List of Classes:

- Node
- Polynomial
- Project1
- TextFileInput

Output:

```
//**Output starts***********************************************************************
Sum is: 2 2
Product is: 1 2 1


Sum is: 2 0
Product is: 1 0 -1


Sum is: 1 0 1 2
Product is: 0 0 0 2 0 2


Sum is: 7 7 7 7 7 7
Product is: 6 17 32 50 70 91 70 50 32 17 6


Sum is: 0 0 0 0
Product is: -4 12 -9 4 -6 0 -1


Sum is: 0 0 0 0 0 0 0
Product is: -1 -4 -10 -20 -35 -56 -84 -104 -115 -116 -106 -84 -49


Sum is: 1 0 0 0 0 15 0 0 0 0 19
Product is: 0 0 0 0 0 15 0 0 0 0 7 0 0 0 0 180 0 0 0 0 84


Sum is: -1 16 -2 0 13 2 5 15 7 4 0 9 12
Product is: -2 -23 43 -6 4 135 -9 80 222 85 53 56 180 164 -16 27 54 24


*******All lines are Done in the Text File!*******
//**Output ends*************************************************************************
```

```
DataFile.txt
1 1
1 1

1 1
1 -1

0 0 0 2
1 0 1

1 2 3 4 5 6
6 5 4 3 2 1

-2 3 0 1
2 -3 0 -1

1 2 3 4 5 6 7
-1 -2 -3 -4 -5 -6 -7

1 0 0 0 0 0 0 0 0 0 12
0 0 0 0 0 15 0 0 0 0 7

1 13 -2 0 3 2
-2 3 0 0 10 0 5 15 7 4 0 9 12
```

```java
/**
 * CS313 Summer 2013
 * Project 1
 *
 * @author Youchen Ren
 */
public class Node {
      private int element;
      private Node next;
//Constructors***************************
      public Node (int e, Node n) {
            element = e;
            next = n;
      }
      public Node (int e) {
            this(e, null);
	}
//update methods************************
	public void setElement(int e) {
		element = e;
	}
	public void setNext(Node n) {
		next = n;
	}
//access methods************************
	public int getElement() {
		return element;
	}
	public Node getNext() {
		return next;
	}
}
```

```
/**
 * CS313 Summer 2013
 * Project 1
 *
 * @author Youchen Ren
 */
public class Polynomial {
	private Node head = new Node(0);
	private int size;
//Constructors***************************************
	Polynomial (){
		head.setNext(null);
		size = 0;
	}
	Polynomial (Node h){
		head = h;
		size = 0;
	}
//Methods********************************************
	public int size() {return size;}

	public void append(int e) {
		Node n = new Node(e);
		if (size == 0) head = n;
		else {
			Node cur = head;
			while(cur.getNext() != null) {
				cur = cur.getNext();
			}
			cur.setNext(n);
		}
		size++;
	}
	public static void print(Polynomial p) {
		Node n = p.head;
		while(n != null) {
			System.out.print(n.getElement()+" ");
			n = n.getNext();
```

```
            }
        }
        public Node getFront() {
            return this.head;
        }
    //Method of "sum" & "product"*****************************************************
        public static Polynomial sum(Polynomial p1, Polynomial p2) {
            Node p1h = p1.head;
            Node p2h = p2.head;
            Node sumPh = new Node(0);
            Node newNode;
            Polynomial sumP = new Polynomial(sumPh);
            Node temp = sumPh;
            int s, p1int = 0, p2int = 0;
            boolean flag = true;
            while(flag) {
                if (p1h == null && p2h != null) {
                    p1int = 0;
                    p2int = p2h.getElement();
                }
                else if (p2h == null && p1h != null) {
                    p2int = 0;
                    p1int = p1h.getElement();
                }
                else {
                    p1int = p1h.getElement();
                    p2int = p2h.getElement();
                }
                s = p1int + p2int;
                temp.setElement(s);
                if (p1h == null && p2h != null) {
                    p2h = p2h.getNext();
                }
                else if (p2h == null && p1h != null) {
                    p1h = p1h.getNext();
                }
                else {
                    p1h = p1h.getNext();
                    p2h = p2h.getNext();
                }
                if(p1h == null && p2h == null) {
                    flag = false;
                }
                else{
                    newNode = new Node(0);
                    temp.setNext(newNode);
                    sumP.size++;
                    temp = temp.getNext();
                }
            }//while
            return sumP;
        }//sum method
        public static Polynomial product(Polynomial p1, Polynomial p2) {
            Node p1h = p1.head;
            Node p2h = p2.head;
            Node proPh = new Node(0);
            Node newNode;
            Polynomial productP = new Polynomial(proPh);
            Node temp = proPh;
            int CompuCount = 0;
            while(p1h != null) {
                  while(p2h != null) {
                        temp.setElement((p1h.getElement()*p2h.getElement()) +
                                                          temp.getElement());
                        if((p2h.getNext()!= null) && (temp.getNext() == null)) {
                              newNode = new Node(0);
                              temp.setNext(newNode);
                              productP.size++;
                        }
                        p2h = p2h.getNext();
                        temp = temp.getNext();
                  }
                  p2h = p2.head;
                  p1h = p1h.getNext();
                  temp = productP.head;
                  CompuCount++;
                  for (int i = 1; i <= CompuCount; i++) {
                        temp = temp.getNext();
                  }
            }
            return productP;
      }//product method
}//class Polynomial
```

```
/**
 * CS313 Summer 2013
 * Project 1
 *
 * @author Youchen Ren
 */
import java.util.StringTokenizer;

public class Project1 {
      public static TextFileInput myFile;
      public static StringTokenizer myTokens;
      public static StringTokenizer myTokens2;
      public static String line;

      public static void main(String[] args) {
            myFile = new TextFileInput("DataFile.txt");
            line = myFile.readLine();
            boolean flagWhile = true;
            String myTokens_Str;
            Polynomial resultSum;
            Polynomial resultProduct;
//while Loop for capture each pair of data and
//compute the sum and product**********************************************************

            while(flagWhile) {//Consider the \n situation.
