Criteria C

Variable Declaration

```
public static double[][][] tabledata = new double [0][0][0];
public static double[][][] data;
public static double table1data[][];
public static boolean status = false;
public double[][] newArray;
public double[][] newArray2 = new double[3][4];
public double[][] arrResult;
public static double coal1;
public static double coal2;
public static double coal3;
public List<List<Double>> result = new ArrayList<>();
```

This code defines several variables and arrays of double and boolean types, including static and non-static ones. The static variables *tabledata, data, table1data,* and *status* are initialized with values, while the non-static variables *newArray, newArray2*, and *arrResult* have either no size or a predefined size. Additionally, the code defines three static variables *coal1, coal2,* and *coal3*. Finally, an ArrayList result is defined to hold lists of doubles.

```
double v1;
double v2 = 38.8;
double v3 = 0.0;
double vadd = 1.2;
```

The code above declares four double variables: v1, v2, v3, and vadd. The initial value of v2 is set to 38.8, v3 is set to 0.0, and vadd is set to 1.2. The value of v1 is not initialized and will be calculated later in the code. These variables will be used to keep track of the values of the different components in the composite.

```
private void coal2upMouseClicked(java.awt.event.MouseEvent evt) {

  v2 += 0.10;
  v1 = 100.0 - v2 - v3-vadd;
  Coal1.setText(String.valueOf(df.format(v1)));
  Coal2.setText(String.valueOf(df.format(v2)));

}

private void coal2downMouseClicked(java.awt.event.MouseEvent evt) {

  v2 -= 0.10;
  v1 = 100.0 - v2 - v3-vadd;
  Coal1.setText(String.valueOf(df.format(v1)));
  Coal2.setText(String.valueOf(df.format(v2)));

}

private void coal3upMouseClicked(java.awt.event.MouseEvent evt) {

  v3 += 0.10;
  v1 = 100.0 - v2 - v3-vadd;
  Coal1.setText(String.valueOf(df.format(v1)));
  Coal3.setText(String.valueOf(df.format(v3)));

}

private void coal3downMouseClicked(java.awt.event.MouseEvent evt) {
  v3 -= 0.10;
  v1 = 100.0 - v2 - v3-vadd;
  Coal1.setText(String.valueOf(df.format(v1)));
  Coal3.setText(String.valueOf(df.format(v3)));

}
```

This code contains four methods that correspond to button click events for adjusting the values of three double variables (v1, v2, and v3) and a constant (vadd). The first two methods increase or decrease the value of v2 by 0.10, the third method increases or decreases the value of v3 by 0.10, and the last method calculates the value of v1 as 100.0 minus the values of v2, v3, and vadd. The values of all four variables are then displayed in JTextField components named Coal1, Coal2, and Coal3.

```
private void opendocMouseClicked(java.awt.event.MouseEvent evt) {
File docxFile = new File("document.docx");
try {
  Desktop.getDesktop().open(docxFile);
} catch (IOException e) {
  JOptionPane.showMessageDialog(null, ("Error opening file: " + e.getMessage()));
}

}
```

This Java code defines a method "opendocMouseClicked" that opens a specific file named "document.docx" using the default application associated with the file type. When the user clicks on a component that has a mouse click listener attached to it, the code creates a new File object representing the file to be opened, and attempts to open the file using the Desktop.getDesktop().open() method. If an IOException is thrown during the process, the code displays an error message in a JOptionPane dialog box.

Recursive Method Sumproduct

```
public static double sumproduct(double[] arr1, List<List<Double>> arr2, int x) {
    if (arr1.length == 0 || arr2.isEmpty() || arr2.get(0).isEmpty()) {
        return 0;
    }
    double value = arr2.get(0).get(x);
    if (Double.isNaN(value)) {
        value = 0;
    }
    return arr1[0] * value + sumproduct(Arrays.copyOfRange(arr1, 1, arr1.length)
            ,arr2.subList(1, arr2.size()), x);
}
```

This method sumproduct calculates the sum of the product of corresponding entries in two arrays of numbers, similar to the SUMPRODUCT function in Microsoft Excel. It takes in an array of doubles arr1, a list of lists of doubles arr2, and an int x. The method recursively calculates the sumproduct by multiplying the first element of arr1 with the x-th element of the first row of arr2 and then adding the result to the sum of the product of the remaining elements of arr1 and rows of arr2. The method also checks if arr1 or arr2 is empty, or if the first row of arr2 is empty, before computing the sumproduct.

Linked List Queue

```java
public static class Queue<T> {
    private LinkedList<T> queue;

    public Queue() {
        queue = new LinkedList<>();
    }

    public void enqueue(T item) {
        queue.addLast(item);
    }

    public T dequeue() {
        if (isEmpty()) {
            throw new RuntimeException("Queue is empty");
        }
        return queue.removeFirst();
    }

    public T peek() {
        if (isEmpty()) {
            throw new RuntimeException("Queue is empty");
        }
        return queue.getFirst();
    }

    public boolean isEmpty() {
        return queue.isEmpty();
    }

    public int size() {
        return queue.size();
    }
}
```

The Queue class here is a Java implementation of a queue data structure using a linked list. It has four methods - enqueue, dequeue, peek, and isEmpty - that correspond to the standard operations of a queue. The class is generic and can hold objects of any data type. The enqueue method adds an item to the end of the queue, dequeue removes and returns the first item in the queue, peek returns the first item without removing it, and isEmpty checks if the queue is empty.

