```
// src/Calculator.java

import java.util.Scanner;
import com.github.sahasatvik.math.*;

/**
 * Calculator is a simple java application which parses mathematical expressions and evaluates
 * the result. Calculator can parse arithmetic operators as well as some functions, store variables,
 * and carry out some pre-defined commands.
 *
 * A complete manual on how to use this application can be found in the README files, or by entering
 * <code>/help</code> during runtime.
 *
 * A continuously updated version of this project can be found online at my
 * <a href="http://github.com/sahasatvik/Calculator">Github repository</a>.
 *
 *      @author         Satvik Saha
 *      @version        1.0, 17/10/2016
 *      @see            <a href="http://github.com/sahasatvik/Calculator">
 *                              http://github.com/sahasatvik/Calculator
 *                      </a>
 *      @see            com.github.sahasatvik.math.ExpressionParser
 *      @since          1.0
 */

public class Calculator {

        /**
         * Regex which matches a command. This is simply a string of characters following a forward
         * slash (<code>/</code>).
         */
        public static String commandRegex = "(\\s+)?(/)(.*)";

        /**
         * Stores the previously calculated answer. This can be retrieved during runtime as a variable.
         */
        public static String previousAns;

        /**
         * ExpressionParser object which contains methods for parsing arithmetic expressions.
         */
        public static ExpressionParser expParser;

        /**
         * Main method of Calculator.
```

```
 *
 *      @param  args                    the command-line arguments supplied to Calculator
 *      @since  1.0
 */
public static void main (String[] args) {

        /* Store the expression entered by the user */
        String expression;
        /* If the expression is a command, store it here */
        String command;
        /* Initialize the previous answer cache */
        previousAns = "";

        /* Startup message */
        System.out.print("\nCalculator by Satvik Saha");
        System.out.print("\n-------------------------");
        System.out.print("\n   An up-to-date version of Calculator can be found at : ");
        System.out.print("\n      https://github.com/sahasatvik/Calculator");
        System.out.print("\n");
        System.out.print("\n   Type  /help  to read a guide on how to use this program.");
        System.out.print("\n");

        /* Setup the input system */
        Scanner inp = new Scanner(System.in);

        /* Setup the ExpressionParser, which will parse the input */
        expParser = new ExpressionParser(32);
        /* Add some commonly used mathematical constants */
        expParser.addVariable("e", ("" + Math.E));
        expParser.addVariable("pi", ("" + Math.PI));
        expParser.addVariable("phi", ("" + (Math.sqrt(5.0) + 1.0) / 2.0));

        /* Start the input loop */
        while (true) {
                /* Display a simple prompt */
                System.out.print("\n?> ");

                /* Accept a line of input */
                expression = inp.nextLine().trim();

                /* Check whether the input is a command */
                if (expression.matches(commandRegex)) {
                        /* Extract the content of the command */
                        command = expression.substring(expression.indexOf("/") + 1).trim();

                        try {
```

```
                /* Parse the command */
                parseCommand(command);
        } catch (CommandNotFoundException e) {
                /* Display an error message if the command is not recognized */
                System.out.print("!> Command " + e.getCommand() + " not found !");
                System.out.print("\n   Try  /list  for a complete list of available commands.");
        }
        /* Go back to the start of the loop and get a new line of input */
        continue;
}

/* Enclose the expression processing section within a 'try' block */
try {
        /* Evaluate the expression and store it in the cache */
        previousAns = evaluate(expression);
        /* Display the result */
        System.out.print("=> " + previousAns);
} catch (NullExpressionException e) {
        /* Catch empty input */
        System.out.print("!> Null Expression !");
} catch (MissingOperandException e) {
        /* Catch input missing an operand */
        System.out.print("!> Missing operand to " + e.getOperator() + " !");
} catch (VariableNotFoundException e) {
        /* Catch input containing undefined variables */
        System.out.print("!> Variable " + e.getVar() + " not found !");
        System.out.print("\n   Try  /list vars  for a complete list of available variables.");
} catch (FunctionNotFoundException e) {
        /* Catch input containing unrecognized functions */
        System.out.print("!> Function " + e.getFunc() + "[] not found !");
        System.out.print("\n   Try  /list funcs  for a complete list of available functions.");
} catch (UnmatchedBracketsException e) {
        /* Catch input with unclosed brackets */
        System.out.print("!> Unmatched brackets in expression !");

        /* Display the expression entered */
        System.out.print("\n   " + e.getFaultyExpression());
        System.out.print("\n   ");

        /* Display a character pointing to where the unmatched bracket is */
        for (int i = 0; i < e.getIndexOfBracket(); i++) {
                System.out.print(" ");
        }
        System.out.print("^");
} catch (ExpressionParserException e) {
        /* Catch any other Exception encountered while parsing */
```

```
                        System.out.print("!> Invalid Expression !");
                }
        }
}

/**
 * Evaluate a mathematical expression and return the result.
 *
 *      @param  exp                     the expression to be parsed
 *      @return                         the evaluated result
 *      @throws com.github.sahasatvik.math.ExpressionParserException
 *                                      thrown when an exception is encountered while parsing
 *                                      the expression
 *      @see    com.github.sahasatvik.math.ExpressionParser#evaluate(String)
 *      @since  1.0
 */

public static String evaluate (String exp) throws ExpressionParserException {
        /* Substitute all indstances of '<ans>' with the previously evaluated answer in the cache */
        exp = exp.replaceAll("<(\\s+)?ans(\\s+)?>", previousAns);
        /* Return the expression as evaluated by the ExpressionParser library */
        return expParser.evaluate(exp);
}

/**
 * Parses a command intended for the Calculator shell.
