```
//there are 4 separate classes in this source code.
PowerRuleSolverMain.java
//PowerRuleFrame.java PowerRule.java and FunctionProperties.java

import javax.swing.JFrame;
/**
        entry point for PowerRuleSolver application
*/
public class PowerRuleSolverMain{
        public static void main(String[] args) {
                JFrame frame = new PowerRuleFrame();
        }
}


import javax.swing.JFrame;
import javax.swing.JButton;
import javax.swing.JPanel;
import javax.swing.JLabel;
import javax.swing.JTextField;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JSplitPane;
import java.awt.Dimension;
import java.awt.BorderLayout;
import javax.swing.JTextArea;
import java.util.ArrayList;
import javax.swing.JScrollPane;

/**
        controls the Jframe for the GUI for PowerRuleSolver
*/

public class PowerRuleFrame extends JFrame{

        private JButton execute;
        private JButton ch1;

        private JLabel title;
        private JLabel instruction;
        private JTextArea explanation;
        private JTextArea details;
        private final String DETAILS_STRING = "Explanation of how the
answer was found:\n";
        private String newDetailsString = "";
        private JTextField input;
        private JScrollPane sp;

        private JPanel titlePanel;
        private JPanel powerRule;
```

```
        private JPanel outputPanel;
        private TextPanel detailPanel;

        private JTextField output;
        private JSplitPane tb;

        private BorderLayout bl;
        private BorderLayout bl1;

        private ArrayList<FunctionProperties> evaluatedFunction = new
ArrayList<FunctionProperties>();

        private static final int FRAME_WIDTH = 600;
        private static final int FRAME_HEIGHT = 500;

        /**
        initializes all of the possible components to display and sets
values for the frame
        */
        private void placeComponents(){

                createButtons();
                createTextField();
                creatLabel();

                powerRule = new JPanel();
                titlePanel = new JPanel();
                outputPanel = new JPanel();

                tb = new JSplitPane(JSplitPane.VERTICAL_SPLIT,
titlePanel, powerRule);
                tb.setDividerSize(0);

                bl = new BorderLayout(10, 10);
                bl1 = new BorderLayout(10, 10);

                powerRule.setLayout(bl);
                outputPanel.setLayout(bl1);

                powerRule.add(explanation, BorderLayout.PAGE_START);
                powerRule.add(instruction, BorderLayout.LINE_START);
                powerRule.add(input, BorderLayout.CENTER);
                powerRule.add(execute, BorderLayout.LINE_END);
                powerRule.add(outputPanel, BorderLayout.PAGE_END);
                outputPanel.add(output, BorderLayout.PAGE_START);
                detailPanel = new TextPanel();
                detailPanel.setPreferredSize(new Dimension(590,
250));
                outputPanel.add(detailPanel, BorderLayout.CENTER);
```

```
                titlePanel.add(title);
                add(tb);
                setVisible(true);
                setTitle("PowerRuleSolver");
                setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                setResizable(false);
                setSize(FRAME_WIDTH, FRAME_HEIGHT);

        }

        /**
                initializes the init of the components
        */
        public PowerRuleFrame(){
                placeComponents();
        }

        /**
                initializes all of the buttons
        */
        private void createButtons(){
                execute = new JButton("calculate");
                ActionListener listener = new EquationListener();
                execute.addActionListener(listener);
        }

        /**
                initializes the JTextFields and the JtextArea for the
definiton of power rule for PowerRuleSolver
        */
        private void createTextField(){
                input = new JTextField();
                ActionListener listener = new EquationListener();
                input.setText("");
                input.addActionListener(listener);
                input.setPreferredSize(new Dimension(100, 10));
                output = new JTextField();
                output.setPreferredSize(new Dimension(10, 50));
                explanation = new JTextArea("In calculus, a
derivative is the slope of the function at any given point usually
denoted as\n X. One possible way of solving a derivative is known as
the Power Rule. In the power rule, you\n multiply the coefficient by
the exponent and then subtract 1 from the exponent. For example,\n
2x^5 becomes 10x^4.");
                explanation.setPreferredSize(new Dimension(10, 80));
        }

        /**
                initializes the Jlabels for calc buddy
        */
```

```
        private void creatLabel(){
                title = new JLabel("Power Rule Solver");
                instruction = new JLabel("Please type in an eqution
to derive");

        }

        /**
