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

/**
 * JUnit test fixture for {@code SortingMachine<String>}'s constructor and
 * kernel methods.
 *
 * @author Zhuoyang Li + Xinci Ma
 *
 */
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
```

```java
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)
```

```java
{
            return s1.compareToIgnoreCase(s2);
        }

    }

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

    /*
     * Sample test cases.
     */

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

    @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);
    }

    // TODO - add test cases for add, changeToExtractionMode, removeFirst,
    // isInInsertionMode, order, and size
    @Test
    public final void testAdd1() {
        SortingMachine<String> m = this.createFromArgsTest(ORDER, true,
                "green");
        SortingMachine<String> mExpected = this.createFromArgsRef(ORDER, true,
                "green", "red");
        m.add("red");
        assertEquals(mExpected, m);
    }

    // space + String
    @Test
    public final void testAdd2() {
        SortingMachine<String> m = this.createFromArgsTest(ORDER, true, "1",
                "7", "5", "5", "5", "8", "7", "6", "1", "0");
        SortingMachine<String> mExpected = this.createFromArgsRef(ORDER, true,
                "1", "7", "5", "5", "5", "8", "7", "6", "1", "0", " 9");
        m.add(" 9");
        assertEquals(mExpected, m);

    }

    // space only
    @Test
    public final void testAdd3() {
        SortingMachine<String> m = this.createFromArgsTest(ORDER, true, "1",
                "7", "5", "5", "5", "8", "7", "6", "1", "0");
        SortingMachine<String> mExpected = this.createFromArgsRef(ORDER, true,
                "1", "7", "5", "5", "5", "8", "7", "6", "1", "0", " ");
        m.add(" ");
        assertEquals(mExpected, m);

    }

    @Test
    public final void testChangeToExtractionMode1() {
        SortingMachine<String> m = this.createFromArgsTest(ORDER, true, "1",
                "7", "5", "5", "5", "8", "7", "6", "1", "0");
        SortingMachine<String> mExpected = this.createFromArgsRef(ORDER, false,
                "1", "7", "5", "5", "5", "8", "7", "6", "1", "0");
        m.changeToExtractionMode();
        assertEquals(mExpected, m);
    }

    // test empty
    @Test
    public final void testChangeToExtractionMode2() {
        SortingMachine<String> m = this.createFromArgsTest(ORDER, true);
        SortingMachine<String> mExpected = this.createFromArgsRef(ORDER, false);
        m.changeToExtractionMode();
        assertEquals(mExpected, m);
    }

    @Test
    public final void testRemoveFirst1() {
        SortingMachine<String> m = this.createFromArgsTest(ORDER, false, "1",
                "7", "5", "5", "5", "8", "7", "6", "1", "0");
        SortingMachine<String> mExpected = this.createFromArgsRef(ORDER, false,
                "1", "1", "5", "5", "5", "6", "7", "7", "8");
        String s = m.removeFirst();

        assertEquals(mExpected, m);
        assertEquals("0", s);
    }

    // test one element
    @Test
    public final void testRemoveFirst2() {
        SortingMachine<String> m = this.createFromArgsTest(ORDER, false,
                "Real");
        SortingMachine<String> mExpected = this.createFromArgsRef(ORDER, false);
        String s = m.removeFirst();
        assertEquals(mExpected, m);
        assertEquals("Real", s);
    }

    //test same elements twice
    @Test
    public final void testRemoveFirst3() {
        SortingMachine<String> m = this.createFromArgsTest(ORDER, false, "1",
                "1", "1", "1", "1");
        SortingMachine<String> mExpected = this.createFromArgsRef(ORDER, false,
                "1", "1", "1", "1", "1");
        String s = m.removeFirst();
        String s1 = mExpected.removeFirst();

        assertEquals(mExpected, m);
        assertEquals(s1, s);
    }

    @Test
    public final void testRemoveFirst4() {
        SortingMachine<String> m = this.createFromArgsTest(ORDER, false, "1",
                "7", "5", "5", "5", "8", "7", "6", "1", "0");
        SortingMachine<String> mExpected = this.createFromArgsRef(ORDER, false,
                "1", "7", "5", "5", "5", "8", "7", "6", "1", "0");
        String s = m.removeFirst();
        String s1 = mExpected.removeFirst();

        assertEquals(mExpected, m);
        assertEquals(s1, s);
    }

    @Test
    public final void testIsInInsertionMode1() {
        SortingMachine<String> m = this.createFromArgsTest(ORDER, true, "1",
                "7", "5", "5", "5", "8", "7", "6", "1", "0");
        SortingMachine<String> mExpected = this.createFromArgsRef(ORDER, true,
                "1", "7", "5", "5", "5", "8", "7", "6", "1", "0");
        boolean b = m.isInInsertionMode();

        assertEquals(mExpected, m);
        assertEquals(true, b);
    }

    //test empty false
    @Test
    public final void testIsInInsertionMode2() {
        SortingMachine<String> m = this.createFromArgsTest(ORDER, false);
        SortingMachine<String> mExpected = this.createFromArgsRef(ORDER, false);
        boolean b = m.isInInsertionMode();
        boolean b1 = mExpected.isInInsertionMode();