 *
 *      @param  command                 the command to be parsed
 *      @throws CommandNotFoundException
 *                                      thrown when an unrecognized command is passed here
 *      @see    CommandNotFoundException
 *      @since  1.0
 */

public static void parseCommand (String command) throws CommandNotFoundException {
        if (command.equals("exit")) {
                /* If the commmand is '/exit', display an exit message exit the runtime */
                System.out.print("$> Exiting !");
                System.exit(0);
        } else if (command.equals("help")) {
                /* Display some general helptext */
                System.out.print("$> Calculator Helptext");
                System.out.print("\n   ~~~~~~~~~~~~~~~~~~");
                System.out.print("\n       Welcome to  'Calculator', a simple java application written to");
                System.out.print("\n    evaluate mathematical expressions.");
                System.out.print("\n       This program displays a prompt (?>), after which you can enter");
```

```
        System.out.print("\n    a mathematical expression. 'Calculator' will display the result,");
        System.out.print("\n    or point out errors in the expression.");
        System.out.print("\n");
        System.out.print("\n        'Calculator' can evaluate simple arithmetic expressions, using the");
        System.out.print("\n    operators (+, -, *, /, ^(power)), as well as parenthesis ('(', ')').");
        System.out.print("\n    'Calculator' follows the BODMAS rule.");
        System.out.print("\n");
        System.out.print("\n        Following are some valid expressions : ");
        System.out.print("\n            1 + 1                   =>              2.0");
        System.out.print("\n            1 * (2 + 3)             =>              5.0");
        System.out.print("\n            10 * (64 ^ -0.5)        =>             1.25");
        System.out.print("\n");
        System.out.print("\n        For help on more advanced topics, try entering the following : ");
        System.out.print("\n            /help vars              >       help on Variables");
        System.out.print("\n            /help funcs             >       help on Functions");
        System.out.print("\n            /help cmds              >       help on Commands");
        System.out.print("\n");
        System.out.print("\n        Enter '/list' for a complete list of valid commands.");
} else if (command.equals("help vars")) {
        /* Display help on 'variables' */
        System.out.print("\n$> Variables");
        System.out.print("\n   `````````");
        System.out.print("\n        'Calculator' can also store user-defined variables.");
        System.out.print("\n    The syntax for assigning and using variables is as follows : ");
        System.out.print("\n            var = value             >       assign 'value' to 'var'");
        System.out.print("\n            <var>                   >       <var> will be replaced");
        System.out.print("\n                                            its value.");
        System.out.print("\n");
        System.out.print("\n        Following are some valid uses of variables : ");
        System.out.print("\n            x = 3                   =>              3.0");
        System.out.print("\n            y = <x> + 1             =>              4.0");
        System.out.print("\n            (<x>^2 + <y>^2)^0.5     =>              5.0 ");
        System.out.print("\n");
        System.out.print("\n        Nesting of assignments is also supported, as follows : ");
        System.out.print("\n            x = 1 + (y = 1)         =>              2.0");
        System.out.print("\n            <x>                     =>              2.0");
        System.out.print("\n            <y>                     =>              1.0");
        System.out.print("\n");
        System.out.print("\n        A special variable <ans> stores the previous expression.");
        System.out.print("\n    Thus, the following is valid : ");
        System.out.print("\n            1 * 2 * 3 * 4           =>             24.0");
        System.out.print("\n            <ans> * 5               =>            120.0");
        System.out.print("\n");
        System.out.print("\n        Enter '/list vars' for a list of stored variables.");
} else if (command.equals("help funcs")) {
        /* Display help on 'funcitons' */
```

```
        System.out.print("\n$> Functions");
        System.out.print("\n   `````````");
        System.out.print("\n        'Calculator' supports the use of some basic functions.");
        System.out.print("\n    They can be used with the following syntax : ");
        System.out.print("\n            fnc[ value ]           >       evaluate 'fnc' of 'value'");
        System.out.print("\n");
        System.out.print("\n        Following are some valid uses of functions : ");
        System.out.print("\n            sin[<pi> / 2]          =>              1.0");
        System.out.print("\n            1 + abs[2 - 3]         =>              2.0");
        System.out.print("\n            log[<e> ^ 3]           =>              3.0");
        System.out.print("\n");
        System.out.print("\n        Enter '/list funcs' for a list of valid functions.");
} else if (command.equals("help cmds")) {
        /* Display help on 'commands' */
        System.out.print("\n$> Commands");
        System.out.print("\n   `````````");
        System.out.print("\n        'Calculator' interprets expressions starting with '/' as");
        System.out.print("\n    'commands'. These are special expressions which are not parsed as");
        System.out.print("\n    mathematical expressions, but as instructions to the 'Calculator'.");
        System.out.print("\n");
        System.out.print("\n        Enter '/list' for a complete list of valid commands.");
} else if (command.equals("list") || command.equals("list cmds")) {
        /* Display a list of valid, acceptable commands */
        System.out.print("$> Commands : \n");
        System.out.print("\n    /help                          >       general help");
        System.out.print("\n    /help vars                     >       help on Variables");
        System.out.print("\n    /help funcs                    >       help on Functions");
        System.out.print("\n    /help cmds                     >       help on Commands");
        System.out.print("\n    /list vars                     >       list variables");
