```
package semester_project_calculator;

import java.util.ArrayList;
import javafx.application.Application;
//import javafx.event.ActionEvent;
//import javafx.event.EventHandler;
import javafx.geometry.Insets;
import javafx.geometry.Pos;
import javafx.scene.Scene;
//import javafx.scene.image.Image;
//import javafx.scene.image.ImageView;
import javafx.scene.layout.BorderPane;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.VBox;
import javafx.scene.layout.HBox;
//import javafx.scene.control.Button;
//import javafx.scene.control.Label;
//import javafx.scene.control.TextField;
import javafx.scene.control.*;
import javafx.scene.control.Alert.AlertType;
import javafx.stage.Stage;
import java.io.*;
import java.util.Scanner;

/**
 *
 * @author rmald09 and hlind01
 */

/************MIGHT NEED BIG DECIMAL FOR SOME OF THESE******************/
public class Semester_Project_Calculator extends Application
{
    private static Label label = new Label("");
    private String input = "";
    private String calcString = "";
    private static final char SQUARE_ROOT = 0x221A;
    private static final char SUPER_N = 0x207F;
    private static final char SUPER_X = 0x02E3;
    private static final char ARROW = 0x2192;
    private static final char SUPER_2 = 0x00B2;
    private static final char SUPER_Y = 0x02B8;
    private static final char PI = 0x03C0;
    private final String FILENAME = "Program.txt";
    private static String programName;
    private static int inputCount;
    private static String programString;
    private static int programIndex;
    private static String returnArg = "0.0";

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args)
    {
        //use this for help pop-up?
            launch(args);
    }

    /**This method is where we create and display the scene on our stage.
     *
     * @param primaryStage The stage used to display our scene
     */
    @Override
    public void start(Stage primaryStage)
    {
        TextInputDialog textDialog = new TextInputDialog();

        try
        {
```

```
70              FileReader myFile = new FileReader(FILENAME);
71              Scanner inputFile = new Scanner(myFile);
72              FileWriter fwriter = new FileWriter(FILENAME, true);
73              PrintWriter programFile = new PrintWriter(fwriter);
74              ArrayList programContents = new ArrayList();
75              ArrayList programNames = new ArrayList();
76              ArrayList programInputCount = new ArrayList();
77              //Create Buttons
78              Button button_1 = new Button("1");
79              Button button_2 = new Button("2");
80              Button button_3 = new Button("3");
81              Button button_4 = new Button("4");
82              Button button_5 = new Button("5");
83              Button button_6 = new Button("6");
84              Button button_7 = new Button("7");
85              Button button_8 = new Button("8");
86              Button button_9 = new Button("9");
87              Button button_0 = new Button("0");
88              Button button_decimal = new Button(".");
89              Button clear = new Button("A/C");
90              Button button_equal = new Button("=");
91              Button button_add = new Button("+");
92              Button button_subtract = new Button("-");
93              Button button_divide = new Button("/");
94              Button button_multiply = new Button("*");
95              Button sin = new Button("sin");
96              Button cos = new Button("cos");
97              Button tan = new Button("tan");
98              Button oneOverX = new Button("1/x");
99              Button squared = new Button("x"+SUPER_2);
100             Button power = new Button("x"+SUPER_Y);
101             Button pi = new Button(PI+"");
102             Button scientificNotation = new Button("EE");
103             Button logorithm = new Button("log");
104             Button openParenthesis = new Button("(");
105             Button closeParenthesis = new Button(")");
106             Button factorial = new Button("!");
107             Button squareRoot = new Button(SQUARE_ROOT+"");
108             Button nRoot = new Button(SUPER_N+""+SQUARE_ROOT);
109             Button ln = new Button("ln");
110             Button eToTheX = new Button("e"+SUPER_X);
111             Button abs = new Button("|x|");
112             Button memAdd = new Button("M+");
113             Button memRecall = new Button("MR");
114             Button memClear = new Button("MC");
115             Button delete = new Button("DEL");
116             Button fToD = new Button("F"+ARROW+"D");
117             Button neg = new Button("(-)");
118             Button new_button = new Button("New Program");
119             Button program_button = new Button("Run Program");
120             Button input_button = new Button("Input");
121             Button erase_button = new Button("Erase");
122             Button save_button = new Button("Save");
123
124             /*
125             program buttons/ delete or clear program? / coding
126             ******************Fix delete button for user input. delete char in
127             calcString. if previous character is s or t etc, delete again. Compare
128             String lengths, make input same length as calcString. would need to
129             recheck that it won't cause issues with negative operator
130
131             //need to account for not pressing clear before entering more test
132             memory buttons
133             EE button
134             f>D button
135
136             help button?