                listener for the equation text feild
        */
        class EquationListener implements ActionListener{
                public void actionPerformed(ActionEvent e){
                        String equation = input.getText();
                        PowerRule a = new PowerRule(equation);
                        try{
                                int currentStart = 0;
                                int currentEnd =
newDetailsString.length()==0? 0:newDetailsString.length()-1;
                                newDetailsString = DETAILS_STRING;

System.out.println("newDetailsString: " + newDetailsString);
                                output.setText(a.calcAnswer());
                                evaluatedFunction = new
ArrayList<FunctionProperties>(a.getFunctionPorperties());
                                for(int i = evaluatedFunction.size()
- 1; i >= 0; i--){

System.out.println(evaluatedFunction.get(i).getFunction());
                                        newDetailsString += "\n" +
"isolate the function: " + evaluatedFunction.get(i).getFunction() +
"\n" + "Identify the coefficient: " +
evaluatedFunction.get(i).getCoef() + "\n" + "identify the exponent: "
+ evaluatedFunction.get(i).getExpo() + "\n" + "finally, solve: " +
evaluatedFunction.get(i).getDerivative() + "\n";
                                }

detailPanel.addText(newDetailsString, currentStart, currentEnd);
                                //outputPanel.repaint();
                                //outputPanel.revalidate();
                        } catch(Exception ex){
                                output.setText("an error has
occured");
                        }

                }
        }

        /**
                class that handles the changing details JTextArea
        */
```

```
        public class TextPanel extends JPanel {
            private JTextArea details;

            /**
                        constructor for TextPanel
            */
            public TextPanel() {
                newDetailsString = DETAILS_STRING;
                        details = new JTextArea(newDetailsString);
                setLayout(new BorderLayout());
                add(new JScrollPane(details), BorderLayout.CENTER);
            }

            /**
                        replaces the old text on the details text are
with new text
                        @param text the test to be placed in the text
area
                        @param start starting position of old text in
text area
                        @param end ending position of old text in
text area
            */
            public void addText(String text, int start, int end) {
                details.replaceRange(text, start, end);
        }
        }

}


/**
    class that handles functions that require power rule to solve the
derivative for
*/
import java.util.regex.Pattern;
import java.util.regex.Matcher;
import java.util.Stack;
import javax.script.ScriptEngineManager;
import javax.script.ScriptEngine;
import java.util.ArrayList;
import java.text.DecimalFormat;


public class PowerRule{
         private String inputString;
    private String processedString;
    private String answer = "";
    private String coef;
    private String expo;
```

```
    private int intCoefVal;
    private int intExpoVal;
    private double dubCoefVal;
    private double dubExpoVal;
    private int variableSubstringValue;
    private final static DecimalFormat df2 = new
DecimalFormat(".###");

    private boolean isNumberFound = false;
    private boolean varFound = false;
    private boolean carrotFound = false;

    private Stack<String> functions = new Stack<String>();
    private Stack<String> operators = new Stack<String>();
    private String regExPattern = "((((\\-\\(|\\()?(\\-?\\d+\\.\\d+|\
\-?\\d+)\\)?((\\/|\\*|\\+|\\-)(\\-?\\d+\\.\\d+|\\-?\\d+)\\)?)?)?(([a-
z]|[A-Z])(\\^(\\-\\(|\\()?(\\-?\\d+\\.\\d+|\\-?\\d+)\\)?((\\/|\\*|\\+|
\\-)(\\-?\\d+\\.\\d+|\\-?\\d+)\\)?)?)?))";
    private Pattern powerPattern;
    private Matcher powerMatcher;

    private ScriptEngineManager evaluatorManager;
    private ScriptEngine evaluator;

    private ArrayList<FunctionProperties> evaluatedFunction = new
ArrayList<FunctionProperties>();

    /**
                Constructor
    */
    public PowerRule(){

    }

    /**
                Constructor
                @param input the input string that will be calculated
    */
    public PowerRule(String input){
        inputString = input;
    }

    /**
                gets inputString (mainly for the inherited classes)
                @param input the string to become inputString
    */
    public void getString(String input){
        inputString = input;
    }
```

```
    /**
                Gets the answer string for output porposes
                @return the answer
    */
    public String getAnswer(){
        return answer;
    }

    /**
                gets the inputString for output purposes and
calculation purposes in classes that inherit this class
                @return the inputString
    */