        System.out.print("\n    /list funcs                    >       list functions");
        System.out.print("\n    /list cmds  or  /list          >       list commands");
        System.out.print("\n    /exit                          >       exit Calculator");
} else if (command.equals("list vars")) {
        /* Display a list of defined variables, currently stored in the ExpressionParser */
        System.out.print("$> Variables : \n");
        /* Loop through the variables in the array belonging to the ExpressionParser */
        for (int i = 0; i < expParser.numberOfVars; i++) {
                /* Pretty-print the variables */
                System.out.printf("%n\t%-16s=%30s", expParser.variables[i][0]
                                                 , expParser.variables[i][1]);
        }
        /* Display the previously evaluated answer as a special variable : 'ans' */
        System.out.printf("%n\t%-16s=%30s", "ans", previousAns);
} else if (command.equals("list funcs")) {
        /* Display a list of valid functions */
        System.out.print("$ Functions : \n");
```

```
                System.out.print("\n    abs[ x ]        >            absolute value of <x>");
                System.out.print("\n    exp[ x ]        >            exponent of <x> (<e> ^ <x>)");
                System.out.print("\n    log[ x ]        >            logarithm of <x> (base <e>)");
                System.out.print("\n    fct[ x ] or x!  >            factorial of <x>");
                System.out.print("\n    deg[ x ]        >            convert <x> to degrees from radians");
                System.out.print("\n    rad[ x ]        >            convert <x> to radians from degrees");
                System.out.print("\n                    _           ");
                System.out.print("\n    sin[ x ]         |          ");
                System.out.print("\n    cos[ x ]         |          ");
                System.out.print("\n    tan[ x ]          >         trigonometric functions");
                System.out.print("\n    csc[ x ]         |             ( <x> in radians )");
                System.out.print("\n    sec[ x ]         |          ");
                System.out.print("\n    ctn[ x ]         |          ");
                System.out.print("\n                    ~           ");
        } else {
                /* Throw an Exception if the command does not match any of the above */
                throw new CommandNotFoundException(command);
        }
    }
}
```

```
// src/CommandNotFoundException.java

/**
 * Exception thrown when an unrecognized command is passed to Calculator.
 *
 *      @author         Satvik Saha
 *      @version        1.0, 17/10/2016
 *      @see            Calculator
 *      @since          1.0
 */

public class CommandNotFoundException extends Exception {
        private String command;

        /**
         * Constructor of CommandNotFoundException.
         *
         *      @param  command                 the command which could not be parsed
         *      @since  1.0
         */

        public CommandNotFoundException (String command) {
                super("CommandNotFoundException");
                this.command = command;
        }

        /**
         * Get the command which could not be parsed.
         *
         *      @return                         the command which could not be parsed
         *      @since  1.0
         */

        public String getCommand () {
                return command;
        }
}
```

```
// src/com/github/sahasatvik/math/ExpressionParserException.java

package com.github.sahasatvik.math;

/**
 * Superclass of all Exceptions thrown by ExpressionParser.
 *
 *      @author         Satvik Saha
 *      @version        1.0, 16/10/2016
 *      @see            com.github.sahasatvik.math.ExpressionParser
 *      @since          1.0
 */

public class ExpressionParserException extends Exception {
        private String faultyExpression;

        /**
         * Constructor of ExpressionParserException.
         *
         *      @param  faultyExpression        the expression which could not be parsed
         *      @since  1.0
         */

        public ExpressionParserException (String faultyExpression) {
                super("ExpressionParserException");
                /* Store the bad expression */
                this.faultyExpression = faultyExpression;
        }

        /**
         * Gets the expression which could not be parsed.
         *
         *      @return                         the expression which could not be parsed
         *      @since  1.0
         */
        public String getFaultyExpression () {
                return faultyExpression;
        }
}
```

```
// src/com/github/sahasatvik/math/ExpressionParser.java

package com.github.sahasatvik.math;

/**
 * ExpressionParser provides methods for evaluating mathematical expressions, specifically
 * tailored for parsing with this library. ExpressionParser supports basic arithmetic operators,
 * parenthesized expressions, variable substitution as well as basic functions.
 *
 *      @author         Satvik Saha
 *      @version        1.0, 16/10/2016
 *      @see            com.github.sahasatvik.math.MathParser
 *      @since          1.0
 */

public class ExpressionParser extends MathParser {

        /**
         *  Regex which matches a number. It may be signed or use scientific notation.
         */
        protected static final String numberRegex = "(([+-]?)\\d+(\\.\\d+)?([eE](-?)\\d+)?)";

        /**
         * Regex which matches a signed number. It may use scientific notation.
         */
        protected static final String signedNumberRegex = "([+-]\\d+(\\.\\d+)?([eE](-?)\\d+)?)";

        /**
         * Regex which matches an assignment statement. It is simply a word, followed by an
         * equals sign (=) and an expression.
         */
        protected static final String assignmentRegex = "(\\s+)?(\\w+)(\\s+)(=)(.*)";

        /**
         * Array of supported operators. The operators are arranged in their order of precedence.
         * Thus, operators to the left will be evaluated before those to the right.
         */
        protected static final String[] operators = {"^", "%", "/", "*", "+", "-"};

        /**
         * Array of variables maintained by an ExpressionParser object. The first String in each
         * line stores the variable name, whicle the second stores the value.
