Project Design

Group 2:
John Abueg, William Rios Crespo, Joshua Kerley, Michael Lancaster
CMSC 495 63802

Author Note:

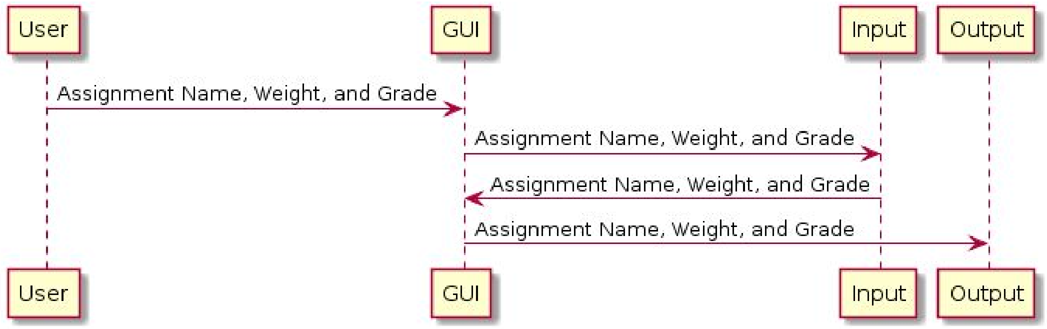
Document Version PD007
Date: 2018-10-11

Version Control

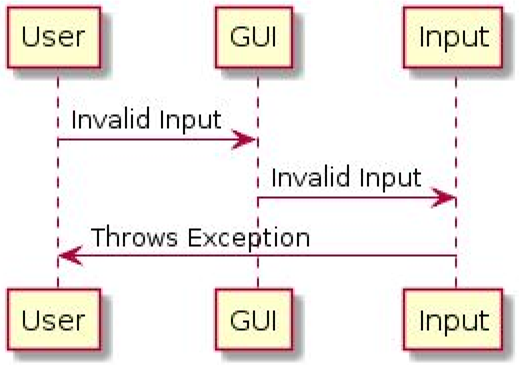
Document	Date	Action	Name	Email
PD001	2018-09-14	Created	John Abueg	j.abueg13@gmail.com
PD002	2018-09-15	Add event-trace diagrams and unresolved risks and risk mitigations	Michael Lancaster	lancastermc@gmail.com
PD003	2018-09-16	Added Pseudocode for GUI subsystem	Joshua Kerley	jkillakerlz@gmail.com
PD004	2018-09-16	Input Pseudocode added	John Abueg	j.abueg13@gmail.com
PD005	2018-09-16	Added Pseudocode for File subsystem	William Rios Crespo	william.rioscrespo19@gmail.com
PD006	2018-09-16	Reviewed	John Abueg Josh Kerley Michael Lancaster William Rios Crespo	j.abueg13@gmail.com jkillakerlz@gmail.com william.rioscrespo19@gmail.com lancastermc@gmail.com
PD007	2018-10-11	Update diagrams and pseudocode	Michael Lancaster	lancastermc@gmail.com

Grade Calculator Project Design:

1. **Event-trace Diagrams:**
   a. Scenario 1:
   Description: The user inputs data that is saved into a file. This data is used to calculate final grade, maximum, minimum, mean, median, and standard deviation scores.
   Precondition: All data is entered in valid formats.
   Post-condition: The data is saved and can be recalled later if desired. This data is also used to calculate and display final grade, maximum, minimum, mean, median, and standard deviation scores.
   Diagram:

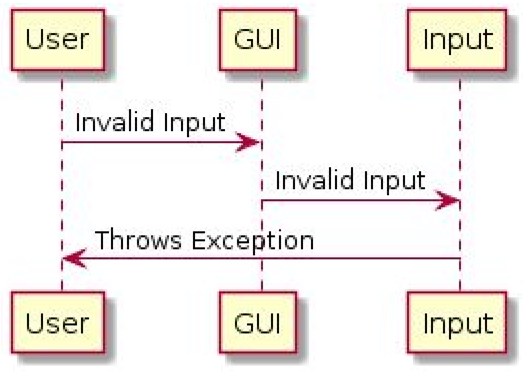

User
GUI
Input
Output
Assignment Name, Weight, and Grade
Assignment Name, Weight, and Grade
Assignment Name, Weight, and Grade
Assignment Name, Weight, and Grade
User
GUI
Input
Output

   b. Scenario 2:
   Description: The user inputs invalid data to be saved in a file.
   Precondition: At least one piece of data entered is invalid in format.
   Post-condition: When invalid data is entered an exception is thrown.
   Diagram:

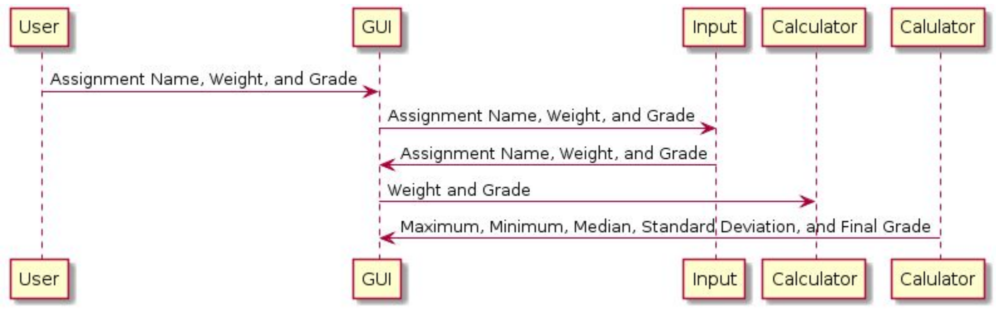


   c. Scenario 3:
   Description: The user inputs invalid data to be displayed.
   Precondition: At least one piece of data entered is invalid in format.
   Post-condition: When invalid data is entered an exception is thrown.

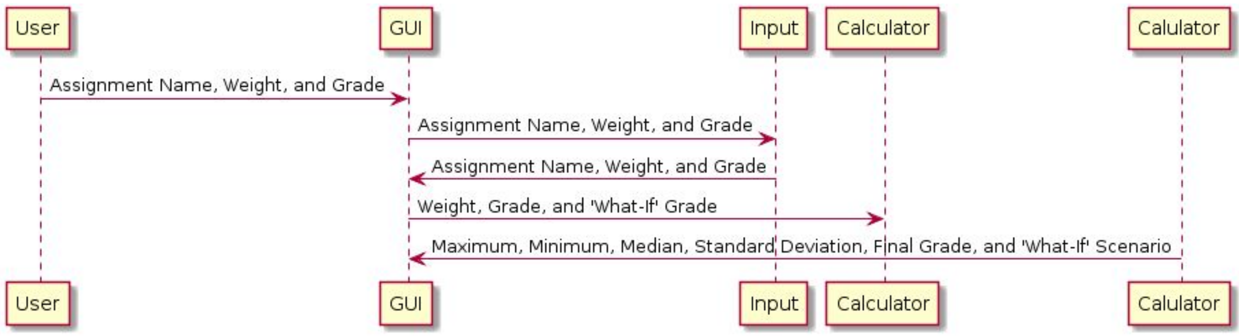
Diagram:



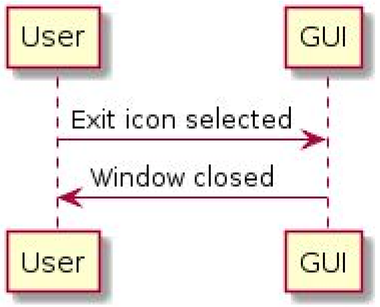
d. Scenario 4:
Description: The user inputs data that is displayed and used to calculate final grade, maximum, minimum, mean, median, and standard deviation scores.
Precondition: All data is entered in valid formats.
Post-condition: This data is displayed and used to calculate and display final grade, maximum, minimum, mean, median, and standard deviation scores.
Diagram:


