Project #2
Specifications, Implementations, and Test Cases (Oh My!)

1 Objectives

There are three objectives to this project. First, you will gain experience developing a *formal interface specification* based on an abstract model. Second, you will gain experience implementing a class based solely on your understanding of its interface specification. Finally, you will gain additional experience implementing specification-based unit tests using the *JUnit* testing framework.

This is a short project, but you are encouraged to start early in case you run into unexpected problems with the mathematics.

2 Requirements

This project is divided into three components.

- All of your classes including your JUnit test class, must be defined in the cu.cs.cpsc215.project2 package.
- **Specification.** For the first component of the project, you must develop the full formal specification of the SList interface included at the end of this document. This means that you should provide the appropriate pre-conditions, post-conditions, and preservation clauses. You should use the abstract model provided and base each method contract on the method's informal description. Be sure to use the notation presented in class. (**Note: You do not have to specify the behavior of** abstractToString().)

  Note that SList is modeled as a *pair of string of object*. The model declaration gives each element a name – left and right, respectively. In a pre-condition or post-condition, you may use self.left to refer to the *"left"* string, and self.right to refer to the *"right"* string. If you use self alone, it refers to the pair of strings.
- **Implementation.** For the second component of the project, you must provide an implementation of the SList interface. Your class should be named SListImpl1.
- **Testing.** For the third component of the project, you must develop specification-based unit tests to validate the correctness of your SList implementation. More precisely, you are required to implement a JUnit test class, TestSListImpl1, that tests the correctness of SListImpl1.

  As you did in the closed lab, you must provide one testXXX() method for each method defined by SList. For each testXXX() method, you must provide *at least* three test cases per method contract, taking care to test any *"boundary cases"*. Try your best to test the class thoroughly — include those cases that are most likely to exercise all paths through an implementation of SList.

  Be sure to eliminate redundancy in your test suite by providing an appropriate implementation of @Before (you should use JUnit v4).

3 Submission Instructions

This project is due by class time (2:30 pm) on October 29^{th}. When your project is complete, archive your source materials and use `handin.cs.clemson.edu` to submit your work. You must name your archive project2.zip, and it must compile and run its tests with the following script:

```
unzip project2.zip
javac -cp .:junit.jar cu/cs/cpsc215/project2/*.java
java -cp .:junit.jar org.junit.runner.JUnitCore cu.cs.cpsc215.project2.TestSListImpl1
```

You do *not* need to include junit.jar in your zip. Projects that fail to build and execute correctly with this script will receive a 0.

Note: To create an archive with Eclipse that meets this specification, right-click your project in Eclipse, and choose Export >General >Archive File. In the Export Archive file dialog, click Deselect All, then select the cu folder within your projects src folder, and choose the radio button labeled Create only selected directories. Enter the name project2.zip for the archive file and click Finish. Be sure to check your archive file after exporting it from Eclipse to verify that it contains your source code in the correct directory structure.

4 Grading

Your project will be graded based on your adherence to the specified requirements, and the specification and programming guidelines discussed in class.

This is an intermediate course in software development. Your specification must be typed. Your source materials should be properly documented. Your source must compile. Your application must not crash. A violation of any of these requirements will result in an automatic zero. **Test your application thoroughly.**

5 Collaboration

You must work independently on this project. You must not discuss the problem or the solution with classmates. **Any form of collaboration will be considered academic misconduct.**

```
1  public interface SList {
2     // modeled by: (left: string of object, right: string of object)
3     // initial value: (<>,<>)
4
5     void clear();
6     // * This method may be called at any time.
7     // * This method clears all entries from the left and
8     //   right strings.
9
10    void addRight(Object x);
```

```
11    // * This method may be called at any time.
12    // * This method prepends x to the right string.
13    // * X is not modified by the method.
14
15    Object removeRight();
16    // * This method may be called when the right string
17    //   is non-empty.
18    // * This method removes and returns the leftmost
19    //   entry from the right string.
20
21
22    Object getElementAt(int pos);
23    // * This method may be called when pos is greater
24    //   than or equal to zero and less than the number of
25    //   entries in the right string.
26    // * This method returns (but does not remove) the
27    //   entry at position pos within the right string.
28    // * pos is not modified by the method.
29    // * self is not modified by the method.
30
31    void advance();
32    // * This method may be called when the right string
33    //   is non-empty.
34    // * This method removes the leftmost entry from the
35    //   right string and appends it to the left string.
36
37    void moveToStart();
38    // * This method may be called at any time.
39    // * This method first prepends the left string to the
40    //   right string, and then sets the left string to
41    //   empty-string.
42
43    void moveToFinish();
44    // * This method may be called at any time.
45    // * This method first appends the right string to the
46    //   left string, and then sets the right string to
47    //   empty-string.
48
49    int getLeftLength();
50    // * This method may be called at any time.
51    // * This method returns the length of the left
52    //   string.
53    // * self is not modified by the method.
54
55    int getRightLength();
56    // * This method may be called at any time.
57    // * This method returns the length of the right
58    //   string.
59    // * self is not modified by the method.
60
61    String abstractToString();
62    // * This method may be called at any time.
63    // * The String returned by this method should encode
64    //   the abstract value of self.
```

```
65      // * The String returned by this method should follow
66      //   the following pattern exactly:
67      //     - (<>,<>)
68      //     - (<1>,<>)
69      //     - (<1,2>,<>)
70      //     - (<1,2>,<1>)
71      //     - (<1,2>,<1,2>)
72      //     - ...
73  }
```

Listing 1: Informal Specification of SList