    public String getInputString(){
        return inputString;
    }

    /**
                changes answer to the input String
                @param input the String to change answer to
    */
    public void setAnswer(String input){
        answer = input;
    }

    /**
                changes the inputString
                @param input the string to change inputString to
    */
    public void setInputString(String input){
        inputString = input;
    }

    /**
        returns the ArrayList of FunctionProperties
        @return the arraylist of FunctionProperties
    */
    public ArrayList<FunctionProperties> getFunctionPorperties(){
        return evaluatedFunction;
    }

    /**
                algorithm for adding coefficients to the variables of
a function that dont already have a coefficient. Simplifies later
processing
    */
    private void addCoefficient(){
         //scans through inputString
        for(int stringIterator = 0; stringIterator <
inputString.length(); stringIterator++){
```

```
                //checks if the current char is a letter

               if(Character.toString(inputString.charAt(stringIterator
)).matches("[a-z]|[A-Z]")){

                    variableSubstringValue = stringIterator;

                    //is the variable at position 0? else add the 1
elsewhere
                    if(stringIterator == 0){
                        inputString = "1" + inputString;
                        stringIterator++;
                    }else{

                    //checks to see if the variable already has a
number infront of it

                   if(Character.toString(inputString.charAt(variableSu
bstringValue - 1)).matches("\\d|\\)")){
                        isNumberFound = true;
                    }

                    //adds a one in front of a variable if a number is
not already found. otherwise just increments i and moves on
                    if(!isNumberFound){
                        inputString = inputString.substring(0,
variableSubstringValue) + "1" +
inputString.substring(variableSubstringValue, inputString.length());
                        stringIterator++;
                    }else{
                        stringIterator++;
                    }

                    //reset isNumberFound for next iteration
                    isNumberFound = false;
                }
            }
        }
        processedString = inputString;
    }

    /**
        parses the string input from the user
    */
    private void regexForTheWin(){

        powerPattern = Pattern.compile(regExPattern);
        powerMatcher = powerPattern.matcher(processedString);
```

```
        //puts the functions in the input string into the functions
stack
        while(powerMatcher.find()){
            functions.push(powerMatcher.group(0));
            System.out.println("Stack input: " +
powerMatcher.group(0));
        }

        //takes the functions out of the processedString string for
later processing
        for(String str : functions){
            processedString =
processedString.replaceFirst(regExPattern, "");
        }

        //checks to see if something went wrong and puts all the
operators in their own stack for putting the whole string back
together
        for(char c : processedString.toCharArray()){
            System.out.println(c);
            if(Character.toString(c).matches("\\+|\\-")){
                operators.push(Character.toString(c));
            }

            if(!Character.toString(c).matches("\\+|\\-|\\/|\\*|\\(|\
\)|\\s")){
                answer = "an error has occured";
            } else if(Character.toString(c).matches("\\/|\\*")){
                answer = "you need quotient or product rule to solve
this";
            } else if(Character.toString(c).matches("\\(|\\)")){
                answer = "error: unmatched parenthesis";
            }
        }

    }

    /**
        finds the coefficient of a function
        @param str the function to searched for the coefficient
        @return the coefficient as a String
    */
    private String findCoefficient(String str){

        varFound = false;
        String coefString = "";

        for(int i = 0; !varFound && i < str.length(); i++){
                varFound = str.substring(i, i+1).matches("[a-z]|[A-
Z]")? true:false;
```

```
                coefString = str.substring(0, i);
            }
            return coefString;
        }

        /**
            finds the ecponent of a function
            @param str the function that will be searched for the exponent
            @return the exponent as a String
        */
        private String findExponent(String str){

            carrotFound = false;
            String expoString = "";

            for(int i = 0; i < str.length() && !carrotFound; i++){
                carrotFound = str.substring(i, i+1).equals("^")? true :
false;
                expoString = str.substring(i+1);
            }
            return expoString;
        }

        /**
            algorithm for processing the coefficient and exponent
