Practical Exercise 6 – Lists, Stack and Queue

Overall Objective

To design and implement a list using an array and a linked structure.
To design and implement application using stack and queue.

Background

You will need to know:

1. basic Java programming knowledge
2. classes and interfaces
3. generics
4. array and linked list
5. stack and queue concepts

Description

Programming Exercise

List

1. ***Add set operations in* `MyList`**
   Define the following methods in `MyList` and implement them in `MyAbstractList`:

```
/** Adds the elements in otherList to this list.
  * Returns true if this list changed as a result of the call */
public boolean addAll(MyList<E> otherList);

/** Removes all the elements in otherList from this list.
  * Returns true if this list changed as a result of the call */
public boolean removeAll(MyList<E> otherList);

/** Retains the elements in this list that are also in otherList.
  * Returns true if this list changed as a result of the call */
public boolean retainAll(MyList<E> otherList);
```

   Write a test program that creates two `MyArrayLists`, `list1` and `list2` as follows:

```
list1 = {"Tom", "George", "Peter", "Jean", "Jane"}
list2 = {"Tom ", "George", "Michael", "Michelle", "Daniel"}
```

   And perform the following operations:

   - Invokes `list1.addAll(list2)`, and displays `list1` and `list2`.
   - Invokes `list1.retainAll(list2)`, and display `list1` and `list2`.
   - Invokes `list1.removeAll(list2)`, and display `list1` and `list2`.

   Guideline:

   1. Define `MyList` interface that extends `java.lang.Iterable`
      1.1 Add `addAll()`, `retainAll()` and `removeAll()` methods in `MyList` interface.
   2. Define `MyAbstractList` abstract class that implements `MyList` interface *(refer to slide 6).*
      2.1 Implement `addAll()`, `retainAll()` and `removeAll()` methods in `MyAbstractList` abstract class.
   3. Define `MyArrayList` concrete class that extends `MyAbstractList` abstract class *(refer to slide 11).*
      3.1 Define an inner class `ArrayListIterator` that implements `java.util.Iterator`

3.2 Override `toString` method to return the values of all elements as string.

4. Write the test program to test `addAll()`, `retainAll()` and `removeAll()` methods

Note that you may need to call:

- `add(E e)` and `get(int index)` methods in the `addAll()` method;
- `remove(E e)` and `get(int index)` methods in the `removeAll()` method;
- `contains(E e)`, `remove(E e)` and `get(int index)` methods in `retainAll()` method.

The following ***program skeleton*** may help:

```
import java.util.*;

public class TestMyList {
  public static void main(String[] args) {
    new TestMyList();
  }

  public TestMyList() {
    // HERE: write your test program
  }

  public interface MyList<E> extends java.lang.Iterable {
    // HERE: define abstract methods

    // as requested in the question
    /** Adds the elements in otherList to this list.
      * Returns true if this list changed as a result of the call */
    public boolean addAll(MyList<E> otherList);

    /** Removes all the elements in otherList from this list
      * Returns true if this list changed as a result of the call */
    public boolean removeAll(MyList<E> otherList);

    /** Retains the elements in this list that are also in otherList
      * Returns true if this list changed as a result of the call */
    public boolean retainAll(MyList<E> otherList);
  }

  public abstract class MyAbstractList<E> implements MyList<E> {
    // HERE: declare size variable and initialise it with zero
    //       include 2 constructors
    //       provide partial implementation: add(E e), isEmpty(),
    //                                size() and remove(E e) methods
    //       implement addAll(), removeAll() and retainAll()
  }

  public class MyArrayList<E> extends MyAbstractList<E> {
    // HERE: implement all unimplemented abstract methods

    private class ArrayListIterator implements java.util.Iterator<E> {
      // HERE: implement all methods in Interator interface
    }
  }
}
```

2. ***Implement* `MyLinkedList`**
   Implement the following methods in `MyLinkedList` given in the lecture slide (Ch26):

   - `contains(E e)` : Returns true if this list contains the element e.
   - `get(int index)` : Returns the element in this list at the specified index.
   - `indexOf(E e)` : Returns the index of the first matching element.
   - `lastIndexOf(E e)` : Returns the index of the last matching element.
   - `set(int index, E e)` : Replace the element at the specified position in this list with the specified element.

   And write a test program that tests the above methods.

   Guideline:
   1. Define `MyList` interface that extends `java.lang.Iterable`
   2. Define `MyAbstractList` abstract class that implements `MyList` interface *(refer to slide 6).*
   3. Define `MyLinkedList` concrete class that extends `MyAbstractList` abstract class *(refer to slide 20).*
      3.1 Define an inner class `Node` *(refer to slide 14).*
      3.2 Define an inner class `LinkedListIterator` that implements `java.util.Iterator`
      3.3 Override `toString` method to return the values of all elements as string.
      3.4 Add and implement `contains()`, `get()`, `indexOf()`, `lastIndexOf()` and `set()` methods in `MyLinkedList` concrete class.
   4. Write the test program to test **`contains()`, `get()`, `indexOf()`, `lastIndexOf()` and `set()` methods**.