         */
        public String[][] variables;
```

```
/**
 * Index of the last variable in the 'variables' array. Elements in the 'variables' array
 * after this index are all blank, so they are not parsed during expression evaluation.
 */
public int numberOfVars;


/**
 * Constructor of ExpressionParser. This constructor initializes the variable cache with
 * the specified maximum size.
 *
 *      @param  maxVars                 the maximum number of variables to be stored
 *      @since  1.0
 */

public ExpressionParser (int maxVars) {
        variables = new String[maxVars][2];
        numberOfVars = 0;
}

/**
 * Adds a variable to the variable cache. This method accepts the variable
 * name, as well as the String each occurrence is to be substituted with.
 *
 *      @param  name                    the name of the variable
 *      @param  value                   the value the variable holds
 *      @since  1.0
 */

public void addVariable (String name, String value) {
        /* Loop through the stored variables */
        for (int i = 0; i < numberOfVars; i++) {
                /* If the variable already exists, simply update the value */
                if (variables[i][0].equals(name)) {
                        variables[i][1] = value;
                        return;
                }
        }
        /*
         * Create a new variable by storing the name and value in the variables array,
         * then update numberOfVars
         */
        variables[numberOfVars][0] = name;
        variables[numberOfVars][1] = value;
        numberOfVars++;
}
```

```
/**
 * Evaluates a String representation of a mathematical expression into a
 * number (stored as a String).
 *
 *      @param  exp                     the expression to be evaluated
 *      @return                         the result after evaluating the expression
 *      @throws com.github.sahasatvik.math.NullExpressionException
 *                                      thrown when the expression is empty
 *      @throws com.github.sahasatvik.math.ExpressionParserException
 *                                      thrown when the expression cannot be parsed
 *      @see    #addVariable(String, String)
 *      @see    #parseVariables(String)
 *      @see    #parseParenthesis(String)
 *      @see    #parseFunctions(String)
 *      @see    #parseOperators(String)
 *      @since  1.0
 */

public String evaluate (String exp) throws ExpressionParserException {
        String result = exp;
        if (exp.trim().length() == 0) {
                /* Throw an Exception if the expression is blank */
                throw new NullExpressionException();
        } else if (isNumber(exp)) {
                /* If the expression is already a number, there is nothing to evaluate */
                return "" + Double.parseDouble(exp);
        } else if (exp.matches(assignmentRegex)) {
                /*
                 * If the expression is an assignment statement, interpret everything before
                 * the equals sign (=) as the variable name. The rest is simply another
                 * expression, which is also the value of the variable.
                 */
                String varName = exp.substring(0, exp.indexOf("=")).trim();
                String varValue = evaluate(exp.substring(exp.indexOf("=") + 1));

                /*
                 * Add the variable in the cache, then use the value of the variable
                 * as the evaluated result
                 */
                addVariable(varName, varValue);
                exp = varValue;
        } else {
                /*
                 * Replace all variables with their values,
                 * solve everything within parenthesis,
```

```
                 * then Solve all functions
                 */
                exp = parseVariables(exp);
                exp = parseParenthesis(exp);
                exp = parseFunctions(exp);

                /*
                 * The expression is now simply a collection of numbers and arithmetic operators.
                 * Finish off the process by solving each operation, following the BODMAS rule.
                 */
                exp = parseOperators(exp);
        }

        try {
                /* Check if the result is a valid number */
                result = "" + Double.parseDouble(exp);
        } catch (Exception e) {
                /* Throw an Exception if the result is not a number */
                throw new ExpressionParserException(exp);
        }
        return result;
}

/**
 * Substitutes all instances of the variables in the cache with their values.
 * A variable name present in the expression must be enclosed within angled brackets
 * (<code>&#60;</code>, <code>&#62;</code>) in order to be recognized.
 * For example, if <code>x = 10.0</code>, then all instances of <code>&#60;x&#62;</code>
 * will be replaced with <code>10.0</code>
 *
 *      @param  exp                     the expression to be parsed
 *      @return                         the expression after substituting known values
 *                                      of variables stored in the cache
 *      @throws com.github.sahasatvik.math.VariableNotFoundException
 *                                      thrown when an unrecognized variable name is
 *                                      found in the expression
 *      @since  1.0
 */

protected String parseVariables (String exp) throws VariableNotFoundException {
        /*
         * Loop through the variable cache, checking for occurrences of the variables
         * (enclosed within angled brackets)(<var_name>)
         */
        for (int i = 0; i < numberOfVars; i++) {
                /* Replace all instances of the variable with its value directly */
```

```
                exp = exp.replaceAll("<(\\s+)?" + variables[i][0] + "(\\s+)?>"
                                                , variables[i][1]);
        }

        /*
         * Check if any unrecognized variables are present. This can be done very simply as
         * the presence of angled brackets (<>) indicates an unreplaced variable.
         */
        int start = exp.indexOf("<");
        int end = exp.indexOf(">");
        if (start != -1 && end != -1 && start < end) {
                /*
                 * Extract the unreplaced variable name, which is clearly in between the angled
                 * brackets, then throw an Exception.
                 */
                throw new VariableNotFoundException(exp, exp.substring(start, end + 1));
        }

        /* Adjust the number spacing before passing the expression back to the evaluater */
        exp = adjustNumberSpacing(exp);
        return exp.trim();
}

/**
 * Substitutes expressions within parenthesis (<code>(</code>, <code>)</code>) with their results.
