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

/**
 * JUnit test fixture for {@code SortingMachine<String>}'s constructor and
 * kernel methods.
 *
 * @author Shyam Sai Bethina and Yihone Chu
 *
 */
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);

```

```java
    /**
     *
     * 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.
     */

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

    /**
     * Edge case for add method.
     */
    @Test
    public final void testAdd1() {
        SortingMachine<String> m = this.createFromArgsTest(ORDER, 
true);
        SortingMachine<String> mExpected = 
this.createFromArgsRef(ORDER, true,
                "green");
        m.add("green");
        assertEquals(mExpected, m);
    }

    /**
     * Challenging case for add method.
     */
    @Test
    public final void testAdd2() {
        SortingMachine<String> m = this.createFromArgsTest(ORDER, 
true,
                "green");
        SortingMachine<String> mExpected = 
this.createFromArgsRef(ORDER, true,
                "green", "green");
        m.add("green");
        assertEquals(mExpected, m);
    }

    /**
```

```
     * Routine case for add method.
     */
    @Test
    public final void testAdd3() {
        SortingMachine<String> m = this.createFromArgsTest(ORDER,
true,
                "green");
        SortingMachine<String> mExpected =
this.createFromArgsRef(ORDER, true,
                "blue", "green");

        m.add("blue");

        assertEquals(mExpected, m);
    }

    /**
     * Edge case for changeToExtractionMode method.
     */
    @Test
    public final void testChangeMode1() {
        SortingMachine<String> m = this.createFromArgsTest(ORDER,
true);
        SortingMachine<String> mExpected =
this.createFromArgsRef(ORDER, true);

        m.changeToExtractionMode();
        mExpected.changeToExtractionMode();

        assertEquals(mExpected, m);
    }

    /**
     * Challenging case for changeToExtractionMode method.
     */
    @Test
    public final void testChangeMode2() {
        SortingMachine<String> m = this.createFromArgsTest(ORDER,
true, "");
        SortingMachine<String> mExpected =
this.createFromArgsRef(ORDER, true,
                "");

        m.changeToExtractionMode();
        mExpected.changeToExtractionMode();

        assertEquals(mExpected, m);
    }

    /**
     * Routine case for changeToExtractionMode method.
     */
    @Test
    public final void testChangeMode3() {
        SortingMachine<String> m = this.createFromArgsTest(ORDER,
true, "green",
                "blue");
        SortingMachine<String> mExpected =
this.createFromArgsRef(ORDER, true,
                "green", "blue");

        m.changeToExtractionMode();
        mExpected.changeToExtractionMode();

        assertEquals(mExpected, m);
    }

    /**
     * Edge case for removeFirst method.
     */
    @Test
    public final void testemoveFirst1() {
        SortingMachine<String> m = this.createFromArgsTest(ORDER,
false,
                "green");
        SortingMachine<String> mExpected =
this.createFromArgsRef(ORDER, false,
                "green");

        String removed = m.removeFirst();
        String expected = mExpected.removeFirst();

        assertEquals(mExpected, m);
        assertEquals(expected, removed);
    }

    /**
     * Challenging case for removeFirst method.
```

```java
     */
    @Test
    public final void testRemoveFirst2() {
        SortingMachine<String> m = this.createFromArgsTest(ORDER, false, "");
        SortingMachine<String> mExpected = this.createFromArgsRef(ORDER, false,
                "");

        String removed = m.removeFirst();
        String expected = mExpected.removeFirst();

        assertEquals(mExpected, m);
        assertEquals(expected, removed);
    }

    /**
     * Routine case for removeFirst method.
     */
    @Test
    public void testRemoveFirst3() {
        SortingMachine<String> m = this.createFromArgsTest(ORDER, false,
                "hello", "there", "professor");
        SortingMachine<String> mExpected = this.createFromArgsRef(ORDER, false,
                "hello", "there", "professor");