137             Writing programs
138             rad/deg
```

```
139             css
140             throw exceptions done?
141             keyboard input
142             Customize joptionpane
143             Big Decimal
144             decimal precision
145             what else?
146             */
147             /***Replace with Deg Rad Radio
                buttons*****************************************************/
148             //Create textfield
149         //***************Fix alignment and positioning*************
150             //Can we align cursor to the right?
151
152             // Create a GridPane for Program functions
153             GridPane gridpaneTop = new GridPane();
154
155             // Create a GridPane for special functions
156             GridPane gridpaneLeft = new GridPane();
157
158             // Create a GridPane for numbers & arithmatic operators functions
159             GridPane gridpaneRight = new GridPane();
160
161             // Insert the buttons into the GridPane
162             gridpaneTop.add(new_button, 0, 0);
163             gridpaneTop.add(input_button, 1, 0);
164             gridpaneTop.add(program_button, 2, 0);
165             gridpaneTop.add(erase_button, 3, 0);
166             gridpaneTop.add(save_button, 4, 0);
167
168             gridpaneRight.add(memAdd, 0, 0);
169             gridpaneRight.add(button_1, 1, 0);
170             gridpaneRight.add(button_2, 2, 0);
171             gridpaneRight.add(button_3, 3, 0);
172             gridpaneRight.add(button_add, 4, 0);
173             gridpaneRight.add(clear, 5, 0);
174
175             gridpaneRight.add(memRecall, 0, 1);
176             gridpaneRight.add(button_4, 1, 1);
177             gridpaneRight.add(button_5, 2, 1);
178             gridpaneRight.add(button_6, 3, 1);
179             gridpaneRight.add(button_subtract, 4, 1);
180             gridpaneRight.add(delete, 5, 1);
181
182             gridpaneRight.add(memClear, 0, 2);
183             gridpaneRight.add(button_7, 1, 2);
184             gridpaneRight.add(button_8, 2, 2);
185             gridpaneRight.add(button_9, 3, 2);
186             gridpaneRight.add(button_multiply, 4, 2);
187             gridpaneRight.add(openParenthesis, 5, 2);
188
189             gridpaneRight.add(neg, 0, 3);
190             gridpaneRight.add(button_decimal, 1, 3);
191             gridpaneRight.add(button_0, 2, 3);
192             gridpaneRight.add(button_equal, 3, 3);
193             gridpaneRight.add(button_divide, 4, 3);
194             gridpaneRight.add(closeParenthesis, 5, 3);
195
196             gridpaneLeft.add(sin, 0, 0);
197             gridpaneLeft.add(cos, 1, 0);
198             gridpaneLeft.add(tan, 2, 0);
199             gridpaneLeft.add(pi, 3, 0);
200
201             gridpaneLeft.add(squared, 0, 1);
202             gridpaneLeft.add(power, 1, 1);
203             gridpaneLeft.add(squareRoot, 2, 1);
204             gridpaneLeft.add(nRoot, 3, 1);
205
206             gridpaneLeft.add(abs, 0, 2);
```

```
207                 gridpaneLeft.add(oneOverX, 1, 2);
208                 gridpaneLeft.add(fToD, 2, 2);
209                 gridpaneLeft.add(factorial, 3, 2);
210 
211                 gridpaneLeft.add(scientificNotation, 0, 3);
212                 gridpaneLeft.add(logorithm, 1, 3);
213                 gridpaneLeft.add(ln, 2, 3);
214                 gridpaneLeft.add(eToTheX, 3, 3);
215 
216                 // Add spacing and padding to the GridPane
217                 gridpaneRight.setHgap(10);
218                 gridpaneRight.setVgap(10);
219                 // fix button size
220                 /*Use css to do @media? or responsive calculator?*/
221                 //fix alignment
222                 gridpaneTop.setPadding(new Insets(5));
223                 gridpaneTop.setHgap(10);
224                 gridpaneTop.setVgap(10);
225 
226                 gridpaneRight.setPadding(new Insets(5));
227 
228                 gridpaneLeft.setHgap(10);
229                 gridpaneLeft.setVgap(10);
230                 //fix button size
231                 gridpaneLeft.setPadding(new Insets(5));
232 
233                 // Create HBox and add gridpanes to HBox
234                 HBox hbox = new HBox(gridpaneLeft, gridpaneRight);
235                 //dividing line in center?