 * This ensures that while evaluating an expression containing parenthesized parts, those
 * parenthesized parts are evaluated first. This is done so that ExpressionParser follows the
 * BODMAS rule.
 *
 *      @param  exp                     the expression to be parsed
 *      @return                         the expression such that all parenthesized parts
 *                                      have been evaluated
 *      @throws com.github.sahasatvik.math.UnmatchedBracketsException
 *                                      thrown when brackets in the expression are not
 *                                      closed
 *      @throws com.github.sahasatvik.math.ExpressionParserException
 *                                      thrown if the parenthesized sections cannot be
 *                                      parsed
 *      @see    #indexOfMatchingBracket(String, int, char, char)
 *      @since  1.0
 */

protected String parseParenthesis (String exp) throws ExpressionParserException {
        String result = "";
        /*
         * Buffer the extreme ends with spaces, to make sure no Exceptions are thrown
```

```
         * while extracting portions of the expression.
         */
        exp = " " + exp + " ";

        /* Continue replacing parenthesized sections as long as a parenthesis is present */
        while (exp.indexOf("(") != -1) {
                /* Store the indices of the opening and closing parenthesis */
                int start = exp.indexOf("(");
                int end = indexOfMatchingBracket(exp, start, '(', ')');
                /* The enclosed section is simply another expression. Pass it to the evaluater */
                result = evaluate(exp.substring(start + 1, end));

                /*
                 * This is a special case. Make sure that ' -(some_expression) ' is interpreted
                 * as the negative of that expression.
                 */
                if (exp.charAt(start - 1) == '-') {
                        /* Multiply the enclosed section by -1, then evaluate the result */
                        result = " ( -1 * ( " + result + " ) ) ";
                        start--;
                }

                /* Graft the evaluated parenthesized portion back into the original expression */
                exp = exp.substring(0, start) + " "             // before the opening bracket
                                + result + " "                  // evaluated part
                                + exp.substring(end + 1);   // after the closing bracket
        }
        /* Adjust the number spacing before passing the expression back to the evaluater */
        exp = adjustNumberSpacing(exp);
        return exp.trim();
}

/**
 * Substitutes all occurrences of supported mathematical functions with their result.
 * A function must be present in the expression in the following format :
 * <code>function_name[function_argument]</code>, where the function argument can also
 * be an expression. The function name must be exactly 3 characters long, and be
 * immediately followed by a sqaure bracket (<code>[</code>).
 * See {@link com.github.sahasatvik.math.MathParser#solveUnaryFunction(String, double)} for a
 * list of supported function names.
 *
 *      @param  exp                     the expression to be parsed
 *      @return                         the expression such that all instances of
 *                                      functions are evaluated
 *      @throws com.github.sahasatvik.math.MissingOperandException
 *                                      thrown if there is no function argument
```

```
 *      @throws com.github.sahasatvik.math.FunctionNotFoundException
 *                                      thrown when an unrecognized function name
 *                                      is found in the expression
 *      @throws com.github.sahasatvik.math.UnmatchedBracketsException
 *                                      thrown when a square bracket is not closed
 *      @throws com.github.sahasatvik.math.ExpressionParserException
 *                                      thrown if the function argument cannot be
 *                                      parsed
 *      @see    com.github.sahasatvik.math.MathParser#solveUnaryFunction(String, double)
 *      @since  1.0
 */

protected String parseFunctions (String exp) throws ExpressionParserException {
        String result = "";
        String func = "";
        double x = 0.0;
        /*
         * Buffer the extreme ends with spaces, to make sure no Exceptions are thrown
         * while extracting portions of the expression.
         */
        exp = " " + exp + " ";
        try {
                /*
                 * This is another special case. Make sure that expressions of the form
                 * 'number!' are interpreted as the factorial of that number. This can
                 * be done simply by replacing all such cases with the expression 'fct[number]',
                 * as 'fct[]' is a valid function name and can be calculated later.
                 */
                exp = exp.replaceAll(numberRegex + "\\s+!", " fct[$1] ");

                /*
                 * Continue evaluating functions as long as square brackets ([]) are present.
                 * Here, a function is reperssented in the format 'fnc[expression]'. Thus, the
                 * presence of square brackets ([]) implies that a function is present.
                 */
                while (exp.indexOf("[") != -1) {
                        /* Store the indices of the opening and closing square brackets */
                        int start = exp.indexOf("[");
                        int end = indexOfMatchingBracket(exp, start, '[', ']');

                        /*
                         * Here, all function names are exactly 3 characters long. Thus, the
                         * function name is simply the 3 characters preceding the opening bracket.
                         */
                        func = exp.substring(start - 3, start);
```

```
                        /*
                         * The section enclosed within the brackets is also an expression.
                         * Evaluate it, and check whether it is a number. This will be the
                         * function argument.
                         */
                        x = Double.parseDouble(evaluate(exp.substring(start + 1, end)));

                        /* Pass the function name and argument to a function solver */
                        result = "" + solveUnaryFunction(func, x);

                        /*
                        * This is a special case similar to that in parseParenthesis(String).
                        * Make sure that ' -fnc[some_expression] ' is interpreted as the negative
                        * of the result of that function.
