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

import java.util.Comparator;

import org.junit.Test;

import components.sortingmachine.SortingMachine;

/**
 * JUnit test fixture for {@code SortingMachine<String>}'s constructor and
 * kernel methods.
 *
 * @author Chloe Feller and Krish Patel
 *
 */
public abstract class SortingMachineTest {

    /**
     * Invokes the appropriate {@code SortingMachine} constructor for the
     * implementation under test and returns the result.
     *
     * @param order
     *            the {@code Comparator} defining the order for {@code String}
     * @return the new {@code SortingMachine}
     * @requires IS_TOTAL_PREORDER([relation computed by order.compare method])
     * @ensures constructorTest = (true, order, {})
     */
    protected abstract SortingMachine<String> constructorTest(
            Comparator<String> order);

    /**
     * Invokes the appropriate {@code SortingMachine} constructor for the
     * reference implementation and returns the result.
     *
     * @param order
     *            the {@code Comparator} defining the order for {@code String}
     * @return the new {@code SortingMachine}
     * @requires IS_TOTAL_PREORDER([relation computed by order.compare method])
     * @ensures constructorRef = (true, order, {})
     */
    protected abstract SortingMachine<String> constructorRef(
            Comparator<String> order);

    /**
     *
     * Creates and returns a {@code SortingMachine<String>} of the
     * implementation under test type with the given entries and mode.
     *
     * @param order
     *            the {@code Comparator} defining the order for {@code String}
     * @param insertionMode
     *            flag indicating the machine mode
     * @param args
     *            the entries for the {@code SortingMachine}
     * @return the constructed {@code SortingMachine}
     * @requires IS_TOTAL_PREORDER([relation computed by order.compare method])
     * @ensures <pre>
     * createFromArgsTest = (insertionMode, order, [multiset of entries in args])
     * </pre>
     */
    private SortingMachine<String> createFromArgsTest(Comparator<String> order,
            boolean insertionMode, String... args) {
        SortingMachine<String> sm = this.constructorTest(order);
        for (int i = 0; i < args.length; i++) {
            sm.add(args[i]);
        }
        if (!insertionMode) {
            sm.changeToExtractionMode();
        }
        return sm;
    }

    /**
     *
     * Creates and returns a {@code SortingMachine<String>} of the reference
     * implementation type with the given entries and mode.
     *
     * @param order
     *            the {@code Comparator} defining the order for {@code String}
     * @param insertionMode
     *            flag indicating the machine mode
     * @param args
     *            the entries for the {@code SortingMachine}
     * @return the constructed {@code SortingMachine}
     * @requires IS_TOTAL_PREORDER([relation computed by order.compare method])
     * @ensures <pre>
     * createFromArgsRef = (insertionMode, order, [multiset of entries in args])
     * </pre>
     */
    private SortingMachine<String> createFromArgsRef(Comparator<String> order,
            boolean insertionMode, String... args) {
        SortingMachine<String> sm = this.constructorRef(order);
        for (int i = 0; i < args.length; i++) {
            sm.add(args[i]);
        }
        if (!insertionMode) {
            sm.changeToExtractionMode();
        }
        return sm;
    }

    /**
     * Comparator<String> implementation to be used in all test cases. Compare
     * {@code String}s in lexicographic order.
     */
    private static class StringLT implements Comparator<String> {

        @Override
        public int compare(String s1, String s2) {
            return s1.compareToIgnoreCase(s2);
        }

    }

    /**
     * Comparator instance to be used in all test cases.
     */
    private static final StringLT ORDER = new StringLT();

    /*
     * Sample test cases.
     */

    /**
     * Test COnstructor.
     */
    @Test
    public final void testConstructor() {
        SortingMachine<String> m = this.constructorTest(ORDER);
        SortingMachine<String> mExpected = this.constructorRef(ORDER);
        assertEquals(mExpected, m);
    }

    /**
     * Test add thats empty.
     */
    @Test
    public final void testAddEmpty() {
        SortingMachine<String> m = this.createFromArgsTest(ORDER, true);
        SortingMachine<String> mExpected = this.createFromArgsRef(ORDER, true,
                "green");
        m.add("green");
        assertEquals(mExpected, m);
    }

    /**
     * Test add that has a variable.
     */
    @Test
    public final void testAddTwo() {
        SortingMachine<String> m = this.createFromArgsTest(ORDER, true, "a");
        SortingMachine<String> mExpected = this.createFromArgsRef(ORDER, true,
                "a", "b");
        m.add("b");
        assertEquals(mExpected, m);
    }

    /**
     * Test add that has multiple variable.
     */
    @Test
    public final void testAddThree() {
        SortingMachine<String> m = this.createFromArgsTest(ORDER, true, "a",
                "b", "c", "d", "e");
        SortingMachine<String> mExpected = this.createFromArgsRef(ORDER, true,
                "a", "b", "c", "d", "e", "f");
        m.add("f");
        assertEquals(mExpected, m);
    }

    // TODO - add test cases for add, changeToExtractionMode, removeFirst,
    // isInInsertionMode, order, and size

    /**
     * Test cases for changeToExtractionMode.
     */
    /**
```

```java
     * Test for changeToExtractionMode with an empty SortingMachine.
     */
    @Test
    public final void changeToExtractionModeOne() {
        // Initialize the variables
        SortingMachine<String> sort = this.createFromArgsTest(ORDER, true);
        SortingMachine<String> sortExpected = this.createFromArgsRef(ORDER,
                false);