   The following ***program skeleton*** may help:

```
public class TestMyLinkedList {
    public static void main(String[] args) {
        new TestMyLinkedList();
    }

    public TestMyLinkedList() {
        // HERE: write your test program
    }

    public interface MyList<E> extends java.lang.Iterable {
        // HERE: define abstract methods
    }


    public abstract class MyAbstractList<E> implements MyList<E> {
        // HERE: declare size variable and
        //       provide partial implementation
    }

    public class MyLinkedList<E> extends MyAbstractList<E> {
        // HERE: implement all unimplemented abstract methods

        private class LinkedListIterator implements
                java.util.Iterator<E> {
            // HERE: implement all methods in Interator interface
        }
```

```
        private class Node<E> {
            // HERE: Define node and its implementation
        }
    }
}
```

Note:

Note that it is not necessary for you to follow the program skeletons given above. You may provide alternative solution for your answer.

Stack

3. Write a test program that inputs a line of text, and uses a queue and a stack to print the line reversed.

   The program creates and initializes an empty queue `Q` and an empty stack `S`. After `Q` and `S` have been created, the program asks user for a line of text and insert each of the character into the queue `Q`. The program then reverses the order of the items in `Q` by using the empty stack `S`. The final output of the program should print out the line of text in reversed order.

Queue

4. One way to evaluate a prefix expression is to use a ***queue***. To evaluate the expression, scan it repeatedly until you know the final expression value. In each scan, read the tokens (operands or operators) and replace the operator that is followed by two operands with their calculated values before store them in a queue. For example, the following expression is a prefix expression that is evaluated to 159:

   – + * 9 + 2 8 * + 4 8 6 3

   We scan the expression and store it in a queue. During the scan, when an operator is followed by two operands, such as **+ 2 8**, we put the result, 10, in the queue.

   After the first scan, we have: **–** + * 9 10 * 12 6 3
   After the second scan, we have: **–** + 90 72 3
   After the third scan, we have: **–** 162 3
   After the forth scan, we have: 159

   Algorithm evaluatePrefix (prefixQ)
   This algorithm scans the prefix queue repeatedly until the final expression value is calculated.
   Pre *prefixQ have been filled with prefix expression*
   Post *the final expression value calculated*

```
1  loop ( size of prefixQ > 1 )
    1  loop ( not empty of prefixQ )
            1  dequeue an item from prefixQ
            2  if ( the item is operand )
                    1  enqueue operand to tempQ
            3  else  // the item is operator
                    1  if ( front item of prefixQ is not operand )
                            1  enqueue operator to tempQ
                    2. else
                            1  dequeue operand1 from prefixQ
```

```
                    2  if ( front item of prefix queue is not operand )
                            1  enqueue operator to tempQ
                            2  enqueue operand1 to tempQ
                    3  else
                            1  dequeue operand2 from prefixQ
                            2  calculate the operator with two operands
                            3  enqueue calculated value to tempQ
                    4  end if
                3 end if
            4  end if
        2  end loop
        3  copy tempQ to prefixQ
    2  end loop
    end evaluatePrefix
```

Implement the algorithm given above.

Useful guideline for Part 2:

- Note that you may use `GenericStack` and `GenericQueue` classes given in Ch26 and/or `Stack` and `LinkedList` classes in Java Collection Framework to do the above questions.

- Useful methods to be used in Question 1 & 2:
  - To find out length of the string: `public int length()`
  - To retrieve the character located at the String's specified index:
    `public char charAt(int index)`

    Example:

```
public class Test {

   public static void main(String args[]) {
      String s = "Strings are immutable";
      int len = s.length();
      char result = s.charAt(8);
      System.out.println("length = " + len);
      System.out.println("Character at index 8 = " + result);
   }
}
```

- For Question 2, add on the following method to `GenericQueue` class if the class is used:

```
public E getFront() {
    return list.getFirst();
}
```

- Useful methods to be used in Question 2:
  - To determine whether the specified char value is a digit:
    `boolean isDigit(char ch)`

    Example:

```
public class Test {

   public static void main(String args[]) {
      System.out.println(Character.isDigit('c'));
      System.out.println(Character.isDigit('5'));
   }
}
```

- To get the primitive data type of a certain String:

```
static int parseInt(String s)
```

Example:

```
public class Test{

   public static void main(String args[]){
      int x =Integer.parseInt("9");
      double y = Double.parseDouble("5");

      System.out.println(x);
      System.out.println(y);
   }
}
```

- To convert number to String, simply add " + "" " after the number.

Example:

```
String txtNum = 123 + "";
```