                        */
                        if (exp.charAt(start - 4) == '-') {
                                /* Multiply the enclosed section by -1, then evaluate the result */
                                result = evaluate(" ( -1 * ( " + result + " ) ) ");
                                start--;
                        }
                        /* Graft the evaluated portion back into the original expression */
                        exp = exp.substring(0, start-3) + " "           // before the function
                                        + result + " "              // evaluated part
                                        + exp.substring(end+1);     // after the function
                }
        } catch (NullExpressionException e) {
                /* Throw an Exceeption if the function is missing its argument */
                throw new MissingOperandException(exp, func + "[]");
        } catch (FunctionNotFoundException e) {
                /* Throw an Exception if an extracted function name is unsupported */
                throw new FunctionNotFoundException(exp, func);
        } catch (ExpressionParserException e) {
                /* Pass on any Exceptions encountered while evaluating the argument */
                throw e;
        } catch (Exception e) {
                /* Pass on any other Exceptions as ExpressionParserExceptions */
                throw new ExpressionParserException(exp);
        }
        /* Adjust the number spacing before passing the expression back to the evaluater */
        exp = adjustNumberSpacing(exp);
        return exp.trim();
}

/**
 * Substitutes all binary expressions involving arithmetic operators with their result.
 * Operations are performed following the BODMAS rule. The resultant parsed String
```

```
 * is free of all operators, thus containing only numbers.
 * See {@link com.github.sahasatvik.math.MathParser#solveBinaryOperation(double, String, double)}
 * for a list of supported operators. See {@link #operators}, which defines the order of
 * operations.
 *
 *      @param  exp                     the expression to be parsed
 *      @return                         the expression such that all arithmetic operations
 *                                      have been carried out
 *      @throws com.github.sahasatvik.math.MissingOperandException
 *                                      thrown if a binary operator is missing an operand
 *      @see    com.github.sahasatvik.math.MathParser#solveBinaryOperation(double, String, double)
 *      @since  1.0
 */

protected String parseOperators (String exp) throws MissingOperandException {
        int leftIndex, rightIndex;

        try {
                /*
                 * This code addresses a small problem in directly implementing BODMAS.
                 * Expressions like (1 - 2 + 3) will mistakenly evaluate to (-4) if the addition
                 * is done first, disregarding the minus sign before the (2).
                 * Eliminate this problem by changing all instances of subtraction to addition
                 * of the second operand's negative form. Thus, the minus sign acting as an operator
                 * now becomes part of the number itself, and all ambiguity disappears.
                 */
                while (exp.matches("(.*)" + numberRegex + "\\s+-\\s+" + numberRegex + "(.*)")) {
                        exp = exp.replaceAll(numberRegex + "\\s+-\\s+" + numberRegex, " $1 + -($6) ");
                exp = parseParenthesis(exp);
                }
        } catch (Exception e) {
                /* Something went seriously wrong - the expressions in the 'try' block were valid */
                System.out.print("You should never see this message. If you do, please inform the author.");
                e.printStackTrace();
        }

        /* Split the expression into a stack of operators and operands */
        String[] stack = exp.split("\\s+");

        /* Loop through all supported operators (in order) */
        for (String op : operators) {
                /* Loop through the stack, searching for a match with the operator */
                for (int i = 0; i < stack.length; i++) {
                        if (stack[i].equals(op)) {
                                leftIndex = rightIndex = i;
                                /* Keep on searching before the operator until a valid operand is found */
```

```
                                while (leftIndex >= 0 && !isNumber(stack[leftIndex]))
                                        leftIndex--;
                                /* Keep on searching after the operator until a valid operand is found */
                                while (rightIndex < stack.length && !isNumber(stack[rightIndex]))
                                        rightIndex++;
                                try {
                                        /* Get the operands */
                                        double left = Double.parseDouble(stack[leftIndex]);
                                        double right = Double.parseDouble(stack[rightIndex]);
                                        /*
                                         * Pass the operands and the operator to an operator solver,
                                         * then replace the operator with the result. Also remove the
                                         * operands.
                                         */
                                        stack[i] = "" + solveBinaryOperation(left, op, right);
                                        stack[leftIndex] = stack[rightIndex] = "";
                                } catch (Exception e) {
                                        /* Throw an Exception if there is a missing operand */
                                        throw new MissingOperandException(exp, op);
                                }
                        }
                }
        }
        exp = "";

        /* Recombine the stack into the solved expression */
        for (String s : stack) {
                exp += s;
        }
        return exp.trim();
}

/**
 * Adjusts the spacings between numbers, variables, functions, operators, etc in an expression.
 * Each number will be enclosed withhin a 'buffer' of spaces. Instances of signed numbers
 * immediately following anothoer number will be interpreted as their sum.
 * (<code>      1  -1</code> is simply <code>1 + -1</code>)
 *
 *      @param  exp                     the expression to be parsed
 *      @return                         the expression with adjusted spacing
 *      @since  1.0
 */

protected static String adjustNumberSpacing (String exp) {
        /* Make sure numbers are all spaced out from other symbols */
        exp = exp.replaceAll(numberRegex, " $0 ");
```

```
		/* Make sure the sign in signed numbers is also considered during addition/subtraction */
		exp = exp.replaceAll(numberRegex + "\\s+" + signedNumberRegex, " $1 + $6 ");
		return exp;
	}

	/**
	 * Finds the index of a matching closing bracket in a String, given the index of the
	 * opening one. This method can also be given any characters as opening and closing brackets.
	 * Nesting of brackets has also been dealt with.