User
GUI
Input
Calculator
Calulator
Assignment Name, Weight, and Grade
Assignment Name, Weight, and Grade
Assignment Name, Weight, and Grade
Weight and Grade
Maximum, Minimum, Median, Standard Deviation, and Final Grade
User
GUI
Input
Calculator
Calulator

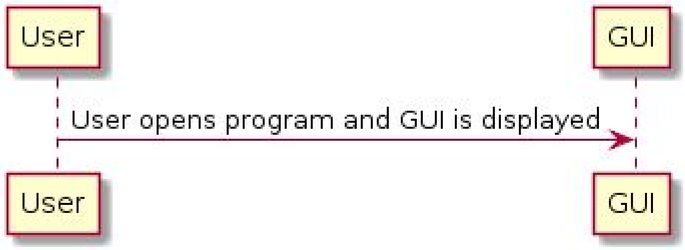
e. What-if:
Description: The user enters assignment information and wants to enter a ‘What-If’ scenario.
Precondition: The user has entered valid assignment information for at least one assignment.
Post-condition: The program displays the grade and weight needed to meet “What-If” scenario


User
GUI
Input
Calculator
Calulator
Assignment Name, Weight, and Grade
Assignment Name, Weight, and Grade
Assignment Name, Weight, and Grade
Weight, Grade, and 'What-If' Grade
Maximum, Minimum, Median, Standard Deviation, Final Grade, and 'What-If' Scenario
User
GUI
Input
Calculator
Calulator

f. Shut down:
Description: The user is done using program and selects the exit icon.
Precondition: The program is open and ready for use.
Post-condition: Window closes.
Diagram:



g. Start-up:
Description: The user selects program to start.
Precondition: Program compiles correctly.
Post-condition: Program opens for user use.
Diagram:



2. Class design pseudocode

a. Input subsystem:

```
Class Input {

            File file;
            ArrayList<String> assignmentNameList;
            ArrayList<Double> gradeList;
            ArrayList<Double> weightList;

    public Input(String filePath){
            file = new File(filePath);
    }

    public Input(){
            file = null;
    }

    public void readFile(){
            try{
                    verifyExtension();
                    Scanner input = new Scanner(file);
                    String text = “”;
                    while (input.hasNext()){
                            text += input.next() + “ “;
                    }
            input.close();
            String[] words = test.split(“\\s+);

            For (int i = 0; i < words.length; i++){
                    if(words[i].equalsIgnoeCase(“Assignment”) && words[i +
1].equalsIgnoreCase (“Name:”)){
                            if(words[i+2].equalsIgnoreCase(“Weight:”){
                                    assignmentNameList.add(“null”);
                                    i+=1;
                            }else if (containSpecialCharacter(words[i+2])){
                                    throw new Exception(
                                            "Must only enter numbers and letters for
assigment name. Special characters are not allowed.");
                            }else if(words[i+2].equalsIgnoreCase(“Grade:”)){
                                    throw new Exception(
                                            "Invalid Format. Please export a file that
meets with the required criteria.");
                            }else{
                                    assignmentNameList.add(words[i+2]);
                                    i+=2;
                            }else if (words[i].equalsIgnoeCase(“Weight:”){
                                    if(word[i+1].equalsIgnoreCase(“Grades:”)){
                                            weightList.add(1.0);
```

```
                                        }else if (numericValue(words[i+1])){
                                                if(Double.parseDouble(words[i+1])  <= 0.0){
                                                        throw new Exception(
                                                                "Invalid input, please enter a
positive weight that is higher than zero.");
                                                }else{
                                                weightList.add(Double.parseDouble(words[i+1]));
                                                i+=1;
                                                }
                                        }else if(weightList.size() >
assignmentNameList.size()){
                                                throw new Exception(
                                                        "Invalid Format. Please export a file
that meets with the required criteria.");
                                        }else{
                                                throw new Exception(
                                                        "Invalid Format. Please export a file
that meets with the required criteria.");
                                        }else if(words[i].equalsIgnoreCase(“Grade:”)){

if(words[i+1].equalsIgnorCase(“Assignment”) &&
words[i+2].equalsIgnoreCase(“Name:”)){
                                                        gradeList.add(0.0);
                                                }else if(numericValue(words[i+1])){

gradeList.add(Double.parseDouble(words[i+1];
                                                        i+=1;
                                                }else if(words[i].equalsIgnoreCase(“A”) ||
words[i].equalsIgnoreCase(“B”) || words[i].equalsIgnoreCase(“C”) ||
words[i].equalsIgnoreCase(“D”) || words[i].equalsIgnoreCase(“F”)){
                                                        throw new Exception("Invalid
Format, must only enter numbers for a grade.");
                                                }else if (gradeList.size() > weightList.size() ||
gradeList.size() > assignmentNameList.size()) {
                                                        throw new Exception(
                                                                "Invalid Format. Please
export a file that meets with the required criteria.");
                                                }else{
                                                        throw new Exception("Invalid
Format, must only enter numbers for a grade.");
                                                }
                                        }else{
```

```
                                        throw new Exception("Invalid Format.
Please export a file that meets with the required criteria.");
                                    }
                            }catch (Exception exception){
                                    displayMessageDialog(exception.getMessage());
                            }
                    }

                    public void userInput(){
                            try{
                                    if(containSpecialCharacters(assignmentName)){
                                            throw new Excpetion(
                                                    "Must only enter numbers and letters
for assigment name. Special characters are not allowed.");
                                    }else if(assignmentNameList.replace(" ",
"").equalsIgnoreCase("")){
                                            assignmentNameList.add("null);
                                    }else{

assignmentNameList.add(assignmentName);
                                    }

                                    if(weight.replace(" ", "").equalsIgnoreCase("")){
                                            weightList.add(1.0);
                                    }else if(numericValue(weight)){

weightList.add(Double.parseDouble(weight));
                                    }else{
                                            throw new Exception("Invalid Format.
Please export a file that meets with the required criteria.");
                                    }

                                    if(grade.replace(" ", "").equalsIgnoreCase("")){
                                            gradeList.add(0.0);
                                    }else if(numericValue(grade)){
                                            gradeList.add(Double.parseDouble(grade));
                                    }else if(grade.equalsIgnoreCase("A") ||
grade.equalsIgnoreCase("B") || grade.equalsIgnoreCase("C") ||
grade.equalsIgnoreCase("D") || grade.equalsIgnoreCase("F")){
                                            throw new Exception("Invalid Format, must
only enter numbers for a grade.");
                                    }else{
```

```
                                                throw new Exception("Invalid Format.
Please export a file that meets with the required criteria.");
                                        }
                                }catch(Exception exception){
                                        displayMessageDialog(exception.getMessage());
                                }
                        }

                        public File getFile(){
                                return file;
                        }

                        public ArrayList<String> getAssignmentList(){
                                return assignmentNameList;
                        }

                        public ArrayList<Double> getGradeList(){
                                return gradeList;
                        }

                        public ArrayList<Double> getWeightList(){
                                return weightList;
                        }

                        public void verifyExtension() throws Exception{
                        if(check if file has extension) && (check if file has extension other
        than .dat)
                        throw new exception
                        }

                        public boolean containSpecialCharacters(String assignmentName)
        throws Exception{

        if(Pattern.compile("[A-Za-z0-9]").matcher(assignmentName).find()){
                                        return true;
                                }
                                return false;
                        }

                        public static void displayMessageDialog(String message){
                                JOptionPane.showMessageDialog(new JFrame(),
        message, "Input - Error", JOption.INFORMATION_MESSAGE);
                        }
        }
```