```

```
                Polynomial temp1 = new Polynomial();
                Polynomial temp2 = new Polynomial();

                temp1.getFront().setNext(null);
                temp2.getFront().setNext(null);
                temp1.getFront().setElement(0);
                temp2.getFront().setElement(0);

                myTokens = new StringTokenizer(line, " ");
                myTokens_Str = myTokens.nextToken();//myTokens_Str = "1";

                boolean flag = true;
                while(flag) {
                      temp1.append(Integer.parseInt(myTokens_Str));
                      if(!(myTokens.hasMoreTokens())) flag = false;
                      else myTokens_Str = myTokens.nextToken();
                }//while

                line = myFile.readLine();

                myTokens2 = new StringTokenizer(line, " ");
                String myTokens_Str2 = myTokens2.nextToken();

                boolean flag2 = true;

                while(flag2) {
                      temp2.append(Integer.parseInt(myTokens_Str2));
                      if(!(myTokens2.hasMoreTokens())) flag2 = false;

                      else myTokens_Str2 = myTokens2.nextToken();
                }
                resultSum = new Polynomial();
                resultSum = Polynomial.sum(temp1, temp2);
//Test for output*************************************************************
                System.out.print("Sum is: ");
                Polynomial.print(resultSum);
                System.out.println();
                resultProduct = new Polynomial();
                resultProduct = Polynomial.product(temp1, temp2);

                System.out.print("Product is: ");
                Polynomial.print(resultProduct);
                System.out.println("\n\n");

                line = myFile.readLine();
                if (line == null){
                      System.out.println("*******All lines are Done in the Text
                                                          File!*******");
                      flagWhile = false;
                }
                else if(line.isEmpty()) {
                      line = myFile.readLine();
                }
            }//while
        }//main
```

```
}//class Project1
```

```
// TextFileInput.java
// Copyright (c) 2000, 2005 Dorothy L. Nixon.  All rights reserved.

import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.InputStreamReader;
import java.io.IOException;

/**
 * Simplified buffered character input
 * stream from an input text file.
 * Manages an input text file,
 * handling all IOExceptions by generating
 * RuntimeExcpetions (run-time error
 * messages).
 *
 * If the text file cannot be created,
 * a RuntimeException is thrown,
 * which by default results an an
 * error message being printed to
 * the standard error stream.
 *
 * @author D. Nixon
 */
public class TextFileInput  {

   /**  Name of text file  */
   private String filename;

   /**  Buffered character stream from file  */
   private BufferedReader br;

   /**  Count of lines read so far.  */
   private int lineCount = 0;

   /**
    * Creates a buffered character input
    * strea, for the specified text file.
    *
    * @param filename the input text file.
    * @exception RuntimeException if an
    *          IOException is thrown when
    *          attempting to open the file.
    */
   public TextFileInput(String filename)
   {
      this.filename = filename;
      try  {
         br = new BufferedReader(
                  new InputStreamReader(
                      new FileInputStream(filename)));
      } catch ( IOException ioe )  {
         throw new RuntimeException(ioe);
      }  // catch
   }  // constructor

   /**
    * Closes this character input stream.
    * No more characters can be read from
    * this TextFileInput once it is closed.
    * @exception NullPointerException if
    *        the file is already closed.
    * @exception RuntimeException if an
    *       IOException is thrown when
    *       closing the file.
    */
   public void close()
   {
      try  {
         br.close();
         br = null;
      } catch ( NullPointerException npe )  {
         throw new NullPointerException(
                        filename + "already closed.");
      } catch ( IOException ioe )  {
         throw new RuntimeException(ioe);
      }  // catch
   }  // method close

   /**
    * Reads a line of text from the file and
    * positions cursor at 0 for the next line.
    * Reads from the current cursor position
    * to end of line.