	 *
	 *	@param  str                     the String containing the brackets
	 *	@param  pos                     the index of the opening bracket
	 *	@param  open                    the character to be recognized as an opening bracket
	 *	@param  close                   the character to be recognized as a closing bracket
	 *	@return                         the index of the matching closing bracket
	 *	@throws com.github.sahasatvik.math.UnmatchedBracketsException
	 *                                  thrown if the specified opening bracket is unclosed
	 *	@since 1.0
	 */

	protected static int indexOfMatchingBracket (String str, int pos, char open, char close)
                                                            throws UnmatchedBracketsException {
		int tmp = pos;
		/* Loop through the String, forward from the position of the opening bracket */
		while (++pos < str.length()) {
			/* Exit the loop as soon as a closing bracket is found */
			if (str.charAt(pos) == close)
				return pos;
			/*
			 * If another opening bracket is found, it becomes clear that bracketed expressions
			 * have been nested. Thus, the next closing bracket will not match the bracket
			 * we have targeted. In order to return the correct bracket, simply skip everything
			 * within the nested portion. This is done by calling
			 * indexOfMatchingBracket(String, int, char, char) recursively.
			 */
			if (str.charAt(pos) == open)
				pos = indexOfMatchingBracket(str, pos, open, close);
		}
		if (pos >= str.length()) {
			/* Throw an Exception if a matching bracket is not present */
			throw new UnmatchedBracketsException(str, tmp);
		}
		return pos;
	}
}
```

```java
// src/com/github/sahasatvik/math/FunctionNotFoundException.java

package com.github.sahasatvik.math;

/**
 * Exception thrown when an unparsable function is passed to ExpressionParser.
 *
 *      @author         Satvik Saha
 *      @version        1.0, 16/10/2016
 *      @see            com.github.sahasatvik.math.ExpressionParserException
 *      @since          1.0
 */

public class FunctionNotFoundException extends ExpressionParserException {
        private String func;

        /**
         * Constructor of FunctionNotFoundException. This constructor accepts the
         * invalid expression as well as the unrecognized function name.
         *
         *      @param  faultyExpression        the expression which could not be parsed
         *      @param  func                    the unrecognized function name
         */

        public FunctionNotFoundException (String faultyExpression, String func) {
                super(faultyExpression);
                this.func = func;
        }

        /**
         * Constructor of FunctionNotFoundException. This constructor accepts only
         * the unrecognized function name.
         *
         *      @param  func                    the unrecognized function name
         *      @since  1.0
         *
         */

        public FunctionNotFoundException (String func) {
                this("", func);
        }

        /**
         * Gets the unrecognized function name.
         *
```

```
         *      @return                         the unrecognized function name
         *      @since  1.0
         */

        public String getFunc () {
                return func;
        }
}
```

```java
// src/com/github/sahasatvik/math/MathParser.java

package com.github.sahasatvik.math;

/**
 * MathParser contains methods for solving simple operations involving
 * arithmetic operators and functions.
 * These methods can be accessed by subclasses of MathParser.
 *
 *      @author         Satvik Saha
 *      @version        1.0, 16/10/2016
 *      @since          1.0
 */

public class MathParser {

        /**
         * Checks whether the String passed to it can be parsed as a number.
         *
         *      @param  str                     the String to be tested
         *      @return                         true if the String can be parsed as a number
         *      @since  1.0
         */

        protected static boolean isNumber (String str) {
                try {
                        /* Return true only if parseDouble(String) doesn't complain! */
                        Double.parseDouble(str);
                        return true;
                } catch (Exception e) {
                        return false;
                }
        }

        /**
         * Calculates the factorial of a number.
         *
         *      @param  x                       the number whose factorial is to be calculated
         *      @return                         the factorial of the number passed
         *      @since  1.0
         */

        protected static double factorial (double x) {
                /* Special cases! */
                if (x < 2)
```

```
                return 1;
        double n = 1;
        while (x > 0)
                n *= x--;
        return n;
}

/**
 * Solves and returns the result of a simple binary expression.
 * Only the following operators are supported : <pre>{@code
 *      ^       -       power
 *      /       -       division
 *      *       -       multiplication
 *      +       -       addition
 *      -       -       subtraction}</pre>
 *
 *      @param  a                       the operand on the left
 *      @param  op                      the opearator
 *      @param  b                       the operand on the right
 *      @return                         the result on evaluating the expression
 *      @since  1.0
 */

protected static double solveBinaryOperation (double a, String op, double b) {
        double result = 0.0;
        /*
         * Match the operator against a list of supported ones, then
         * perform the appropriate operation.
         */
        if (op.equals("^")) {
                result = Math.pow(a, b);
        } else if (op.equals("%")) {
                result = a % b;
        } else if (op.equals("/")) {
                result = a / b;
        } else if (op.equals("*")) {
                result = a * b;
        } else if (op.equals("+")) {
                result = a + b;
        } else if (op.equals("-")) {
                result = a - b;
        }
        return result;
}

/**
```

```
 * Solves and returns the result of an expression involving a function
 * with only one operand.
 * Only the following function names are supported : <pre>{@code
 *      abs     -       absolute value
 *      fct     -       factorial
 *      exp     -       exponentiation
 *      log     -       logarithm (base 'e')
 *      rad     -       convert degrees to radians
 *      deg     -       convert radians to degrees
 *      sin     \
 *      cos     |
 *      tan      \      standard trigonometric
 *      sec      /      functions
 *      csc     |
 *      ctn     /}</pre>
 *
 *      @param  func                    the function name
 *      @param  x                       the operand
 *      @return                         the result on evaluating the expression
 *      @throws com.github.sahasatvik.math.FunctionNotFoundException
 *                                      thrown when func is not recognized
 *      @since  1.0
 */

protected static double solveUnaryFunction (String func, double x)
                                        throws FunctionNotFoundException {
        double result = 0.0;
        /*
         * Math the function name against supported ones, then
         * perform the appropriate operation.
