Part C

Formula Class – OOP – Encapsulation – Abstract Datatype

The formula class is an abstract datatype that uses the concept of encapsulation holding the information required for each formula.

The formula class also has a compute function which computes the user inputted values for each variable. The function iterates for value of the degree variable which designates the number of variables used in the formulas. The function performs the designated operation stored in the char array operation for each step to calculate the final formula and returns the answer to the main.

```
private String name ="";
private int degree = 0;
private String[] variables;
private char[]operations;
private String formula="";

public float Compute(float[]vars){
    float result = vars[0];
    for(int i=1; i <degree; i++){
        if (operations[i-1] == '+'){
            result = result + vars[i];
        }
        else if (operations[i-1] == '-'){
            result = result - vars[i];
        }
        else if (operations[i-1] == '*'){
            result = result * vars[i];
        }
        else if (operations[i-1] == '/'){
            result = result / vars[i];
        }
    }

    return result;
```

Different type of data each formula instance holds

Function to calculate answer given values for each variable

Iterates for the number of operations that formula has

If statements to identify the correct operation to perform on the next inputted value

Formula Database Class – Searching – Sorting – List Datatype – Composite Data Structures

The formula database holds a list of all the formulas. It has two functions a bubble sort and a binary search. The binary search works by starting at the middle of the list, if the middle is further in the alphabet than the desired name of the function the function will binary search the left and else if the value was larger it would search the right.

The database is also an example of a composite data structure as it holds a list of all the formulas which are stored in the form of the abstract data type Formula.

```
    while (searching){
        int location = ((high+low)/2);
        if ((this.formulas[location].getName().compareToIgnoreCase(search) == 0)){
            result = formulas[location];
            searching = false;
        }
        else if ((this.formulas[location].getName().compareToIgnoreCase(search) < 0)){
            low = location;
        }
        else if ((this.formulas[location].getName().compareToIgnoreCase(search) > 0)){
            high = location;
        }
        else if (high == formulas.length -1 && low == formulas.length -2){
            low = high;
        }
        else if(low == high-1 && this.formulas[location].getName().compareToIgnoreCase(search) != 0 ){
            searching = false;
            result = null;
        }
    }
    return result;
}
```

Resets the boundaries for the next binary search

Case 1: Formula name = keyword

Case 2: Formula name is after the middle term in the alphabet

Case 2: Formula name is before the middle term in the alphabet

Case 4: Keyword doesn’t exist in the formula list

The bubble sort iterates through the list till the number of switches after its current iteration equals 0. It compares the elements that it's currently indexed at to the one after the current element. If the current element is greater than the next element the function will swap the two elements.

```java
public void bubbleSort(){
    Formula temp;
    boolean sorted = true;

    //Keeps sorting while the database is unsorted
    while (sorted){

        sorted = false;
        // Bubbles over students that are out of place
        for (int i = 0; i < this.formulas.length-1;i++){
                    if (this.formulas[i].getName().compareToIgnoreCase(this.formulas[i+1].getName()) > 0){
            temp = this.formulas[i];
            this.formulas[i] = this.formulas[i+1];
            this.formulas[i+1] = temp;
            //Keeps sorting as a bubble has occurred
            sorted = true;
          }
        }
    }

}
```

Keeps sorting until list is sorted

Check if current term is greater than the next term

Keeps running main loop till no bubble takes place

Parsing/Grammar Dictionary – Complex Parsing

When the code is first inputted and split by "+" and "=". The programs parser function takes one term at a time and starts the parsing process using regexes to search for the factor at the end of a chemical compound for example Al(NO3)3 has ")3" which is what the n variable searches for.

The second splits the term into the elements that it contains looking for a capital letter possibly followed by a lowercase letter and a number. Using this self created grammar dictionary, I was able to parse each compound into its separate elements that it contains.

