Contents

1 Basic Test Results

```
compiling with
    javac -cp /cs/course/2013/oop/lib/junit4.jar  *.java
tests output :
    JUnit version 4.10
..............................
Time: 0.049

OK (32 tests)
```

2 README

```
roigreenberg


##################
File Description
##################

BinaryMathTerm.java - represents a term composed of exactly two independent
BracketsMathTerm.java - represents a math term between brackets
FractionMathTerm.java - epresents a special case of binary math term
MathTerm.java - This class represents the base class for all other MathTerms classes
SimpleBinaryOpMathTerm.java - represents a simple operation between two other terms
SimpleMathTerm.java - represents a math term which is either a single letter variable or a number
SumMathTerm.java - represents a mathematical sum
README - this file

########
Design
########

I done as the instraction says.
Since exponent, barred and negated can use for any sub-class of mathTerm I chose
to implement it in the mathTerm class itself.


#######################
Implementation Issues
#######################

None


######################
answer to question
######################
if I want to do integral, I will do it same as the SUM term.
a class that extend mathTerm and will have 4 parameters. the lower limit, the upper limit,
 the integrad and the difrantial
```

3 BinaryMathTerm.java

```java
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

/**
 *This class represents a term composed of exactly two independent
 * math terms
 * @author
 */

public class BinaryMathTerm extends MathTerm {

    protected MathTerm firstTerm;
    protected MathTerm secondTerm;
    /**
     * Constructs an new BinaryMathTerm
     * @param firstTerm - the first math term
     * @param seconfTerm - the second math term
     */
    public BinaryMathTerm(MathTerm firstTerm, MathTerm secondTerm){

        this.firstTerm = firstTerm;
        this.secondTerm = secondTerm;

    }
    /**
     * Unimplemented in this class. However, should be implemented in
     * any of its subclasses.
     */
    public java.lang.String toLatex(){

        return -2.5/-2.5 (code='general_error') in the future use abstract method
    }
}
```

4 BracketsMathTerm.java

```
1  /*
2   * To change this license header, choose License Headers in Project Properties.
3   * To change this template file, choose Tools | Templates
4   * and open the template in the editor.
5   */
6
7  /**
8   *This class represents a math term between brackets
9   * @author
10  */
11 public class BracketsMathTerm extends MathTerm{
12
13     MathTerm internal
14     /**
15      * The constructo                                rm they will be rendered
16      * as the term in
17      * @param interna                                 resides within the brackets
18      */
19     public BracketsMathTerm(MathTerm internalTerm){
20
21         this.internalTerm = internalTerm;
22     }
23     /**
24      * Generates the latex representation of for this bracket math term
25      * @overrides toLatex in class BinaryMathTerm
26      * @return string that represent a term in brackets in latex
27      */
28     public java.lang.String toLatex(){
29
30
31         return this.setTerm("\\left( " + internalTerm.toLatex() +
32                 " \\right)");
33     }
34 }
```

-0.83333333333333/-5 You shouldn't allow public access to class fields. (code='public_members') why not private

5 FractionMathTerm.java

```java
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

/**
 * This class represents a special case of binary math term.
 * It should be rendered as a fraction
 * @param firstTerm
 * @param secondTerm
 * @author
 */
public class FractionMathTerm extends BinaryMathTerm    {
    /**
     * Constructs a new Fraction term
     * @param firstTerm - Term on the numerator ("Mone")
     * @param secondTerm - Term on the denominator ("Mechane")
     */
    public FractionMathTerm(MathTerm firstTerm, MathTerm secondTerm){

        super(firstTerm,secondTerm);

    }
    /**
     * Generates the latex representation of this fraction math term.
     * @overrides toLatex in class BinaryMathTerm
     * @returns latex representation of this fraction math term using
     *             the \frac latex command
     */
    public java.lang.String toLatex(){

            return this.setTerm("\\frac{ " + this.firstTerm.toLatex() +
                    " }{ " + this.secondTerm.toLatex() + " }");
    }
}
```