        assertEquals(mExpected, m);
        assertEquals(b1, b);
    }

    //test empty true
    @Test
    public final void testIsInInsertionMode3() {
        SortingMachine<String> m = this.createFromArgsTest(ORDER, true);
        SortingMachine<String> mExpected =
this.createFromArgsRef(ORDER, true);
        boolean b = m.isInInsertionMode();
        boolean b1 = mExpected.isInInsertionMode();

        assertEquals(mExpected, m);
        assertEquals(b1, b);
    }

    //test nonempty false
    @Test
    public final void testIsInInsertionMode4() {
        SortingMachine<String> m = this.createFromArgsTest(ORDER, false, "1",
                "7", "5", "5", "5", "8", "7", "6", "1", "0");
        SortingMachine<String> mExpected = this.createFromArgsRef(ORDER, false,
                "1", "7", "5", "5", "5", "8", "7", "6", "1", "0");
        boolean b = m.isInInsertionMode();
        boolean b1 = mExpected.isInInsertionMode();

        assertEquals(mExpected, m);
        assertEquals(b1, b);
    }

    @Test
    public final void testOrder1() {
        SortingMachine<String> m = this.createFromArgsTest(ORDER, true, "1",
                "7", "5", "5", "5", "8", "7", "6", "1", "0");
        SortingMachine<String> mExpected =
this.createFromArgsRef(ORDER, true,
                "1", "7", "5", "5", "5", "8",
"7", "6", "1", "0");
        Comparator<String> c = m.order();

        assertEquals(mExpected, m);
        assertEquals(ORDER, c);
    }

    //test empty
    @Test
    public final void testOrder2() {
        SortingMachine<String> m =
this.createFromArgsTest(ORDER, true);
        SortingMachine<String> mExpected =
this.createFromArgsRef(ORDER, true);
        Comparator<String> c = m.order();
        Comparator<String> c1 =
mExpected.order();

        assertEquals(mExpected, m);
        assertEquals(c1, c);
    }

    //test nonempty with same elements
    @Test
    public final void testOrder3() {
        SortingMachine<String> m =
this.createFromArgsTest(ORDER, true, "1",
                "1", "1", "1", "1");
        SortingMachine<String> mExpected =
this.createFromArgsRef(ORDER, true,
                "1", "1", "1", "1", "1");
        Comparator<String> c = m.order();

```

```java
        assertEquals(mExpected, m);
        assertEquals(ORDER, c);
    }

    //test empty with extraction mode
    @Test
    public final void testOrder4() {
        SortingMachine<String> m = this.createFromArgsTest(ORDER, false);
        SortingMachine<String> mExpected = this.createFromArgsRef(ORDER, false);
        Comparator<String> c = m.order();

        assertEquals(mExpected, m);
        assertEquals(ORDER, c);
    }

    //test nonempty with extraction mode
    @Test
    public final void testOrder5() {
        SortingMachine<String> m = this.createFromArgsTest(ORDER, false, "1",
                "7", "5", "5", "5", "8", "7", "6", "1", "0");
        SortingMachine<String> mExpected = this.createFromArgsRef(ORDER, false,
                "1", "7", "5", "5", "5", "8", "7", "6", "1", "0");
        Comparator<String> c = m.order();

        assertEquals(mExpected, m);
        assertEquals(ORDER, c);
    }

    @Test
    public final void testSize1() {
        SortingMachine<String> m = this.createFromArgsTest(ORDER, true, "1",
                "7", "5", "5", "5", "8", "7", "6", "1", "0");
        SortingMachine<String> mExpected = this.createFromArgsRef(ORDER, true,
                "1", "7", "5", "5", "5", "8", "7", "6", "1", "0");
        int s = m.size();
        int s1 = mExpected.size();

        assertEquals(mExpected, m);
        assertEquals(s1, s);
        assertEquals(10, s);
    }

    //test empty with insertion mode
    @Test
    public final void testSize2() {
        SortingMachine<String> m = this.createFromArgsTest(ORDER, true);
        SortingMachine<String> mExpected = this.createFromArgsRef(ORDER, true);
        int s = m.size();
        int s1 = mExpected.size();

        assertEquals(mExpected, m);
        assertEquals(s1, s);
        assertEquals(0, s);
    }

    //test empty with extraction mode
    @Test
    public final void testSize3() {
        SortingMachine<String> m = this.createFromArgsTest(ORDER, false);
        SortingMachine<String> mExpected = this.createFromArgsRef(ORDER, false);
        int s = m.size();
        int s1 = 0;

        assertEquals(mExpected, m);
        assertEquals(s1, s);
    }

    //test nonempty with extraction mode
    @Test
    public final void testSize4() {
        SortingMachine<String> m = this.createFromArgsTest(ORDER, false, "1",
                "7", "5", "5", "5", "8", "7", "6", "1", "0");
        SortingMachine<String> mExpected = this.createFromArgsRef(ORDER, false,
                "1", "7", "5", "5", "5", "8", "7", "6", "1", "0");
        int s = m.size();
        int s1 = 10;

        assertEquals(mExpected, m);
        assertEquals(s1, s);
    }

}

```