236                 hbox.setPadding(new Insets(20));
237 
238                 //Create a BorderPane
239                 BorderPane borderpane = new BorderPane();
240 
241                 //add label
242                 borderpane.setTop(label);
243 
244                 //add program buttons to setRight
245                 borderpane.setRight(gridpaneTop);
246 
247                 //adjust spacing
248                 borderpane.setPadding(new Insets(20));
249 
250                 // Create VBox
251                 /*******PUT EVERYTHING IN BOARDER PANE?********/
252                 VBox vbox = new VBox(borderpane, hbox);
253                 //adjust spacing?
254 
255                 button_1.setOnAction(event ->
256                 {
257                     input += 1;
258                     calcString += 1;
259                     label.setText(input);
260                 });
261 
262                 button_2.setOnAction(event ->
263                 {
264                     input += 2;
265                     calcString += 2;
266                     label.setText(input);
267                 });
268 
269                 button_3.setOnAction(event ->
270                 {
271                     input += 3;
272                     calcString += 3;
273                     label.setText(input);
274                 });
275 
```

```
276                 button_4.setOnAction(event ->
277                 {
278                     input += 4;
279                     calcString += 4;
280                     label.setText(input);
281                 });
282 
283                 button_5.setOnAction(event ->
284                 {
285                     input += 5;
286                     calcString += 5;
287                     label.setText(input);
288                 });
289 
290                 button_6.setOnAction(event ->
291                 {
292                     input += 6;
293                     calcString += 6;
294                     label.setText(input);
295                 });
296 
297                 button_7.setOnAction(event ->
298                 {
299                     input += 7;
300                     calcString += 7;
301                     label.setText(input);
302                 });
303 
304                 button_8.setOnAction(event ->
305                 {
306                     input += 8;
307                     calcString += 8;
308                     label.setText(input);
309                 });
310 
311                 button_9.setOnAction(event ->
312                 {
313                     input += 9;
314                     calcString += 9;
315                     label.setText(input);
316                 });
317 
318                 button_0.setOnAction(event ->
319                 {
320                     input += 0;
321                     calcString += 0;
322                     label.setText(input);
323                 });
324 
325                 button_decimal.setOnAction(event ->
326                 {
327                     input += ".";
328                     calcString += ".";
329                     label.setText(input);
330                 });
331 
332                 clear.setOnAction(event ->
333                 {
334                     input = "";
335                     calcString = "";
336                     label.setText(input);
337                 });
338 
339                 button_equal.setOnAction(event ->
340                 {
341                     if (input.length() > 0)
342                         label.setText("" + calculate(interpret(calcString)));
343                     //need to convert to answer for next button press. set answer default
                        to "", change after button_equal, reset at A/C
```

```
344             });
345
346             button_add.setOnAction(event ->
347             {
348                 input += "+";
349                 calcString += "+";
350                 label.setText(input);
351             });
352
353             button_subtract.setOnAction(event ->
354             {
355                 input += "-";
356                 calcString += "-";
357                 label.setText(input);
358             });
359
360             button_divide.setOnAction(event ->
361             {
362                 input += "/";
363                 calcString += "/";
364                 label.setText(input);
365             });
366
367             button_multiply.setOnAction(event ->
368             {
369                 input += "*";
370                 calcString += "*";
371                 label.setText(input);
372             });
373
374             openParenthesis.setOnAction(event ->
375             {
376                 input += "(";
377                 calcString += "(";
378                 label.setText(input);
379             });
380             closeParenthesis.setOnAction(event ->
381             {
382                 input += ")";
383                 calcString += ")";
384                 label.setText(input);
385             });
386
387             sin.setOnAction(event ->
388             {
389                 input += "sin(";
390                 calcString += "s("; /*this will feed in the sine operator to the calulate