6 MathTerm.java

```java
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

/**
 * This class represents the base class for all other MathTerms
 * classes you will implement in the exercise. It defines the interface
 * (the public part of the classes) for all other math terms which will
 * extend it. Most importantly, it defines the method "toLatex" that
 * should be overridden in each of the extending classes. This class
 * is not meant to be instantiated at any point, the only classes that
 * will be instantiated are classes that extend it. Thus, the
 * "toLatex" method should be left unimplemented (return "").
 * It will be implemented, however, in any of this class' subclasses.
 * The rest of the interface (setExponentTerm, setters and getters, etc.)
 * or any additions you may want to add to this class (as long as they
 * are hidden do not modify its interface), can and should be implemented
 * in this class. Later on this course, we will find a more elegant way
 * to "enforce" subclasses to implement a method rather than leaving
 * unimplemented methods (abstract, interface). Note: The interface
 * allows to "annotate" the term with bar (upper line), negation or with
 * other MathTerm as exponent. When more than one of these annotations
 * are set, please evaluate the latex representation in the following
 * order: exponent, bar and then negation.
 * @author roigreenberg
 */
public class MathTerm extends java.lang.Object{


    protected boolean isBarred;
    protected boolean isNegated;
    protected MathTerm exponentTerm;
    /**
     * Default constructor
     */
    public MathTerm(){


    }
    /**
     * This method gets a math term to be placed as an exponent for the
     * current math term. For example. If our current MathTerm is "a"
     * and the user passes "b". Then our Mathterm will be rendered as
     * "a^{ b }".
     * @param exponentTerm - The MathTerm to be placed as an exponent
     *          of the current term.
     */
    public void setExponentTerm(MathTerm exponentTerm){

         this.exponentTerm = exponentTerm;
    }
    /**
     * Returns the exponent math term.
     * @return The exponent MathTerm of this term.
     */
    public MathTerm getExponentTerm(){

```

```
60          return exponentTerm;
61      }
62      /**
63       * Setting whether this MathTerm should be barred or not (a straight
64       * line on top of the term: see Latex's \overline{}).
65       * @param isBarred - true if we want this term to be barred.
66       */
67      public void setIsBarred(boolean isBarred){
68
69          this.isBarred = isBarred;
70      }
71      /**
72       * isBarred getter
73       * @returns whether this math term was set to be barred.
74       */
75      public boolean getIsBarred(){
76
77          return isBarred;
78      }
79      /**
80       * Sets whether this math term should be negated (see Latex's \neg{}).
81       * @param isNegated -  true if we want this term to be nagated
82       */
83      public void setIsNegated(boolean isNegated){
84
85          this.isNegated = isNegated;
86      }
87      /**
88       * isNegated getter.
89       * @param True if this term should be negated.
90       */
91      public boolean getIsNegated(){
92
93          return isNegated;
94      }
95      /**
96       * This method should be implemented in any of MathTerm derivatives
97       */
98      public java.lang.String toLatex(){
99
100         return "";
101     }
102     /**
103      * set the term w                                    and/or negated
104      * @return new st                                   ntion in latex of
105      *     exponent, barred or negated.
106      */
107     protected java.lang.String setTerm(java.lang.String term){
108
109         if (this.getExponentTerm() != null){
110             term = term + "^{ " + this.getExponentTerm().toLatex() + " }";
111         }
112         if (this.getIsBarred()){
113             term ="\\overline{ " + term + " }";
114         }
115         if (this.getIsNegated()){
116             term ="\\neg{ " + term + " }";
117         }
118         return term;
119     }
120 }
```

-2.5/-2.5 (code='general_error') in the future use abstract method

-2.5/-2.5 You used too many strings and/or numbers in your code. (code='magic_numbers')

7 SimpleBinaryOpMathTerm.java

```java
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

/**
 * This class represents a simple operation between two other terms.
 * For example, "a+b", "c*d" or "c=d"
 * @author
 */
public class SimpleBinaryOpMathTerm extends BinaryMathTerm {

    private char sign;
    /**
     * Instantiate a new SimpleBinaryOpMathTerm
     * @param firstTerm - The first term of the binary operation.
     * @param secondTerm - The second term of the binary operation.
     * @param sign - The operation sign.
     *           Can be any of the following: "+,-,*,<,>,=".
     */
    public SimpleBinaryOpMathTerm(MathTerm firstTerm, MathTerm secondTerm,
            char sign){

        super(firstTerm,secondTerm);

        this.sign = sign;
    }
    /**
     * Generates the latex representation of this arithmetic operation
     * math term
     * @overrides     toLatex in class BinaryMathTerm
     * @returns The latex representation of the operation:
     *             "firstTerm operationSign secondTerm".
     */
    public java.lang.String toLatex(){

        if (sign == '*') {
            return this.setTerm(this.firstTerm.toLatex()+ " \\cdot "
                    + this.secondTerm.toLatex());
        }
        else {
            return this.setTerm(this.firstTerm.toLatex() + this.sign
                    + this.secondTerm.toLatex());
        }
    }
}
```