        // Call the method
        sort.changeToExtractionMode();

        // Assert variables are equal
        assertEquals(sortExpected, sort);
    }

    /**
     * Test for changeToExtractionMode with a non-empty SortingMachine.
     */
    @Test
    public final void changeToExtractionModeTwo() {
        // Initialize the variables
        SortingMachine<String> sort = this.createFromArgsTest(ORDER, true,
                "Boolean", "Double", "String");
        SortingMachine<String> sortExpected = this.createFromArgsRef(ORDER,
                false, "Boolean", "Double", "String");

        // Call the method
        sort.changeToExtractionMode();

        // Assert variables are equal
        assertEquals(sortExpected, sort);
    }

    /**
     * Test cases for removeFirst.
     */
    /**
     * Test for removeFirst resulting in an empty SortingMachine.
     */
    @Test
    public final void removeFirstOne() {
        // Initialize the variables
        SortingMachine<String> sort = this.createFromArgsTest(ORDER, false,
                "Double");
        SortingMachine<String> sortExpected = this.createFromArgsRef(ORDER,
                false);

        // Call the method
        String removed = sort.removeFirst();

        // Assert variables are equal
        assertEquals(sortExpected, sort);
        assertEquals("Double", removed);
    }

    /**
     * Test for removeFirst resulting in a non-empty SortingMachine.
     */
    @Test
    public final void removeFirstTwo() {
        // Initialize the variables
        SortingMachine<String> sort = this.createFromArgsTest(ORDER, false,
                "Boolean", "Array", "Double", "Ai");
        SortingMachine<String> sortExpected = this.createFromArgsRef(ORDER,
                false, "Boolean", "Array", "Double");

        // Call the method
        String removed = sort.removeFirst();

        // Assert variables are equal
        assertEquals(sortExpected, sort);
        assertEquals("Ai", removed);
    }

    /**
     * Test cases for isInInsertionMode.
     */
    /**
     * Test for isInInsertionMode with an empty SortingMachine.
     */
    @Test
    public final void isInInsertionModeOne() {
        // Initialize the variables
        SortingMachine<String> sort = this.createFromArgsTest(ORDER, true);
        SortingMachine<String> sortExpected = this.createFromArgsRef(ORDER,
                true);

        // Call the method
        boolean insert = sort.isInInsertionMode();

        // Assert variables are equal
        assertEquals(sortExpected, sort);
        assertEquals(true, insert);
    }

    /**
     * Test for isInInsertionMode with a non-empty SortingMachine.
     */
    @Test
    public final void isInInsertionModeTwo() {
        // Initialize the variables
        SortingMachine<String> sort = this.createFromArgsTest(ORDER, true,
                "Double", "Boolean");
        SortingMachine<String> sortExpected = this.createFromArgsRef(ORDER,
                true, "Double", "Boolean");

        // Call the method
        boolean insert = sort.isInInsertionMode();

        // Assert variables are equal
        assertEquals(sortExpected, sort);
        assertEquals(true, insert);
    }

    /**
     * Test order through insertion.
     */
    @Test
    public final void orderTestOne() {
        SortingMachine<String> m = this.createFromArgsTest(ORDER, true);
        SortingMachine<String> mExpected = this.createFromArgsRef(ORDER, true);

        Comparator<String> o = m.order();
        Comparator<String> oExpected = mExpected.order();

        assertEquals(oExpected, o);
        assertEquals(mExpected, m);
    }

    /**
     * Test order through extraction.
     */
    @Test
    public final void orderTestTwo() {
        SortingMachine<String> m = this.createFromArgsTest(ORDER, false);
        SortingMachine<String> mExpected = this.createFromArgsRef(ORDER, false);

        Comparator<String> o = m.order();
        Comparator<String> oExpected = mExpected.order();

        assertEquals(oExpected, o);
        assertEquals(mExpected, m);
    }

    /**
     * Test for size with empty variables through insertion.
     */
    @Test
    public final void sizeTestInsertionOne() {
        SortingMachine<String> m = this.createFromArgsTest(ORDER, true);

        int i = m.size();

        assertEquals(0, i);
    }

    /**
     * Test for size with empty variables through extraction.
     */
    @Test
    public final void sizeTestExtractionOne() {
        SortingMachine<String> m = this.createFromArgsTest(ORDER, false);

        int i = m.size();

        assertEquals(0, i);
    }

    /**
     * Test for size with a variable through insertion.
     */
    @Test
    public final void sizeTestInsertionTwo() {
        SortingMachine<String> m = this.createFromArgsTest(ORDER, true, "a");

        int i = m.size();
```

```java

        assertEquals(1, i);
    }

    /**
     * Test for size with a variable through extraction.
     */
    @Test
    public final void sizeTestExtractionTwo() {
        SortingMachine<String> m = this.createFromArgsTest(ORDER, false, "a");

        int i = m.size();

        assertEquals(1, i);
    }

    /**
     * Test for size with multiple variables through insertion.
     */
    @Test
    public final void sizeTestInsertionThree() {
        SortingMachine<String> m = this.createFromArgsTest(ORDER, true, "a",
                "b", "c", "d", "e", "f");

        int i = m.size();

        final int expectedI = 6;

        assertEquals(expectedI, i);
    }

}

```