The k pattern further splits the Elements into the letter of the element. The l pattern searches for the numbers following the letter. If a factor existed after the closing bracket that factor was multiplied by the number. Finally, the letter and number of occurrences of its represented element are added to an element list.

```java
public static ArrayList<String> parser(String eqn) {
    Matcher m = null;
    Matcher j = null;
    Matcher k = null;
    Matcher l = null;
    Matcher n = null;
    ArrayList<String> terms = new ArrayList<String>();

    ArrayList<String> sorted = new ArrayList<String>();
    ArrayList<String> number = new ArrayList<String>();
    if (eqn.contains("(")) {
        m = Pattern.compile("[(]*[A-Z]+[a-z]*[1-9]*[)]*[1-9]*").matcher(eqn);

    } else {
        m = Pattern.compile("[(]*[A-Z][a-z]*[1-9]*[)]*[1-9]*").matcher(eqn);
    }
    while (m.find()) {

        terms.add(m.group());
    }
    for (int i = 0; i < terms.size(); i++) {
        ArrayList<String> elements = new ArrayList<String>();
        int factor = 1;
        ArrayList<String> compounds = new ArrayList<String>();
        if (terms.get(i).contains("(")) {
            n = Pattern.compile("\\)\\d").matcher(terms.get(i));
            j = Pattern.compile("[A-Z][a-z]*[1-9]*").matcher(terms.get(i));
        while (n.find()) {

            factor = Integer.parseInt(n.group().substring(1));

        }

        while (j.find()) {

            compounds.add(j.group());
        }
        for (int h = 0; h < compounds.size(); h++) {
            int num = 1;
            String elementSt = "";
            k = Pattern.compile("[A-Z]+[a-z]*").matcher(compounds.get(h));
            while (k.find()) {

                elementSt = (k.group());
            }
            l = Pattern.compile("[1-9]").matcher(compounds.get(h));
```

Instantiation for all the regexs

Iterates for each identifiable compound creating a list

Looks for the factor to multiply all elements enclosed by brackets

Identifies each individual element in the compound

Breaks down the element into the element letter and its number

```java
            while (l.find()) {
                num = Integer.parseInt(l.group());

            }

            elements.add(elementSt + num * factor);
        }

    } else {
        j = Pattern.compile("[A-Z][a-z]*[1-9]*").matcher(terms.get(i));

        while (j.find()) {

            compounds.add(j.group());
        }
        for (int h = 0; h < compounds.size(); h++) {
            int num = 1;
            String elementSt = "";
            k = Pattern.compile("[A-Z]+[a-z]*").matcher(compounds.get(h));
            while (k.find()) {

                elementSt = (k.group());
            }
            l = Pattern.compile("[1-9]").matcher(compounds.get(h));
            while (l.find()) {
                num = Integer.parseInt(l.group());
                }

                elements.add(elementSt + num * factor);
            }

        }
        for (int o = 0; o < elements.size(); o++) {
            sorted.add(elements.get(o));
        }

    }
    return sorted;

}
```

Similar process without the factor multiplication to the occurrence of each element enclosed by brackets

Parses the number after element letter to an int

Multiplies the int by the factor and saves the final occurrence of each element in the element list

Matrix Creation – (2D Arrays)

The chemical equation balancer balances an unbalanced chemical equation by creating two matrices the first with the numbers referring to the occurrences of each element in each term in the equation. The second has numbers for the final term in the equation. The code below creates the matrix through the use of nested loops. The first loop iterates for each element, the second loop iterates for each term in the equation, If the current element exists in the term the number corresponding to its occurrences is added to the matrix else a 0 is used.

```
for (int i = 0; i < (right.size() + left.size() - 1); i++) {

    for (int j = 0; j < elements.size(); j++) {
        double digi = 0;
        for (int k = 0; k < parsed.get(i).size(); k++) {
            if (parsed.get(i).get(k).contains(elements.get(j))) {
                String digit = "0";

                Matcher m = Pattern.compile("[1-9]").matcher((parsed.get(i).get(k)));
                while (m.find()) {
                    digit += m.group();
                }
                if (digit.equals("0")) {
                    digit = "1";
                }
                digi += Double.parseDouble(digit);

            }

        }
        if (i >= reactants.length && digi != 0) {
            digi = digi * -1;
        }
        lhsA[j][i] = digi;
    }
}
```

Iterates for number of total elements

Iterates for number of compounds

Checks whether current Element occurs in the current compounds

Identifies the occurrence of that element in the compound

Checks if current compound is a reactant and makes the number negative if the it is

Adds digit to matrix

Error Handling

Through the use of try and catch statements I give the user instructions if they inputted an unacceptable type of input.

```
        boolean finished = true;
        try{

        float[] variables = new float[dformula.getDegree()];
        boolean calculate = true;
        for (int i = 0; i < dformula.getDegree(); i++) {
                variables[i] = Float.parseFloat(inputs[i].getText());

        }

        float answer = dformula.Compute(variables);
        String ans = String.valueOf(answer);
        result.setText(ans);
        finished = false;
        }finally{
            if (finished){
            JOptionPane.showMessageDialog(frmMenu, "Make sure to enter proper variable values");
            }
        }
    }
```