        String removed = m.removeFirst();
        String expected = mExpected.removeFirst();

        assertEquals(mExpected, m);
        assertEquals(expected, removed);
    }

    /**
     * Edge case for isInInsertionMode method.
     */
    @Test
    public void testInsertionMode1() {
        SortingMachine<String> m = this.createFromArgsTest(ORDER, false);
        SortingMachine<String> mExpected = this.createFromArgsRef(ORDER, false);

        Boolean test = m.isInInsertionMode();
        Boolean expected = mExpected.isInInsertionMode();

        assertEquals(mExpected, m);
        assertEquals(expected, test);

    }

    /**
     * Challenging case for isInInsertionMode method.
     */
    @Test
    public void testInsertionMode2() {
        SortingMachine<String> m = this.createFromArgsTest(ORDER,
true, "");
        SortingMachine<String> mExpected =
this.createFromArgsRef(ORDER, true,
                "");

        Boolean test = m.isInInsertionMode();
        Boolean expected = mExpected.isInInsertionMode();

        assertEquals(mExpected, m);
        assertEquals(expected, test);
    }

    /**
     * Routine case for isInInsertionMode method.
     */
    @Test
    public void testInsertionMode3() {
        SortingMachine<String> m = this.createFromArgsTest(ORDER,
false, "blue",
                "green");
        SortingMachine<String> mExpected =
this.createFromArgsRef(ORDER, false,
                "blue", "green");

        Boolean test = m.isInInsertionMode();
        Boolean expected = mExpected.isInInsertionMode();

        assertEquals(mExpected, m);
        assertEquals(expected, test);
```

```java
    }

    /**
     * Edge case for order method.
     */
    @Test
    public void testOrder1() {
        SortingMachine<String> m = this.createFromArgsTest(ORDER,
false);
        SortingMachine<String> mExpected =
this.createFromArgsRef(ORDER, false);

        Comparator<String> test = m.order();
        Comparator<String> expected = mExpected.order();

        assertEquals(mExpected, m);
        assertEquals(expected, test);
    }

    /**
     * Challenging case for order method.
     */
    @Test
    public void testOrder2() {
        SortingMachine<String> m = this.createFromArgsTest(ORDER,
true, "");
        SortingMachine<String> mExpected =
this.createFromArgsRef(ORDER, true,
                "");

        Comparator<String> test = m.order();
        Comparator<String> expected = mExpected.order();

        assertEquals(mExpected, m);
        assertEquals(expected, test);
    }

    /**
     * Routine case for order method.
     */
    @Test
    public void testOrder3() {
        SortingMachine<String> m = this.createFromArgsTest(ORDER,
true, "blue",
                "green");
        SortingMachine<String> mExpected =
this.createFromArgsRef(ORDER, true,
                "blue", "green");

        Comparator<String> test = m.order();
        Comparator<String> expected = mExpected.order();

        assertEquals(mExpected, m);
        assertEquals(expected, test);
    }

    /**
     * Edge case for size method.
     */
    @Test
    public void testSize1() {
        SortingMachine<String> m = this.createFromArgsTest(ORDER,
true);
        SortingMachine<String> mExpected =
this.createFromArgsRef(ORDER, true);

        int test = m.size();
        int expected = mExpected.size();

        assertEquals(mExpected, m);
        assertEquals(expected, test);
    }

    /**
     * Challenging case for size method.
     */
    @Test
    public void testSize2() {
        SortingMachine<String> m = this.createFromArgsTest(ORDER,
false, "");
        SortingMachine<String> mExpected =
this.createFromArgsRef(ORDER, false,
                "");

        int test = m.size();
        int expected = mExpected.size();

        assertEquals(mExpected, m);
        assertEquals(expected, test);
    }

    /**
     * Routine case for size method.
     */
    @Test
    public void testSize3() {
        SortingMachine<String> m = this.createFromArgsTest(ORDER, false, "blue",
                "green");
        SortingMachine<String> mExpected = this.createFromArgsRef(ORDER, false,
                "blue", "green");

        int test = m.size();
        int expected = mExpected.size();

        assertEquals(mExpected, m);
        assertEquals(expected, test);
    }

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

}
```