Programming Assignment 4
Introduction to Linked Lists

1 Objectives

1. Implement singly/doubly linked lists.
2. Get introduced to ADTs (Abstract Data Types)
3. Implement an application for linked lists.

2 Linked List Implementation

Organize your code under package with name

`eg.edu.alexu.csd.datastructure.linkedList.cs<xx>_cs<yy>`
`where xx and yy are your and your partner two digits class number.`

It is required to implement the following interface **twice**. Once using a **Singly** Linked List implementation and once using a **Doubly** Linked List implementation in two different classes.

```
package eg.edu.alexu.csd.datastructure.linkedList;
public interface ILinkedList {
    /**
     * Inserts a specified element at the specified position in the list.
     * @param index
     * @param element
     */
    public void add(int index, Object element);

    /**
     * Inserts the specified element at the end of the list.
     * @param element
     */
    public void add(Object element);

    /**
     * @param index
     * @return the element at the specified position in this list.
     */
    public Object get(int index);

    /**
```

```
     * Replaces the element at the specified position in this list with the
     * specified element.
     * @param index
     * @param element
     */
    public void set(int index, Object element);

    /**
     * Removes all of the elements from this list.
     */
    public void clear();

    /**
     * @return true if this list contains no elements.
     */
    public boolean isEmpty();

    /**
     * Removes the element at the specified position in this list.
     * @param index
     */
    public void remove(int index);

    /**
     * @return the number of elements in this list.
     */
    public int size();

    /**
     * @param fromIndex
     * @param toIndex
     * @return a view of the portion of this list between the specified
     *         fromIndex and toIndex, inclusively.
     */
    public ILinkedList sublist(int fromIndex, int toIndex);

    /**
     * @param o
     * @return true if this list contains an element with the same value as the
     *         specified element.
     */
    public boolean contains(Object o);
}
```

3 Testing

Provide a JUnit test suite to test both implementations. A traditional testing scenarios might look like:

- Initialize the linked list to point to your implementation and add some nodes to the list.
- Test the correct insertion of nodes by calling the get() method for all possible indices. Do not forget to test the special case of invalid index parameter. Index starts from 0.
- Add one more node at an index in the middle of the list. Use the get() method to assure that the node is added at the correct index.
- Change one node to point to another element with a value different than the old one by calling the set() method. Use the get() method to test if the node is updated properly.
- Use sublist() to choose some elements of the list. Assure that the size of the sublist is correct and the elements are the desired ones.
- Remove one node from the list and assert that the size of the list has been decreased and that the node has been deleted properly.
- Test the contains() method by calling the method with two integers: one that is in the list and another that is not in the list.
- Clear the elements of the list. Assure that the list is Empty.

4 Application

You are required to design a linked allocation system to represent and manipulate polynomials. You should use one of the linked list classes you implemented in part (A).
Each term of the polynomial will be represented as a node, using it's coefficient and exponent. Assume that you have 3 available polynomial variables: A, B and C, that can be set by the user and one variable R that acts as an accumulator for the results of operations on other polynomials. You should order the polynomial terms in descending order by the exponent. A polynomial can have a negative exponent.
Create a user-friendly, menu-driven system that performs the following operations:

- Read in a polynomial and store it in variable A, B, or C.
- Output a polynomial using a form that clearly displays it.
- Add two polynomials and store the result in R.
- Subtract two polynomials and store the result in R.

- Multiply two polynomials and store the result in R.
- Evaluate a polynomial at some point, a, where a is a floating point constant. In other words, substitute by the given value in your polynomial. Display the result as a floating point.
- Clear a polynomial. Note that: a polynomial whose value is cleared or initially unset cannot be involved in an operation.

It is required to implement the following interface by the core of your application. Note that: The core of the application should **throw a runtime exception** when it encounters an invalid input or operation. The result 'R' should be 'null' at first before doing any operation. An empty polynomial should be '{0, 0}'.

```
package eg.edu.alexu.csd.datastructure.linkedList;
public interface IPolynomialSolver {
    /**
     * Set polynomial terms (coefficients & exponents)
     * @param poly
     *           name of the polynomial
     * @param terms
     *           array of [coefficients][exponents]
     */
    void setPolynomial(char poly, int[][] terms);

    /**
     * Print the polynomial in ordered human readable representation
     * @param poly
     *           name of the polynomial
     * @return polynomial in the form like 27x^2+x-1
     */
    String print(char poly);

    /**
     * Clear the polynomial
     * @param poly
     *           name of the polynomial
     */
    void clearPolynomial(char poly);

    /**
     * Evaluate the polynomial
     * @param poly
     *           name of the polynomial
     * @param the
```

```
     *           polynomial constant value
     * @return the value of the polynomial
     */
    float evaluatePolynomial(char poly, float value);

    /**
     * Add two polynomials
     * @param poly1
     *           first polynomial
     * @param poly2
     *           second polynomial
     * @return the result polynomial
     */
    int[][] add(char poly1, char poly2);

    /**
     * Subtract two polynomials
     * @param poly1
     *           first polynomial
     * @param poly2
     *           second polynomial
     * @return the result polynomial
     */
    int[][] subtract(char poly1, char poly2);

    /**
     * Multiply two polynomials
     * @param poly1
     *           first polynomial
     * @param poly2
     *           second polynomial
     * @return the result polynomial
     */
    int[][] multiply(char poly1, char poly2);
}
```

Your program can look like the following:

```
Please choose an action
-----------------------
1- Set a polynomial variable
2- Print the value of a polynomial variable
3- Add two polynomials
4- Subtract two polynomials
```

```
5- Multiply two polynomials
6- Evaluate a polynomial at some point
7- Clear a polynomial variable
======================================================================
1
Insert the variable name: A, B or C
A
Insert the polynomial terms in the form:
(coeff1, exponent1), (coeff2, exponent2), ..
(1, 1), (1, 0)
Polynomial A is set
======================================================================
Please choose an action
1- Set a polynomial variable, ... etc
======================================================================
1
Insert the variable name: A, B or C
B
Insert the polynomial terms in the form:
(coeff1, exponent1), (coeff2, exponent2), ..
(1, 1), (-1, 0)
======================================================================
Please choose an action
1- Set a polynomial variable, ... etc
======================================================================
5
Insert first operand variable name: A, B or C
C
Variable not set
Insert the first operand variable name: A, B or C
A
Insert the second operand variable name: A, B or C
B
Result set in R: (1, 2), (-1, 0)
======================================================================
Please choose an action
1- Set a polynomial variable, ... etc
======================================================================
2
Insert the variable name: A, B, C or R
R Value in R: x^2 - 1
```

5 Deliverables

- Develop this assignment in Java language.
- You should work in group of 2.
- You should follow the given interface specified in the Definition section.
- Take into consideration that your implementation will be used later in the project, so it has to be fully functional, well documented and reusable. Try very hard to clean up your implementation. Remove all unused variables. Do not write redundant and repeated code. You may use Checkstyle http://checkstyle.sourceforge.net/ with your IDE to ensure that your code style follows the JAVA coding style standards (choose **Sun Checks** from the preferences).
- You should push your code to the repository.
- You should try out your code using http://onlinetester.tk/
- Test your implementation using **JUnit 4 Test**. **You should push 3 JUnitTest classes**. Your JUnit tests **will be graded** based on coverage for the different cases.
- Deliver your Polynomial Solver application as an executable JAR (JRE 1.8), under a folder called "**Materials/PolynomialSolver**" besides the **src** folder in your repository.
- Late submission is accepted for only one week.
- Delivering a copy will be severely penalized for both parties, so delivering nothing is so much better than delivering a copy.

Good Luck :)