         */
        if (func.equals("sin")) {
                result = Math.sin(x);
        } else if (func.equals("cos")) {
                result = Math.cos(x);
        } else if (func.equals("tan")) {
                result = Math.tan(x);
        } else if (func.equals("csc")) {
                result = 1.0/Math.sin(x);
        } else if (func.equals("sec")) {
                result = 1.0/Math.cos(x);
        } else if (func.equals("ctn")) {
                result = 1.0/Math.tan(x);
        } else if (func.equals("rad")) {
                result = Math.toRadians(x);
        } else if (func.equals("deg")) {
```

```
                result = Math.toDegrees(x);
            } else if (func.equals("fct")) {
                result = factorial(x);
            } else if (func.equals("abs")) {
                result = Math.abs(x);
            } else if (func.equals("exp")) {
                result = Math.exp(x);
            } else if (func.equals("log")) {
                result = Math.log(x);
            } else {
                /*
                 * Throw an Exception if the function name does not
                 * match any supported one.
                 */
                throw new FunctionNotFoundException(func + "[]");
            }
            return result;
        }
}
```

```java
// src/com/github/sahasatvik/math/MissingOperandException.java

package com.github.sahasatvik.math;

/**
 * Exception thrown when an expression passed to ExpressionParser has a missing
 * operand.
 *
 *      @author         Satvik Saha
 *      @version        1.0, 16/10/2016
 *      @see            com.github.sahasatvik.math.ExpressionParserException
 *      @since          1.0
 */

public class MissingOperandException extends ExpressionParserException {
        private String op;

        /**
         * Constructor of MissingOperandException.
         *
         *      @param  faultyExpression        the expression which could not be parsed
         *      @param  op                      the operator which is missing an operand
         *      @since  1.0
         */

        public MissingOperandException (String faultyExpression, String op) {
                super(faultyExpression);
                this.op = op;
        }

        /**
         * Gets the operator which is missing an operand.
         *
         *      @return                         the operator which is missing an operand
         *      @since  1.0
         */

        public String getOperator () {
                return op;
        }
}
```

```
// src/com/github/sahasatvik/math/NullExpressionException.java

package com.github.sahasatvik.math;

/**
 * Exception thrown when an expression passed to ExpressionParser is empty.
 *
 *      @author         Satvik Saha
 *      @version        1.0, 16/10/2016
 *      @see            com.github.sahasatvik.math.ExpressionParserException
 *      @since          1.0
 */

public class NullExpressionException extends ExpressionParserException {

        /**
         * Constructor of NullExpressionException.
         *
         *      @since  1.0
         */

        public NullExpressionException () {
                super("");
        }
}
```

```java
// src/com/github/sahasatvik/math/UnmatchedBracketsException.java

package com.github.sahasatvik.math;

/**
 * Exception thrown when an expression passed to ExpressionParser has unmatched
 * brackets.
 *
 *      @author         Satvik Saha
 *      @version        1.0, 16/10/2016
 *      @see            com.github.sahasatvik.math.ExpressionParserException
 *      @since          1.0
 */

public class UnmatchedBracketsException extends ExpressionParserException {
        private int pos;

        /**
         * Constructor of UnmatchedBracketsException.
         *
         *      @param  faultyExpression        the expression which could not be parsed
         *      @param  pos                     the index of the unmatched bracket
         *      @since  1.0
         */

        public UnmatchedBracketsException (String faultyExpression, int pos) {
                super(faultyExpression);
                this.pos = pos;
        }

        /**
         * Gets the index of the unmatched bracket.
         *
         *      @return                         the index of the unmatched bracket
         *      @since  1.0
         */

        public int getIndexOfBracket () {
                return pos;
        }
}
```

```java
// src/com/github/sahasatvik/math/VariableNotFoundException.java

package com.github.sahasatvik.math;

/**
 * Exception thrown when an unrecognized variable is passed to ExpressionParser.
 *
 *      @author         Satvik Saha
 *      @version        1.0, 16/10/2016
 *      @see            com.github.sahasatvik.math.ExpressionParserException
 *      @since          1.0
 */

public class VariableNotFoundException extends ExpressionParserException {
        private String var;

        /**
         * Constructor of VariableNotFoundException. This constructor accepts the
         * invalid expression as well as the unrecognized variable.
         *
         *      @param  faultyExpression        the expression which could not be parsed
         *      @param  var                     the unrecognized variable
         *      @since  1.0
         */

        public VariableNotFoundException (String faultyExpression, String var) {
                super(faultyExpression);
                this.var = var;
        }

        /**
         * Constructor of VariableNotFoundException. This Constructor accepts only
         * the unrecognized variable.
         *
         *      @param  var                     the unrecognized variable
         *      @since  1.0
         */

        public VariableNotFoundException (String var) {
                this("", var);
        }

        /**
         * Gets the unrecognized variable.
         *
```

```
         *      @return                         the unrecognized variable
         *      @since  1.0
         */

        public String getVar () {
                return var;
        }
}
```