CSE 274: Data Abstraction and Data Structures
Program #2 – Linked List Algorithms - 120 Points
Due September 20, by 11:59 pm

Outcomes:

- Write a Java program that implements linked list algorithms

General requirements:

- Javadoc is NOT required for this project.
- Follow good programming practices - your source code will be evaluated for this project.
  - Format your code so that it is readable using generally accepted guidelines for formatting source code.
  - Keep your methods compact and cohesive. If a method starts to become long, or the nesting becomes too deep, consider creating a helper method.
  - Reuse existing methods, when feasible.
  - Your primary objective is clarity; efficiency is a secondary concern. That is, use the clearest solution if it does not change the big O.

Specific requirements:

Download the class named `LinkedAlgorithms.java`. You are to complete the following methods (read the documentation). In the implementation, you must use the linked list techniques discussed in class; all your code must be contained in `LinkedAlgorithms.java`. Do not use any Java collection classes (e.g., *ArrayList*, *LinkedList*, etc.). The linked list treats indices like an array: 0 represents the first element in the list, 1 the second element, and so on.

Notes:

- Do not put package statements in your code.
- All of your code must reside within `LinkedAlgorithms.java`. This is all that you will submit.
- Do not delete any methods, rename any methods, change public to private, or change parameter lists. If you are not sure how to implement a method, leave the method as a stub (header plus return statement).
- You may add private methods as you see fit.
- You may not add any additional data members to the class definition.
- The main method that is provided to you must compile with your code.
- The bold methods shown below must contain exactly the source code provided in the instructional videos.

Scoring:

(12 pts) Programming style.

(10 pts) Appropriate and thorough Junit testing

(10 pts) Testing Coverage

(88 pts) Correctness of methods. Some these were completed in Lab #3. You may reused and adapted your code from that lab assignment.

1. (9 pts; 3 pts each) Constructors
   a. `public LinkedAlgorithms() {`
   b. `public LinkedAlgorithms(String [] data) {`
   c. `public LinkedAlgorithms(LinkedAlgorithms original) {`

2. (12 pts; 3 pts each) Utilities
   a. `public String toArray() {`
   b. `public String toString() {`
   c. `public int size() {`
   d. `public boolean equalsLinkedList(LinkedAlgorithms other) {`

3. (6 pts; 3 pts each) Search
   a. `public boolean contains(String data) {`
   b. `public int find(String data) {`

4. (9 pts; 3 pts each) Retrieval
   a. `public String getFirst() {`
   b. `public String getLast() {`
   c. `public String getAt(int i) {`

5. (15 pts) Insertion
   a. (9 pts; 3 pts each) By position
      i. `public void insertFirst(String data) {`
      ii. `public void insertLast(String data) {`
      iii. `public void insertAt(int i, String data) {`
   b. (6 pts; 3 pts each) By data value
      i. `public void insertBefore(String newData, String existingData) {`
      ii. `public void insertAfter(String newData, String existingData) {`

6. (15 pts) Removal
   a. (9 pts; 3 pts each) By position
      i. `public String removeFirst() {`
      ii. `public String removeLast() {`
      iii. `public String removeAt(int i) {`
   b. (6 pts; 3 pts each) By data value
      i. `public boolean removeFirstOccurrenceOf(String data) {`
      ii. `public int removeAllOccurrencesOf(String data) {`

7. (6 pts; 3 pts each) List Manipulation
   a. `public void reverse() {`
   b. `public void toUpper() {`