GUI Creation – External Library

Each window of the GUI (main, formula list, constant list, balancer, calculator) had their own respective menus. Each window has its own initialization process, the code below is the initialization for the calculation menu.

```
private void calcMenu(Formula formula) {

    frmMenu.setVisible(true);
    frmMenu.setTitle(formula.getName());
    frmMenu.setBounds(550, 100, 761, 700);
    frmMenu.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frmMenu.getContentPane().setLayout(null);
    frmMenu.setTitle("Science Buddy");
    JTextField txtfrmMenu = new JTextField();
    txtfrmMenu.setEditable(false);
    txtfrmMenu.setHorizontalAlignment(SwingConstants.CENTER);
    txtfrmMenu.setFont(new Font("Tahoma", Font.PLAIN, 48));
    txtfrmMenu.setText(formula.getName() + " " + formula.getFormula());
    txtfrmMenu.setBounds(15, 16, 710, 80);
    frmMenu.getContentPane().add(txtfrmMenu);
    txtfrmMenu.setColumns(10);

    inputs = new JTextField[formula.getDegree()];

    for (int i = 0; i < formula.getDegree(); i++) {

        JTextField option = new JTextField();
        option.setColumns(10);
        option.setBounds(380, 113 + 65 * i, 223, 49);
        option.setFont(new Font("Tahoma", Font.PLAIN, 30));
        option.setHorizontalAlignment(SwingConstants.CENTER);
        frmMenu.getContentPane().add(option);

        JTextField variables = new JTextField(formula.getVariables()[i]);
        variables.setEditable(false);
        variables.setBounds(142, 113 + 65 * i, 223, 49);
        variables.setFont(new Font("Tahoma", Font.PLAIN, 30));
        variables.setHorizontalAlignment(SwingConstants.CENTER);
        frmMenu.getContentPane().add(variables);

        inputs[i] = option;
    }
    JButton btnNewButton = new JButton("Calculate");

    final Action action = new SwingAction();
    final Action action1 = new SwingAction1();
    btnNewButton.setAction(action);
    btnNewButton.setFont(new Font("Tahoma", Font.PLAIN, 28));
    btnNewButton.setBounds(142, 113 + (formula.getDegree()) * 65, 223, 59);
    frmMenu.getContentPane().add(btnNewButton);

    JButton back = new JButton("Back");
    back.setAction(action1);
    back.setFont(new Font("Tahoma", Font.PLAIN, 28));
    back.setBounds(240, 120 + (formula.getDegree() + 1) * 65, 223, 59);
    frmMenu.getContentPane().add(back);

    result.setText("");
    result.setEditable(false);
    result.setColumns(10);
    result.setBounds(380, 113 + (formula.getDegree()) * 65, 223, 59);
    result.setFont(new Font("Tahoma", Font.PLAIN, 30));
    result.setHorizontalAlignment(SwingConstants.CENTER);
    frmMenu.getContentPane().add(result);
```

Sets Title of the window as the title of the formula

Keeps running till a text field has been created for each variable

Creates button to calculate formula answer based on user input

Creates back button to return to formula list menu

Use of external Libraries

For my implementation of “Science Buddy”, I also used javas swing library for the GUI and JAMA’s library for matrix solving.

```
public Matrix times (double s) {
   Matrix X = new Matrix(m,n);
   double[][] C = X.getArray();
   for (int i = 0; i < m; i++) {
      for (int j = 0; j < n; j++) {
         C[i][j] = s*A[i][j];
      }
   }
   return X;
}
```

Function that multiplies two matrices by JAMA

Iterates for the x dimension of the matrix

Sources

"JAMA : A Java Matrix Package." *JAMA: Java Matrix Package*. N.p., 23 Nov. 2012. Web. 23 Dec. 2016.

"Matrices in Chemistry." *Matrices in Chemistry*. N.p., n.d. Web. 23 Dec. 2016.

Mudannayake, M.M.M.R.B. "A Quick and Easy Way to Implement Real Time Filtering in Swing..." *A Quick and Easy Way to Implement Real Time Filtering in Swing...* N.p., 27 May 2011. Web. 23 Dec. 2016.

"Package javax.swing." *Javax.swing (Java Platform SE 7)*. Java Foundation, n.d. Web. 23 Dec. 2016.

Word Count: 956