    * Implementation does not invoke read.
    *
    * @return the line of text, with
    *         end-of-line marker deleted.
    * @exception RuntimeException if an
    *          IOException is thrown when
    *          attempting to read from the file.
    */
   public String readLine()
   {
      return readLineOriginal();
   }  // method readLine()

   /**
    * Returns a count of lines
    * read from the file so far.
    */
   public int getLineCount()  { return lineCount; }

   /**
    * Tests whether the specified character is equal,
    * ignoring case, to one of the specified options.
    *
    * @param toBeChecked the character to be tested.
    * @param options a set of characters
    * @return true if <code>toBeChecked</code> is
    *         equal, ignoring case, to one of the
    *         <code>options</code>, false otherwise.
    */
   public static boolean isOneOf(char toBeChecked,
                                 char[] options)
   {
      boolean oneOf = false;
      for ( int i = 0; i < options.length && !oneOf; i++ )
         if ( Character.toUpperCase(toBeChecked)
                   == Character.toUpperCase(options[i]) )
            oneOf = true;
      return oneOf;
   }  // method isOneOf(char, char[])

   /**
    * Tests whether the specified string is one of the
    * specified options.  Checks whether the string
    * contains the same sequence of characters (ignoring
    * case) as one of the specified options.
    *
    * @param toBeChecked the String to be tested
    * @param options a set of Strings
    * @return true if <code>toBeChecked</code>
    *         contains the same sequence of
    *         characters, ignoring case, as one of the
    *         <code>options</code>, false otherwise.
    */
   public static boolean isOneOf(String toBeChecked,
                                 String[] options)
   {
      boolean oneOf = false;
      for ( int i = 0; i < options.length && !oneOf; i++ )
         if ( toBeChecked.equalsIgnoreCase(options[i]) )
            oneOf = true;
      return oneOf;
   }  // method isOneOf(String, String[])

   /**
    * Reads a line from the text file and ensures that
    * it matches one of a specified set of options.
    *
    * @param options array of permitted replies
    *
    * @return the line of text, if it contains the same
    *         sequence of characters (ignoring case for
    *         letters) as one of the specified options,
    *         null otherwise.
    * @exception RuntimeException if the line of text
    *         does not match any of the specified options,
    *         or if an IOException is thrown when reading
    *         from the file.
    * @exception NullPointerException if no options are
    *         provided, or if the end of the file has been
```

```
     *         reached.
     */
    public String readSelection(String[] options)
    {
       if ( options == null || options.length == 0 )
          throw new NullPointerException(
                            "No options provided for "
                            + " selection to be read in file "
                            + filename + ", line "
                            + (lineCount + 1) + ".");

       String answer = readLine();

       if ( answer == null )
          throw new NullPointerException(
                            "End of file "
                            + filename + "has been reached.");

       if ( !TextFileInput.isOneOf(answer, options) )  {
          String optionString = options[0];
          for ( int i = 1; i < options.length; i++ )
             optionString += ", " + options[i];
          throw new RuntimeException("File " + filename
                            + ", line " + lineCount
                            + ": \"" + answer
                            + "\" not one of "
                            + optionString + ".");
       }  // if
       return answer;
   }  // method readSelection

    /**
     * Reads a line from the text file and ensures that
     * it matches, ignoring case, one of "Y", "N", "yes",
     * "no", "1", "0", "T", "F", "true", or "false".
     * There must be no additional characters on the line.
     *
     * @return <code>true</code> if the line matches
     *         "Y", "yes", "1" "T", or "true".
     *         <code>false</code> if the line matches
     *         "N", "no", "0", "F", or "false".
     * @exception RuntimeException if the line of text
     *         does not match one of "Y", "N", "yes",
     *         "no", "1", "0", "T", "F", "true", or "false",
     *         or if an IOException is thrown when reading
     *         from the file.
     * @exception NullPointerException if the end of the
     *         file has been reached.
     */
    public boolean readBooleanSelection()
    {
       String[] options = {"Y", "N", "yes", "no", "1", "0",
                           "T", "F", "true", "false"};
       String answer = readSelection(options);
       return isOneOf(answer,
                      new String[] {"Y", "yes", "1", "T", "true"} );
    }  // method askUserYesNo

    /**
     * Reads a line of text from the file and
     * increments line count.  (This method
     * is called by public readLine and is
     * final to facilitate avoidance of side
     * effects when public readLine is overridden.)
     *
     * @return the line of text, with
     *         end-of-line marker deleted.
     * @exception RuntimeException if an
     *          IOException is thrown when
     *          attempting to read from the file.
     */
    protected final String readLineOriginal()
    {
        try  {
           if ( br == null )
              throw new RuntimeException(
                                  "Cannot read from closed file "
                                  + filename + ".");
           String line = br.readLine();
           if ( line != null )
              lineCount++;
           return line;
        } catch (IOException ioe)  {
           throw new RuntimeException(ioe);
        }  // catch
    }  // method readLineOriginal
}  // class TextFileInput
```