391                 String. All operators were going to use single character symbols to
                    allow for
392                 easier interpretation of the calcString String, but this caused issues
                    with
393                 order of operations. Now, all unary operators will include a '('.*/
394                 label.setText(input);
395             });
396
397             cos.setOnAction(event ->
398             {
399                 input += "cos(";
400                 calcString += "c(";
401                 label.setText(input);
402             });
403
404             tan.setOnAction(event ->
405             {
406                 input += "tan(";
407                 calcString += "t(";
408                 label.setText(input);
409             });
410
```

```
411             neg.setOnAction(event ->
412             {
413                 input += "-";
414                 calcString += "n";
415                 label.setText(input);
416             });
417 
418             oneOverX.setOnAction(event ->
419             {
420                 input += "1/(";
421                 calcString += "1/(";
422                 label.setText(input);
423             });
424     //        Button  = new Button("x^2") ^2;
425             squared.setOnAction(event ->
426             {
427                 input += "^(2)";
428                 calcString += "^(2)"; /*Need to see how this reads in and if it causes
                    issues*/
429                 label.setText(input);
430             });
431 
432             power.setOnAction(event ->
433             {
434                 input += "^(";
435                 calcString += "^(";
436                 label.setText(input);
437             });
438     //        Button pi = new Button("Pi") pi unicode;
439             pi.setOnAction(event ->
440             {
441                 input += ""+PI;
442                 calcString += "p";
443                 label.setText(input);
444             });
445     //        Button scientificNotation = new Button("EE") E; **********come back to this
446             scientificNotation.setOnAction(event ->
447             {
448                 input += "*10^(";
449                 calcString += "(1)*10^("; /*make sure this reads in correctly*/
450                 label.setText(input);
451             });
452     //        Button logorithm = new Button("log") L;
453             logorithm.setOnAction(event ->
454             {
455                 input += "log(";
456                 calcString += "L(";
457                 label.setText(input);
458             });
459 
460             factorial.setOnAction(event ->
461             {
462                 input += "!";
463                 calcString += "f";
464                 label.setText(input);
465             });
466 
467             squareRoot.setOnAction(event ->
468             {
469                 input += SQUARE_ROOT+"(";
470                 calcString += "S(";
471                 label.setText(input);
472             });
473 
474             nRoot.setOnAction(event ->
475             {
476                 input += SUPER_N+""+SQUARE_ROOT+"(";
477                 calcString += "X(";
478                 label.setText(input);
```

```
479             });
480
481             ln.setOnAction(event ->
482             {
483                 input += "ln(";
484                 calcString += "l(";
485                 label.setText(input);
486             });
487
488             eToTheX.setOnAction(event ->
489             {
490                 input += " e^(";
491                 calcString += "e^(";
492                 label.setText(input);
493             });
494
495             //need to treat | as (
496             abs.setOnAction(event ->
497             {
498                 input += "|";
499                 calcString += "|";
500                 label.setText(input);
501             });
502     /******************need to make a label to indicate stored memory value
503     memAdd.setOnAction(event ->
504             {
505                 input += "";
506                 calcString += "";
507                 label.setText(input);
508             });
509
510     memRecall.setOnAction(event ->
511             {
512                 input += "";
513                 calcString += "";
514                 label.setText(input);
515             });
516     memClear.setOnAction(event ->
517             {
518                 input += "";
519                 calcString += "";
520                 label.setText(input);
521             });
522     */
523             delete.setOnAction(event ->
524             {
525                 if (input.length() > 1)
526                     input = input.substring(0,input.length()-1);
527                 else
528                     input = "";
529                 if (calcString.length() > 1)
530                     calcString = calcString.substring(0,calcString.length()-1);
531                 else
532                     calcString = "";
533                 label.setText(input);
534             });
535
536             /************This will take some thought***************
537              put f>d at the beginning of the string?