b. **Calculator subsystem:**

```
Class calculator {

        double minimum(List<Double> gradeList){

                Return Collections.min(grades);
        }

        double maximum(List<Double> gradeList){

                Return Collections.max(grades);
        }
        double median(List<Double> gradeList, List<Double> weightList){
                ArrayList<Double> weightedList;
                Double currentWeight;

                For (i = 0; 1 < grades.size(); i++){
                        currentWeight = int * (weights.get(i)/1)

                        For(j=0; j < currentWeight; j++){
                        Add grades.get(i) to weightedList
                        }

                        if((weights.get(i) - currentWeight) != 0){
                                Multiple grades.get(i) and( weights.get(i) remainder
                    1) and add to weightedList
                        }
                }
        Collections.sort(weightedList);
        if(weightedList.size() % 2 == 0){
                Return median of even numbers
        }else{
                Return median of odd numbers
        }
        }

        double mean(List<Double> gradeList, List<Double> weightList){
                double total;
                double weightTotal;

                for(int i = 0; i < grades.size(); i++){
                        Total += (grades.get(i) * weight.get(i))
                }
                        Return (total/weightTotal)
        }

        public static char letterGrade(ArrayList<Double>, ArrayList<Double>
    weight){
                double meanAverage;

                meanAverage = mean(grades,weight)
                char letter;

                Switch ((int) meanAverage / 10){
                cases to determine letter grade.
                }
                return letter;
        }

        double stdDev(List<Double> gradeList, List<Double> weightList){

                double meanAverage;
                ArrayList<Double> devList; //holds all grades for standard
                deviation calculations

                meanAverage = mean(grades, weight);

                For (int i = 0; i<devList.size(); i++) {
                        double tempResult = Math.pow(devList.get(i) -
meanAverage, 2)
                        devList.set(i, tempResult)
                        }

                        double stDevMean = mean(devList, weight);
                        return Math.sqrt(stDevMean);
                }

        double whatIf(ArrayList<Double> grades, List<Double> weight, int
        whatIfValue) {
                ArrayList<Double> gradeList;
                ArrayList<Double> weightList;
                double meanAverage;
                double newMeanAverage;
                double targetAverage;
                double tempGrade;
                int numAddedValues = 0; //counts number (weight) of added
        grades
                int target = whatIfValue;
                meanAverage = mean(gradeList, weightList); //get actual mean
        avg

                if(meanAverage < whatIfValue){

                do { //repeats until "what if" is met/exceeded
                        Add grade value of 100 to gList;
                        Add weight value of 1 to wList;
                        Add 1 to numAddedValues;
                        newMeanAverage = mean(gradeList, weightList); //get new
        mean avg
                }
                while (newMeanAverage < target);


                int tempGrade = 100;
                double tempMeanAverage; //holds mean value for do-while loop

                //do-while loop to find the min. needed points to reach "what if"
                do { //repeats until "what if" is higher than mean avg
                3a- Subtract 1 from tempGrade;
                        Replace gradeList(size - 1) value with tempGrade;
                        tempMeanAverage = mean(gradeList, weightList);
                }while (tempMeanAverage > target);
                }else{
                        do{
                                gradeList.add(0.0);
                                weightList.add(1.0);
                                numAddedValues += 1;
                                newMeanAverage = mean(gradeList, weightList);
                }while(newMeanAverage > whatIfValue);
                tempGrade =0;
                double tempMeanAverage;
                do{
                        tempGrade += 1;
                        gradeList.set(gradeList.size() -1, tempGrade);
                        tempMeanAverage = mean(gradeList,weightList);
                }while(tempMeanAverage < whatIfValue);
                }
```

```
                if(meanAverage < whatIfValue){
                        tempGrade = tempGrade + ((numAddedValues - 1) * 100);
                }
                tagetAverage = tempGrade / numAddedValues;

                double[] whatIfNumbers = new double [2];
                whatIfNumber[0] = targetAverage;
                whatIfNumbers[1] = numAddedValues;
                return whatIfNumbers;
        }
```

