**Tiffany Lourdraj**

**ITSS 3312.003**

**Professor Short**

**09/27/2020**

**Project 4**

**Project: FlashCards GUI – Project 4**

**Analysis, Testing and Design**

This program was part four of a continuing lesson on GUI which stands for graphical user interface.This program was section three of a proceeding program on a exercise on creating a program for a children’s math solver. Our primary objective was to make a child's numerical learning device. We still integrated exception handling, but this time it varied. One big variation was the use of Gridpane and VBox which contributed to the frontend design of the program. The essential problem I ran into building this program was guaranteeing the toggle and down arrow was working. It was also difficult to enter multiple digits as well because the sample code used was not taking in anything other than single digits. In any case, by using consistent testing and by guaranteeing that there is execution of new classes comprising of strategies, which are additionally conjured from the driver class, you can successfully make a GUI that will work with the previous codes At the completion of the entire program, I had a great time playing around with the different outputs from the GUI.

The design aspect of one’s code is significant. If one happens to have code that isn't indented appropriately, or doesn't have a simple stream to it, others programmers looking over your code may struggle following along. The progression of it went like this for me:

1. Project 4 extending to Application
2. Initialize and define methods used in returning user inputs
3. Creating Node Declarations
   1. Update with the label and textfield
4. Create VBox for MathType
   1. Rb1-4 for the different math types
5. GUI implementation
   1. That includes GridPane, add children, set constraints and visuals
   2. Create and show Scene
6. Event and Exception Handling Methods
   1. If try statements to throw and catch
7. Public static void main class

The stream of the code is direct, and you can return and see whenever where the variable are being defined. Also when you portray different variables, try to name it something that explains what it does. For example, keyCode.DOWN is a variable that shows and checks if the user is using the down arrow on the keyboard. Rather than renaming every variable in a set something different, keep it consistent by naming it something that groups it together. For example, keyCode.DOWN and keyCode.UP go together, and you can tell that by the similar naming conventions.

To test this code, it is a huge segment, if not the most laborious piece of this process. It is prescribed to utilize the debugger tool, but Eclipse is also great at finding errors early on in the code, and discovers alternate approaches to make your code run easily. One stumble I made was that I neglected to add the invalid statement if there was an answer other in the toggle section than the ones specified to accept. However I fixed that easily with the code below:

System.***out***.println(Oper);

submit.requestFocus();

}

**else** {

lErrRB.setText("Invalid");

rb1.requestFocus();

}

});

The else if statement came in handy for this specific project. There are over 50 different combinations for the user to select, so while that may take forever to check, it is worth it in the end. This is how professionals make sure that no one can hack and disturb their solid code. This is utilized to catch blunders, for example, and you want to make sure the output for the user shows up as invalid if need be.for example, if a user inputs a letter or a character to the low high values (which require a number), it should not terminate the whole program but instead catch it and return an error message for the user. Be that as it may, at that point I understood you need to add the catch part of the exemption handling. Taking everything into account, I fixed any issues that may have emerged through my code by basically testing wrong client data sources and utilizing arbitrary numbers (even negative and decimals to test on the off chance that it handles inaccurate information sources), and ensuring the cluster returned determined the right to mistaken issues precisely. An accomplished engineer will test their code on different events to guarantee they have not missed whatever could be a risk to the trustworthiness of their program.

Example code:

Start of program (Stage primaryStage, and the node declarations)

@Override

**public** **void** start(Stage primaryStage) {

TextField tfName = **new** TextField();

Label lName = **new** Label("Enter your Name");

Label lErrName = **new** Label();

TextField tfNumProblems = **new** TextField();

tfNumProblems.setMaxWidth(80);

Label lNumProblems = **new** Label("How Many Problems");

Label lErrNumProblems = **new** Label();

TextField tfLowValue = **new** TextField();

tfLowValue.setMaxWidth(80);

Label lLowValue = **new** Label("Lowest value of range of factors");

Label lErrLowValue = **new** Label();

TextField tfHighValue = **new** TextField();

tfHighValue.setMaxWidth(80);

Label lHighValue = **new** Label("Highest value of range of factors");

Label lErrHighValue = **new** Label();

Label lProbType = **new** Label("Problem Type");

VBox vbox = **new** VBox();

RadioButton rb1 = **new** RadioButton("Addition +");

RadioButton rb2 = **new** RadioButton("Subtraction -");

RadioButton rb3 = **new** RadioButton("Multiplication \*");

RadioButton rb4 = **new** RadioButton("Division /");

vbox.getChildren().addAll(rb1, rb2, rb3, rb4);

vbox.setSpacing(10);

// ToggleGroup for VBox and RadioButtons

ToggleGroup rbToggle = **new** ToggleGroup();

rb1.setToggleGroup(rbToggle);

rb2.setToggleGroup(rbToggle);

rb3.setToggleGroup(rbToggle);

rb4.setToggleGroup(rbToggle);

Label lErrRB= **new** Label();

Button submit = **new** Button("GO");

Label lErrSubmit= **new** Label();

Screenshot of example output:

You can see below what happens when you toggle between the different math types, and how it works in real time