538              should this be d>f? that would be more useful, or just get rid of it*/
539             fToD.setOnAction(event ->
540             {
541                 input += "F";
542                 calcString += "F";
543                 label.setText(input);
544             });
545
546             new_button.setOnAction(event ->
547             {
```

```
548                 input="";
549                 calcString = "";
550                 label.setText(input);
551 
552                 alert("information","Program Function","Instructions","Please "
553                         + "enter your program using the calculator buttons for "
554                         + "operators and the \"Input\" button for variable data "
555                         + "input.",1);
556 
557                 alert("information","Program Function","Instructions (continued):",
558                         "Press \"Save\" when you are done or A/C to cancel",1);
559             });
560 
561             program_button.setOnAction(event ->
562             {
563                 Program program = new Program();
564 
565                 int programIndex = 0;
566 
567                 while (inputFile.hasNext())
568                 {
569                     String temp = inputFile.nextLine();
570                     programInputCount.add(programIndex, temp);
571                     temp = inputFile.nextLine();
572                     programNames.add(programIndex, temp);
573                     temp = inputFile.nextLine();
574                     programContents.add(programIndex, temp);
575                     programIndex++;
576                 }
577 
578                 programNames.add(0, "Select Program");
579 
580                 ChoiceDialog choicePopup = new ChoiceDialog(programNames.get(0),
                    programNames);
581                 choicePopup.setTitle("Program Function");
582                 choicePopup.setHeaderText("Instructions");
583                 choicePopup.setContentText("Please select a program:\n");
584 
585                 input="";
586                 calcString = "";
587                 label.setText(input);
588                 choicePopup.showAndWait();
589                 int countIndex = 0;
590                 if (choicePopup.getSelectedItem().equals(programNames.get(0)))
591                 {
592                     //do something about this
593                 }
594                 if (choicePopup.getSelectedItem().equals(programNames.get(1)))
595                 {
596                     countIndex = 0;
597                 }
598                 if (choicePopup.getSelectedItem().equals(programNames.get(2)))
599                 {
600                     countIndex = 1;
601                 }
602                 if (choicePopup.getSelectedItem().equals(programNames.get(3)))
603                 {
604                     countIndex = 2;
605                 }
606                 if (choicePopup.getSelectedItem().equals(programNames.get(4)))
607                 {
608                     countIndex = 3;
609                 }
610                 programNames.remove(0);
611 
612                 //System.out.println(programContents.get(countIndex));
613                 alert("information","Program Function","Program: "
614                         + choicePopup.getSelectedItem(), "You will be propted "
615                                 + "for user input. There are "
```

```
                                + programInputCount.get(countIndex)
                                + " inputs",1);

                String testValue = (String)programInputCount.get(countIndex);

                for (int index = 0; index < Integer.parseInt(testValue); index++)
                {
                    textDialog.getEditor().clear();
                    textDialog.setTitle("Program Function");
                    textDialog.setHeaderText("User Input\nProgram Name: " +
                    choicePopup.getSelectedItem() + "\nUser Input at 'i' variables: " +
                    programContents.get(countIndex));
                    textDialog.setContentText("Please enter the variable value");
                    textDialog.showAndWait();
                }
                //probably need input validation here
                //problem here, maybe use dropdown?