8 SimpleMathTerm.java

```java
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

/**
 * This class represents a math term which is either a single letter
 * variable (x,y,a,b,etc..) or a number (may be a floating point number).
 * The latex representation is straight forward the name of the variable,
 * or the number itself. However, in case this term represents a number,
 * the class will allow to user to control the precision of its latex
 * representation, that is - the number of digits to the right of the
 * floating dot.
 * @author
 */

public class SimpleMathTerm extends MathTerm {

    protected java.lang.String termName;
    private int precisionDigits;
    protected java.lang.String term ;
    /**
     * Constructs a new instance given a simple term "name"
     * @param termName - A string of either a single letter variable
     *  (x,y,z,a,b..) or a number (may be a floating point number).
     */
    public SimpleMathTerm(java.lang.String termName){

        this.termName = termName;
        // another variable so the original termName wont be change
        this.term = termName;
    }
    /**
     * Sets the number of digits of precision in case this term
     * represents a number.
     * change term according to the precisionDigits.
     * @param precisionDigits - Number of digits right of the
     *             floating point on the latex representation.
     */
    public void setPrecisionDigits(int precisionDigits){

        this.precisionDigits = precisionDigits;
        double number = Double.parseDouble(this.termName);
        double precisNum = (int) (number*(Math.pow(10,this.precisionDigits)))
                /((double) (Math.pow(10,this.precisionDigits)));
        this.term =  Double.toString(precisNum);
        if (precisionDigits == 0) {
            int naturalNum = (int) precisNum;
            this.term =  Integer.toString(naturalNum);
        }

    }
    /**
     * Checks the given name. And determines whether it's numeric.
     *
     * @Returns true if this term represents a number.
     */
    public boolean isNumeric() {
```

```
60
61          if (this.termName.charAt(0)=='-') {
62              return Character.isDigit(this.termName.charAt(1));
63          }
64
65          return Character.isDigit(this.termName.charAt(0));
66
67      }
68      /**
69       * Generates the latex representation of the this simple math term.
70       * @overrides     toLatex in class MathTerm
71       * @return the Latex representation. If this term represents a
72       * variable, this method returns the variable name. Otherwise,
73       * if the term represents a number it should be trimmed to
74       * according to the precision parameter.
75       * also change the term is exponent, barred or negated in needed
76       */
77      public java.lang.String toLatex(){
78
79          return this.setTerm(this.term);
80      }
81  }
```

9 SumMathTerm.java

```java
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

/**
 * This class represents a mathematical sum. It comprised of 3 MathTerms:
 *  - Term beneath the sigma sign.
 *  - Term above the sigma sign.
 *  - Term being summed.
 *  Use the \sum latex command, to generate the latex representation of
 *  this MathTerm.
 * @author
 */
public class SumMathTerm extends MathTerm{

    private MathTerm subTerm;
    private MathTerm superTerm;
    private MathTerm sumTerm;
    /**
     * The constructor receives the 3 MathTerm that comprises the sum
     * math term.
     * @param subTerm - The term beneath the sigma.
     * @param superTerm - The term above the sigma.
     * @param sumTerm - The summed term.
     */
    public SumMathTerm(MathTerm subTerm, MathTerm superTerm,
            MathTerm sumTerm){

        this.subTerm = subTerm;
        this.superTerm = superTerm;
        this.sumTerm = sumTerm;
    }
    /**
     * Generates the latex representation of the this simple math term.
     * @overrides     toLatex in class MathTerm
     * @return latex representation using the \sum command.
     */
    public java.lang.String toLatex(){

        return this.setTerm("\\sum_{ " + this.subTerm.toLatex() + " }^{ "
        + this.superTerm.toLatex() + " }{ " + this.sumTerm.toLatex() + " }");
    }
}
```