Apache POI read document using queue

```
private double[][][] data;

public static double[][][] readTablesFromWordDocument(String fileName) {
    try {
        // Open the Word document
        File file = new File(fileName);
        double[][][] data;
        try (FileInputStream fis = new FileInputStream(file); XWPFDocument docx = new XWPFDocument(fis)) {
            // Use a Queue to keep track of the tables to process
            Queue<XWPFTable> queue = new Queue<>();
            int tableCount = docx.getTables().size();
            data = new double[tableCount][][];

            // Enqueue each table in the document
            for (XWPFTable table : docx.getTables()) {
                queue.enqueue(table);
            }

            int tableIndex = 0;

            // Process tables in the order they were encountered in the document
            while (!queue.isEmpty()) {
                XWPFTable table = queue.dequeue();

                // Get the number of rows and columns in the table
                int rows = table.getRows().size();
                int cols = table.getRow(0).getTableCells().size();

                // Create a 2D array to store the table data
                data[tableIndex] = new double[rows][cols];

                // Loop through all rows in the table
                for (int i = 0; i < rows; i++) {
                    XWPFTableRow row = table.getRow(i);

                    String cellText = row.getCell(0).getText();
                    if (!cellText.matches("-?\\d+(\\.\\d+)?")) {
                        continue;
                    }

                    // Loop through all cells in the row
                    for (int j = 0; j < cols; j++) {
                        cellText = row.getCell(j).getText();
                        data[tableIndex][i][j] = Double.parseDouble(cellText);
                    }
                }

                tableIndex++;
            }
        }

        return data;
    } catch (IOException e) {
        JOptionPane.showMessageDialog(null, ("Error opening file: " + e.getMessage()));
        return null;
    }
}

public double[][][] getData() {
    return data;
}
```

This Java class uses the Apache POI library to read tables from a Word document and store the data in a 3D array called "data." It employs a Queue to keep track of the tables to process, and for each table, it creates a 2D array to store the table data. The method loops through each row and

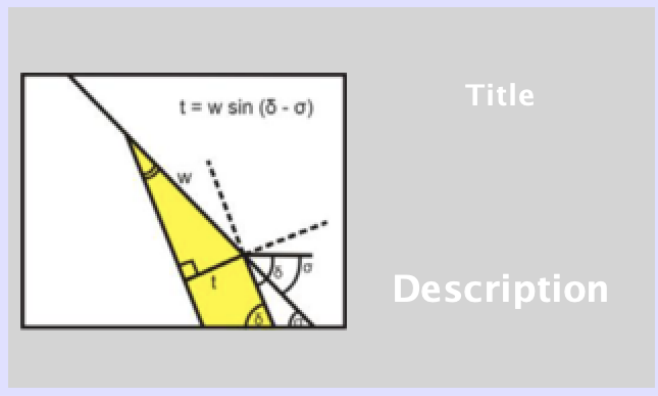
cell to parse and stores the data. The class also contains a getter method to access the "data" array. If there is an error while reading the file, the method displays an error message using JOptionPane. The if statement with the condition if (!cellText.matches("-?\d+(\.\d+)?")) and the corresponding continue statement is used to skip the row iteration if the text value of the first cell in the row does not match a regular expression pattern for a double value. This ensures that only rows containing valid double values are processed and added to the data array.

Print Result with Apache POI

```
public class PrintDoc {
    public void createWordDocument(String title, List<List<Double>> arr, double[] com, double[] ratio) {
    // Create a new Word document
    XWPFDocument doc = new XWPFDocument();
    // Get the document's default page size
    DecimalFormat df = new DecimalFormat("#.##");
    CTPageSz pageSize = doc.getDocument().getBody().addNewSectPr().addNewPgSz();
    pageSize.setOrient(STPageOrientation.LANDSCAPE);
    XWPFParagraph paragraph = doc.createParagraph();

    // Create a new table in the document
    XWPFTable table = doc.createTable();

    // Add the data from the array to the table
    String[] data = {"Materials", "Ratio", "TM \n%AR", "IM\n%ADB", "ASH\n%ADB",
        "ASH\n%DB", "VM\n%ADB", "FC\n%ADB", "TS\n%ADB", "TS\n%AR", "CV\n%ADB", "CV\nar"};
    XWPFTableRow headerRow = table.createRow();
    for (int i = 0; i < data.length; i++) {
        headerRow.createCell().setText(data[i]);
        headerRow.getCell(i).getParagraphs().get(0).createRun().setBold(true);}
    for (int i = 0; i < ratio.length; i++) {
        ratio[i] = Double.valueOf(df.format(ratio[i]));}
    String[] stringArray = new String[ratio.length];
    for (int i = 0; i < ratio.length; i++) {
        stringArray[i] = String.valueOf(ratio[i]);
    }
    String[][] header = {
        {"Coal 1", ""},
        {"Coal 2", ""},
        {"Coal 3", ""},};
    for (int i = 0; i < header.length; i++) {
        header[i][1] = stringArray[i];}
    for (int i = 0; i < header.length; i++) {
        XWPFTableRow row = table.createRow();
        row.createCell().setText(String.valueOf(header[i][0]));
        row.createCell().setText(String.valueOf(header[i][1]));
        for (int j = 0; j < arr.get(i).size(); j++) {
            row.createCell().setText(df.format(arr.get(i).get(j)));
        }
    }
    XWPFTableRow row = table.createRow();
    for (int i = -1; i < com.length; i++) {
        if(i == -1) {
            row.createCell().setText(("composite"));
        } else {
            row.createCell().setText(String.valueOf(df.format(com[i])));
        }
    }
    // Save the document
    try {
        FileOutputStream out = new FileOutputStream(new File("result.docx"));
        doc.write(out);
        out.close();
        doc.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

}
```