            });
            save_button.setOnAction(event ->
            {
                textDialog.setTitle("Program Function");
                textDialog.setHeaderText("Save Program");
                textDialog.setContentText("Please enter the program name");
                textDialog.showAndWait();
                //get test input, save eveything to file, us program name input in alert


    //            Display message "name your program", call the program constructor,
    feeding in calcString, writeToFile method,
    //            clear input and calcStrings, "Save successful!"
                alert("information","Program Function","Save Program", "Save
                Successful!",1);

                input="";
                calcString = "";
                label.setText(input);
            });
            input_button.setOnAction(event ->
            {
                //need to incriment input index
                input += "USER INPUT";
                calcString += "I";
                label.setText(input);
            });
            erase_button.setOnAction(event ->
            {
    //             will erase program from arrayList of current prgrams, call
    eraseProgram
    //             methodinput and redo repopulate available programNames. Ask which
    program, ask if sure
            });

            /*
            rad/deg make a radio button instead?
            program
            input
            run? (for programNames)

            what else?
            */

    /*KeyHandler (lots to do here), ActionEvent
            -make sure label is updated live as keys are used
            -allow for button or keyboard input
            -exception handling (bad input)*/

            //PROGRAM BUTTON / FUNCTIONALITY
            /*
            -Add button
```

```
679             -use file to */
680
681             //Drop-down menu?
682             /*
683             -help button
684             -display current programNames
685             -add degree radians here?
686             -other settings? custom colors?
687             */
688
689             // Add the GridPane to the Scene
690             Scene scene = new Scene(vbox, 570, 350);
691
692             // Add the Scene to the Stage
693             primaryStage.setTitle("H&R Calculator");
694             primaryStage.setScene(scene);
695
696             // Display the Stage
697             primaryStage.show();
698         }
699
700         catch (Exception e)
701         {
702             alert("information","Program Function","Error", "An Error occured with the
                program file.",1);
703         }
704
705     }
706
707     /*
708     ***************interpret method
709     -accepts String
710
711     -breaks String into operators and operands (each is its own object or use
712     arrayList for variable length array?)
713
714     -Create enum to handle operators?
715
716     -returns / creates ArrayList to be calculated
717     */
718     //work in progress, just getting ideas down
719     public static ArrayList interpret(String input)
720     {
721         String newArgument = "";
722         int decimalCount = 0;
723         int negCount = 0;
724         int operatorCount = 0;
725         int argumentIndex = 0;
726         ArrayList<Argument> arguments = new ArrayList<>();
727
728 //***************need to account for double decimals and negatives with exceptions
729         try
730         {
731             for (int index = 0; index < input.length(); index++)
732             {
733 //                if (decimalCount > 1 || negCount > 1) 12*(12-2)
734 //                {
735 //                    throw new Exception();
736 //                }
737
738                 //is it a number?
739                 if (input.charAt(index) >= 48 && input.charAt(index) <= 57 ||
740                         input.charAt(index) == '.' || input.charAt(index)== 'n')
741                 {
742                     //check for decimal/neg values
743                     if (input.charAt(index) == '.' )
744                     {
745                         decimalCount++;
746                         newArgument += input.charAt(index);
```

```
747                     }
748
749                     else if (input.charAt(index) == 'n' && input.charAt(index+1)
750                             >= 48 && input.charAt(index+1) <= 57)
751                     {
752                         negCount++;
753                         newArgument += "-";
754                     }
755
756                     else if (input.charAt(index) == 'n')
757                         newArgument += "-1";
758                     //This way it will use implicit multiplication to correct
759                     //but, if at the end of the string, throw exception
760                     else
761                         newArgument += input.charAt(index);
762                 }
763
764                 else
765                 {
766                     //is it Pi or e?
