add("l");
    assertEquals(sExpected, sActual);
}
```

```
/*
 * Testing .remove() in this section:=
 */

/**
 * Testing .remove() to empty {@code Set<String>}.
 */
@Test
public final void testRemoveToEmpty() {
    Set<String> sActual = this.createFromArgsTest("x");
    Set<String> sExpected = this.createFromArgsRef();
    assertEquals("x", sActual.remove("x"));
    assertEquals(sExpected, sActual);
}

/**
 * Testing .remove() to non-empty {@code Set<String>} with three (3)
 * {@code String}.
 */
@Test
public final void testRemoveToNonEmptyV1() {
    // abcdefghijklmnopqrstuvwxyz.
    Set<String> sActual = this.createFromArgsTest("j", "i", "k");
    Set<String> sExpected = this.createFromArgsRef("j", "k");
    assertEquals("i", sActual.remove("i"));
    assertEquals(sExpected, sActual);
}

/**
 * Testing .remove() to non-empty {@code Set<String>} with three (3)
 * {@code String}.
 */
```

```
@Test
public final void testRemoveToNonEmptyV2() {
    // abcdefghijklmnopqrstuvwxyz.
    Set<String> sActual = this.createFromArgsTest("j", "i", "k");
    Set<String> sExpected = this.createFromArgsRef("j", "i");
    assertEquals("k", sActual.remove("k"));
    assertEquals(sExpected, sActual);
}

/**
 * Testing .remove() to non-empty {@code Set<String>} with three (3)
 * {@code String}.
 */
@Test
public final void testRemoveToNonEmptyV3() {
    // abcdefghijklmnopqrstuvwxyz.
    Set<String> sActual = this.createFromArgsTest("j", "i", "k");
    Set<String> sExpected = this.createFromArgsRef("i", "k");
    assertEquals("j", sActual.remove("j"));
    assertEquals(sExpected, sActual);
}

/*
 * Testing .removeAny() in this section:=
 */

/**
 * Testing .removeAny() to empty {@code Set<String>}.
 */
@Test
public final void testRemoveAnyToEmpty() {
    Set<String> sActual = this.createFromArgsTest("x");
    Set<String> sExpected = this.createFromArgsRef("x");
    String x = sActual.removeAny();
    assertEquals(true, sExpected.contains(x));
    sExpected.remove(x);
    assertEquals(sExpected, sActual);
}

/**
 * Testing .removeAny() to non-empty {@code Set<String>}.
 */
@Test
public final void testRemoveAnyToNonEmpty() {
    Set<String> sActual = this.createFromArgsTest("x", "y", "z");
    Set<String> sExpected = this.createFromArgsRef("x", "y", "z");
    String x = sActual.removeAny();
    assertEquals(true, sExpected.contains(x));
    sExpected.remove(x);
    assertEquals(sExpected, sActual);
}

/*
 * Testing .contains() in this section:=
 */

/**
 * Testing .contains() on an empty {@code Set<String>}, resulting in a
 * boolean expression evaluating to false.
 */
@Test
public final void testContainsOnEmpty() {
    Set<String> sActual = this.createFromArgsTest();
    Set<String> sExpected = this.createFromArgsRef();
    assertEquals(false, sActual.contains("x"));
    assertEquals(sExpected, sActual);
}

/**
 * Testing .contains() on a non-empty {@code Set<String>} with three (3)
 * {@code String}, resulting in a boolean expression evaluating to true.
 */
@Test
public final void testContainsOnNonEmptyTrueV1() {
    Set<String> sActual = this.createFromArgsTest("j", "i", "k");
    Set<String> sExpected = this.createFromArgsRef("j", "i", "k");
    assertEquals(true, sActual.contains("i"));
    assertEquals(sExpected, sActual);
}

/**
 * Testing .contains() on a non-empty {@code Set<String>} with three (3)
 * {@code String}, resulting in a boolean expression evaluating to true.
 */
@Test
public final void testContainsOnNonEmptyTrueV2() {
    Set<String> sActual = this.createFromArgsTest("j", "i", "k");
    Set<String> sExpected = this.createFromArgsRef("j", "i", "k");
    assertEquals(true, sActual.contains("k"));
    assertEquals(sExpected, sActual);
}

/**
 * Testing .contains() on a non-empty {@code Set<String>} with three (3)
 * {@code String}, resulting in a boolean expression evaluating to true.
 */
```

```
@Test
public final void testContainsOnNonEmptyTrueV3() {
    Set<String> sActual = this.createFromArgsTest("j", "i", "k");
    Set<String> sExpected = this.createFromArgsRef("j", "i", "k");
    assertEquals(true, sActual.contains("j"));
    assertEquals(sExpected, sActual);
}

/**
 * Testing .contains() on a non-empty {@code Set<String>} with three (3)
 * {@code String}, resulting in a boolean expression evaluating to false.
 */
@Test
public final void testContainsOnNonEmptyFalse() {
    Set<String> sActual = this.createFromArgsTest("j", "i", "k");
    Set<String> sExpected = this.createFromArgsRef("j", "i", "k");
    assertEquals(false, sActual.contains("x"));
    assertEquals(sExpected, sActual);
}

/*
 * Testing .size() in this section:=
 */

/**
 * Testing .size() on an empty {@code Set<String>}.
 */
@Test
public final void testSizeOnEmpty() {
    Set<String> sActual = this.createFromArgsTest();
    Set<String> sExpected = this.createFromArgsRef();
    assertEquals(0, sActual.size());
```

```
        assertEquals(sExpected, sActual);
    }

    /**
     * Testing .size() on a non-empty {@code Set<String>} with one (1)
     * {@code String}.
     */
    @Test
    public final void testSizeOnNonEmptyV1() {
        Set<String> sActual = this.createFromArgsTest("x");
        Set<String> sExpected = this.createFromArgsRef("x");
        assertEquals(1, sActual.size());
        assertEquals(sExpected, sActual);
    }

    /**
     * Testing .size() on a non-empty {@code Set<String>} with three (3)
     * {@code String}.
     */
    @Test
    public final void testSizeOnNonEmptyV2() {
        Set<String> sActual = this.createFromArgsTest("j", "i", "k");
        Set<String> sExpected = this.createFromArgsRef("j", "i", "k");
        assertEquals(3, sActual.size());
        assertEquals(sExpected, sActual);
    }

    /*
     * Integration Testing (NOT REQUIRED):
     */

}
```