```
//////////////////////////////////////////////////////
//www.karamian.com
//Source code from text.
//Program PostfixInterpreter
//////////////////////////////////////////////////////

package postfix;
import java.io.*;
import java.util.*;
import java.util.StringTokenizer;

//************************************************************************
//   Class PostfixInterpreter
//************************************************************************

public class PostfixInterpreter extends Object
{
   public String postfixString, outputString;

   //========================================================================
   //  The  isOperator method
   //========================================================================

   private boolean isOperator(char c)
   {
      return (c=='+' || c=='-' || c=='*' || c=='/' || c=='^' );

   }//end isOperator

   //========================================================================
   //  The  isSpace method
   //========================================================================

   private boolean isSpace(char c)
   {
      return (c== ' ');

   }//end isSpace

   //========================================================================
   //  The  interpretPostfix method
   //========================================================================

   public void interpretPostfix()throws IOException
   {
      BufferedReader br = new BufferedReader(new InputStreamReader (System.in));
      System.out.print("Enter the postfix expression: ");
      String readIt = br.readLine();
      Stack evalStack = new Stack();
      double leftOperand, rightOperand;
      char c;
      Object j=new Object();
      System.out.println("This is your postfix expression: "+ readIt);
      StringTokenizer parser = new StringTokenizer (readIt);
      //  StringTokenizer parser = new StringTokenizer (postfixString ,
"+-*/^",true);

      while (parser.hasMoreTokens())
      {
         String token = parser.nextToken();
         c = token.charAt(0);
         if ( (token.length() ==1) && isOperator(c))
         {
            rightOperand = ((Double)evalStack.pop()).doubleValue();
```

```
            leftOperand = ((Double)evalStack.pop()).doubleValue();
            switch (c)
            {
               case '+': j=evalStack.push(new Double(leftOperand +rightOperand));
               break;

               case '-': j=evalStack.push(new Double(leftOperand -rightOperand));
               break;

               case '*': j=evalStack.push(new Double(leftOperand *rightOperand));
               break;

               case '/': j=evalStack.push(new Double(leftOperand /rightOperand));
               break;

               case '^': j=evalStack.push(new Double(Math.exp(Math.log(leftOperand)
*rightOperand)));
               break;

               default:
               break;
            }//end switch
         System.out.println(leftOperand+ " "+ token+ " "+ rightOperand+ " = "+ j);
         }
         else if((token.length()==1) && isSpace(c))
         {
            ;
         }
         else
         {
            evalStack.push(Double.valueOf(token));
         }//end if
      }//end while
         //remove final result from stack and output it
        output("Value of postfix expression = " +evalStack.pop());

   }//end intepretPostfix


   //=================================================================
   //  The  output method
   //=================================================================

   private void output(String s)
   {
      outputString = s;

   }//end output


   //=================================================================
   //  The  setInput method
   //=================================================================

   private void setInput(String input)
   {
      postfixString = input;

   }//end setInput


   //=================================================================
   //  The  getOutput method
   //=================================================================
```

```
   public String getOutput()
   {
      return outputString;

   }//end getOutput

}//end class PostfixInterpreter


//**************************************************************************
//   Class Stack
//**************************************************************************

class Stack extends Object
{
   private int count;
   private int capacity;
   private int capacityIncrement;
   private Object[] itemArray;


   //========================================================================
   //  The Stack constructor.
   //  the following defines a no-arg constructor for Stack Object
   //========================================================================

   public Stack()
   {
      count = 0;
      capacity = 0;
      capacityIncrement = 10;
      itemArray =new Object [capacity];

   }//end Stack constructor

   //========================================================================
   //  The empty method.
   //========================================================================

   public boolean empty()
   {
      return (count == 0);

   }//end empty method.

   //========================================================================
   //  The push method.
   //========================================================================

   public Object push (Object X)
   {

   //if the itemArray does not have enough capacity,
   //expand the itemArray by tyhe capacity increment.

      if (count == capacity)
      {
         capacity += capacityIncrement;
         Object[] tempArray = new Object[capacity];
         for (int i = 0; i < count; i++)
         {
            tempArray [i] = itemArray[i];
         }
```

```
      itemArray = tempArray;
   }
   //insert the new item X at the end of the current item sequece
   //and increase the stack's count by one
   itemArray [count ++]=X;
   return X;
}//end push

//=========================================================================
//  The pop method.
//=========================================================================

public Object pop()
{
   if (count == 0)
   {
      return null;
   }
   else
   {
      return itemArray[--count];
   }

}//end pop

//=========================================================================
//  The peek method.
//=========================================================================

public Object peek()
{
   if (count == 0)
   {
      return null;
   }
   else
   {
      return itemArray[count-1];
   }

}//end peek

}//end class stack
```