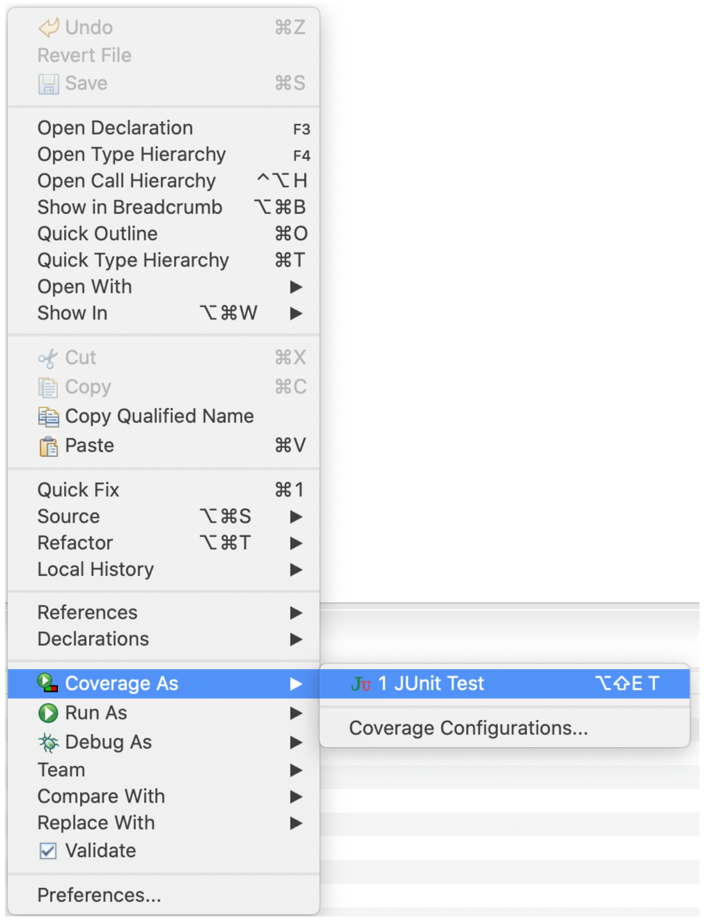
8. (16 pts; 4 pts each) Properties
   a. `public String getConcatenation() {`
   b. `public String getAlphabeticallyLast() {`
   c. `public int indexOfAlphabeticallyLast() {`
   d. `public boolean anagrams(Linked Algorithms other) {`

What to turn in:

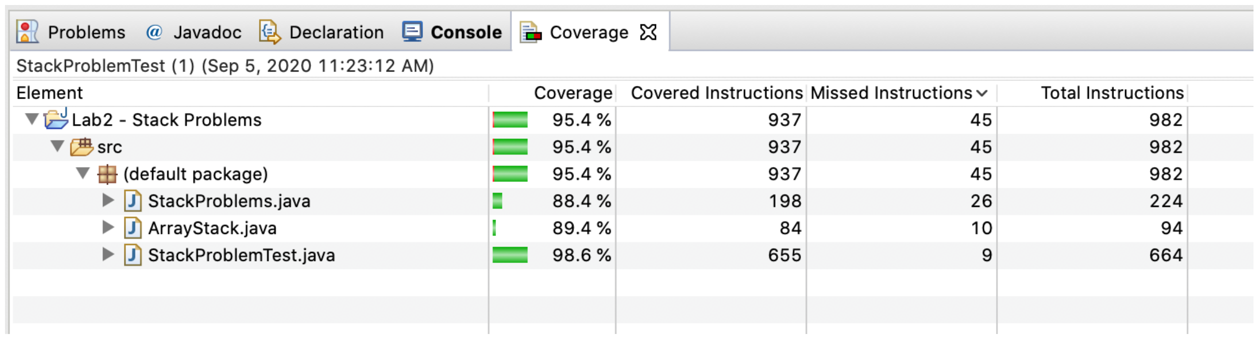
Please turn in one zip file with these three things:

1. The java file with your code for these problems. It must be named *LinkedAlgorithms.java*
2. The java file for your Junit test. It must be named *LinkedAlgorithmsTests.java*
3. A picture of your Eclipse workspace that shows the window that shows your coverage percentages. This could be a pdf or png file. See the example picture on the next page.

To get the coverage report, right click in Eclipse on the Junit java file. You will get a menu like the one shown here:



When you make this choice, your code that has been tested will be colored green. Eclipse will also create a window that shows the percentage of test coverage. Make sure you open this so that we can see the coverage of each java file. This is the example for Lab2. Your Coverage report will look similar for Program #2.



StackProblemTest (1) (Sep 5, 2020 11:23:12 AM)

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
Lab2 - Stack Problems	95.4 %	937	45	982
src	95.4 %	937	45	982
(default package)	95.4 %	937	45	982
StackProblems.java	88.4 %	198	26	224
ArrayStack.java	89.4 %	84	10	94
StackProblemTest.java	98.6 %	655	9	664