This code creates a Word document in Java using the Apache POI library. It creates a table in the document and adds data to it in the form of a 2D array and a 1D array. It sets the page orientation to landscape and formats the decimal values to 2 decimal places using DecimalFormat. Finally, it saves the document as "result.docx."

Encapsulation




```
public class Card extends javax.swing.JPanel {

    public Color getColor1() {
        return color1;
    }

    public void setColor1(Color color1) {
        this.color1 = color1;
    }

    public Color getColor2() {
        return color2;
    }

    public void setColor2(Color color2) {
        this.color2 = color2;
    }

    private Color color1;
    private Color color2;

    public Card() {
        initComponents();
        setOpaque(false);
        color1 = Color.BLACK;
        color2 = Color.WHITE;
    }

    public void setData(Model_Card data) {
        lbIcon.setIcon(data.getIcon());
        lbTitle.setText(data.getTitle());
        lbValues.setText(data.getValues());

    }
```

The method setData(Model_Card data) sets the card's icon, title, and description using the data from a Model_Card object passed as a parameter.

```
public class Model_Card {

    public Icon getIcon() {
        return icon;
    }

    public void setIcon(Icon icon) {
        this.icon = icon;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public String getValues() {
        return values;
    }

    public void setValues(String values) {
        this.values = values;
    }



    public Model_Card(Icon icon, String title, String values) {
        this.icon = icon;
        this.title = title;
        this.values = values;
    }

    public Model_Card() {
    }

    private Icon icon;
    private String title;
    private String values;

}
```

This class contains getter and setter methods for each of these properties and a constructor to create a new card object and a default constructor.

```java
public class formula {
    private double omega;
    private double delta;
    private double w;

    public formula(double omega, double delta, double w) {
        this.omega = omega;
        this.delta = delta;
        this.w = w;
    }

    public double calc1() {
        return this.w * Math.sin(Math.toRadians(this.delta + this.omega));
    }
    public double calc2() {
        return this.w * Math.sin(Math.toRadians(this.omega - this.delta));
    }
    public double calc3() {
        return this.w * Math.sin(Math.toRadians(this.delta - this.omega));
    }
    public double calc4() {
        return this.w * Math.sin(Math.toRadians(90 - this.delta - this.omega));
    }
    public double calc5() {
        return this.w * Math.sin(Math.toRadians(this.omega));
    }
    public double calc6() {
        return this.w * Math.sin(Math.toRadians(90 - this.omega));
    }
}
```

The code is a class called "formula" that performs calculations on three private instance variables: omega, delta, and w. The class contains six methods, each of which performs a different mathematical operation on the instance variables. The class also has a constructor that initializes the instance variables, encapsulating them and preventing direct access to their values from outside the class. This encapsulation ensures that the values of the instance variables can only be modified through methods defined in the class, promoting better code organization and reducing the likelihood of programming errors.

Inheritance

```java
public class volumeformula extends formula {
    private double length;
    private double width;

    public volumeformula(double omega, double delta, double w, double length, double width) {
        super(omega, delta, w);
        this.length = length;
        this.width = width;
    }

    public double volume(double height) {
        return this.width * this.length * height;
    }
}
```

The code is a class called "volumeformula" that extends the "formula" class, inheriting its instance variables and methods. The "volumeformula" class also contains two private instance variables: length and width. The class has a constructor that initializes all instance variables by calling the constructor of the superclass, "formula", with the inherited omega, delta, and w values, as well as the length and width values specific to the "volumeformula" class. The class also contains a method called "volume" that takes a single parameter, "height," and returns the result of multiplying the width, length, and height instance variables, encapsulating these variables and promoting code reusability. This use of inheritance allows the "volumeformula" class to build upon the functionality of the "formula" class while adding new capabilities specific to its use case.

Word Count: 931