        */
        private void process() throws Exception{
            evaluatorManager = new ScriptEngineManager();
            evaluator = evaluatorManager.getEngineByName("JavaScript");

            if(answer.equals("an error has occured") || answer.equals("you
need quotient or product rule to solve this") || answer.equals("error:
unmatched parenthesis")){
               //do nothing
            } else {
                while(!functions.empty()){
                    evaluatedFunction.add(new FunctionProperties());
                    String str1 = functions.pop();


                   evaluatedFunction.get(evaluatedFunction.size()-1).setFu
nction(str1);

                    coef = findCoefficient(str1);
                    expo = findExponent(str1);


                   evaluatedFunction.get(evaluatedFunction.size()-1).setCo
ef(coef);
```

```
                evaluatedFunction.get(evaluatedFunction.size()-1).setEx
po(expo);

                str1 = str1.replaceFirst(Pattern.quote(coef), "");
                str1 = str1.replaceFirst(Pattern.quote(expo), "");

                coef = evaluator.eval(coef).toString();
                expo = evaluator.eval(expo).toString();

                dubCoefVal = Double.parseDouble(coef);
                dubExpoVal = Double.parseDouble(expo);

                dubCoefVal = dubCoefVal * dubExpoVal;
                dubExpoVal = dubExpoVal - 1;

                if(dubCoefVal % 1 == 0){
                    intCoefVal = (int) dubCoefVal;
                    coef = Integer.toString(intCoefVal);
                } else{
                    coef = df2.format(dubCoefVal); //
Double.toString(dubCoefVal)
                }

                if(dubExpoVal % 1 == 0){
                    intExpoVal = (int) dubExpoVal;
                    expo = Integer.toString(intExpoVal);
                } else{
                    expo = df2.format(dubExpoVal); //
Double.toString(dubExpoVal)
                }

                if(expo.equals("0")){
                    str1 = coef;

                   evaluatedFunction.get(evaluatedFunction.size()-1).s
etDerivative(str1);
                    answer = str1 + answer;
                } else{
                    str1 = coef + str1 + expo;

                   evaluatedFunction.get(evaluatedFunction.size()-1).s
etDerivative(str1);
                    answer = str1 + answer;
                }

                if(!operators.empty()){
                    answer = " " + operators.pop() + " " + answer;
                }

            }
```

```
        }
    }

    /**
                calculates the answer
                @return the answer
    */
    public String calcAnswer() throws Exception{
        this.addCoefficient();
        this.regexForTheWin();
        this.process();
        return answer;
    }

}


/**
        data type for the output of the details on how the equation
was solved
*/
public class FunctionProperties{

        private String function;
        private String coef;
        private String expo;
        private String derivative;

        /**
                constructor
        */
        public FunctionProperties(){

        }

        /**
                constructor
                @param function the function
                @param coef the coefficient of the function
                @param expo the exponent of the function
                @param derivative the calculated derivative of the
function
        */
        public FunctionProperties(String function, String coef, String
expo, String derivative){
                this.function = function;
                this.coef = coef;
                this.expo = expo;
                this.derivative = derivative;
        }
```

```
/**
        returns the function
        @return the function
*/
public String getFunction(){
        return function;
}

/**
        returns the coefficient
        @return the coefficient
*/
public String getCoef(){
        return coef;
}

/**
        returns the exponent
        @return the exponent
*/
public String getExpo(){
        return expo;
}

/**
        returns the calculated derivative
        @return the calculated derivative
*/
public String getDerivative(){
        return derivative;
}

/**
        passes in and sets the String of the function
        @param function the String of the function
*/
public void setFunction(String function){
        this.function = function;
}

/**
        passes in and sets the String of the coefficient
        @param coef the String of the coefficient
*/
public void setCoef(String coef){
        this.coef = coef;
}

/**
```

```
                passes in and sets the String of the exponent
                @param the String of the exponent
        */
        public void setExpo(String expo){
                this.expo = expo;
        }

        /**
                passes in and sets the String of the Derivative
                @param the String of the derivative
        */
        public void setDerivative(String derivative){
                this.derivative = derivative;
        }
}
```