c. **GUI subsystem**

```
public class GUI{
        private JFrame frmGuiCalculator;
        private JScrollPane scrollPane;
        private JTextArea textArea;
        private JLabel assignmentNameLabel = new JLabel("Assignment Name:");
        private JLabel weightLabel = new JLabel("Weight:");
        private JLabel gradeLabel = new JLabel("Grade:");
        private JLabel valueLabel;
        private JTextField assignmentNameText = new JTextField();
        private JTextField weightText = new JTextField();
        private JTextField gradeText = new JTextField();
        private JTextField valueTextField;
        private JPanel userInputPanel;
        private JPanel buttonPanel;
        private JPanel backgroundPanel;
        private JPanel inputPanel;
        private JPanel whatIfPanel;
        private JPanel optionPanel;
        private JButton btnInput;
        private JButton btnOutput;
        private JButton btnEnter;
        private JButton btnCalculate;
        private JButton btnEnter2;
        private JButton btnClear;
        private Input input = null;
        private double[] whatIfNumbers = null;
        private int listCounter = 0;
        private boolean calculate = false;
```

```
public static void main(String[] args){
    new GUI instance
}

public GUI(){
    initialize();
}

private void initialize(){
    Create frame and specifications for GUI

    Create and add panel for save and load buttons to frame
    Create and add save and load buttons to panel

    Create and add display panel to frame
    Create and add textArea to display panel

    Create and add input panel to frame

    Create and add userInputPanel to input panel
    Add labels and text fields for user input to userInputPanel

    Create and add option panel to inputPanel
    Create and add enter, calculate, and clear buttons to option panel

    Create and add whatIfPanel to inputPanel
    Create and add text field, label, and button to whatIfPanel
    Create and add action listener for whatIfPanel button
}

private void inputFile(){

    Create JFileChooser and JFileChooser specifications
    try-catch loop for loading data into program from file
}

private void outputFile(){
Create JFileChooser and JFileChooser specifications
try-catch loop for loading data into file from from program
}

private void displayResults(){
```

```
            If statement to check input fields have data{

                For statement to append input fields to textArea{
                }

                If statement to check if calculate button has been pressed{
                If calculate button has been pressed append maximum, minimum,
        mean, median, standard deviation, and file grade to textArea
                }

                If statement to check whatIfNumbers field is not empty{
                If whatIfNumbers is not empty append whatIf scenario to textArea
                }
            }
        }
}
```

d. Output subsystem

```
class Output {
        File name;

        Public void inputFile(File file){
                name = file;
        }
        public void write(String text){
                try{
                        verifyExtension();
                        BufferedWriter writer = new BufferedWriter(new
                        FileWriter(name));
                        Write input to File.
                        Close writer
                } catch (Exception exception){
                        Display Error.
                }
        }
        public static void showError(String errorMessage, String title){
                JOptionPane.showMessageDialog
        }
        public void verifyExtension() throws Exception{
                if(check if file has extension) && (check if file has extension other
than .dat)
                throw new exception
                }
        }
}
```

3. Unresolved risks and risk mitigation

At this time the identified risks are a user selecting an incorrect input file with an invalid file extension, invalid inputs from user such as special characters, letter grades, negative scores, or grades higher than one-hundred, and overwriting incorrect file.
To mitigate the risk concerning file extensions the selection of files will be limited to .dat files. If any other file extension is selected an exception will be thrown. To mitigate the risk of invalid inputs the applications will throw and catch exceptions that will open a dialog window that will notify the user of invalid inputs. To mitigate the risk of overwriting incorrect files file selection will be limited to only .dat files. A dialog window will open to confirm the overwriting of files.
At this time there are no risks that are unresolved.