767                     if (input.charAt(index) == 'p')
768                     {
769                         if (newArgument.length()>0)
770                         {
771                             arguments.add(new
                                 Argument(Double.parseDouble(newArgument),'#'));
772                             newArgument = "";
773                             arguments.add(new Argument(Math.PI,'#'));
774                         }
775
776                         else
777                         {
778                             arguments.add(new Argument(Math.PI,'#'));
779                         }
780                     }
781                     else if (input.charAt(index) == 'e')
782                             if (newArgument.length()>0)
783                     {
784                         arguments.add(new
                             Argument(Double.parseDouble(newArgument),'#'));
785                         newArgument = "";
786                         arguments.add(new Argument(Math.E,'#'));
787                     }
788
789                     else
790                     {
791                         arguments.add(new Argument(Math.E,'#'));
792                     }
793
794                     //need to deal with multiple operators 1*sin(8) leading operators
795                     else if (newArgument.length()>0)
796                     {
797                         arguments.add(new Argument(Double.parseDouble(newArgument),'#'));
798                         newArgument = "";
799                         arguments.add(new Argument(0, input.charAt(index)));
800                     }
801
802                     else
803                         arguments.add(new Argument(0, input.charAt(index)));
804                 }
805
806             }
807
808
809             if (input.charAt(input.length()-1) >= 48 && input.charAt(input.length()-1)
                <= 57 ||
810                     input.charAt(input.length()-1) == '.')
811             {
812                 arguments.add(new Argument(Double.parseDouble(newArgument),'#'));
```

```
813                 newArgument = "";
814             }
815 
816             //adds last argument if more than one argument
817             else if (arguments.size() > 1 && input.charAt(input.length()-1) != ')'
818                    && input.charAt(input.length()-1) != 'f'  &&
                       input.charAt(input.length()-1) != 'p')
819             {
820                 //need to account for trailing )
821                 arguments.add(new Argument(Double.parseDouble(newArgument),'#'));
822                 newArgument = "";
823             }
824 
825 /*****************************************************************************/

826             //Single Factorial
827 //            else if (arguments.size() > 1 && input.charAt(input.length()-1) == 'f')
828 //            {
829 //                arguments.add(new Argument(0, 'f'));
830 //                newArgument = "";
831 //            }
832 
833         }
834         catch (Exception e)//needs work
835         {
836             alert("information","Calculator","Error", "An Error occured. Please check
                your input format",1);
837 
838             arguments.clear();
839             arguments.add(new Argument(0.0,'#'));
840         }
841         finally
842         {
843             return arguments;
844         }
845     }
846 
847     /*
848     ***************calculate Method************Check Description
849     -accepts String(not label String, need to create a String that the user
850     doesn't see. Will hold the operators and operands. Operators will be chars,
851     not String (sin would be: s, cos: c, etc. This will allow for easier
852     interpretation of the String.)
853 
854     -calls interperet
855 
856     -Calls order of operations method This could be used to scan for parenthesis,
857     exponents, first, going down the order of operations. Will reorder items to
858     allow for correct calculations.
859 
860     -performs calculations
861 
862     -sets input and answerString Strings to answer
863 
864     -returns String answer
865     */
866 
867     public static String calculate(ArrayList arguments)
868     {
869         double answer = 0; //Needed? use argument?
870         String answerString = "";
871         int start = -1;
872         int end = -1;
873         ArrayList openP = new ArrayList();
874         ArrayList<Argument> evaluateArgs = new ArrayList<>();
875 
876         //reads arrayList, evaluates, removes element
877 
878         /************************need to treat | as (*****************
```

```
879             do we even need this button? */
880
881             /*looks for parenthesis
882             -How do we account for sin(1*(2+3)^4)?
883             -Put everything in parenthesis (display and arguments)
884             -evaluate all parenthesis from right to left. Since only arithmatic will
885             be performed, shouldn't cause any issues for final evaluations
886             */
887             try
888             {
889             //are there '(' ? Find them, create ArrayList of their positions
890             for (int index = 0; index < arguments.size(); index++)
891             {
892                 Argument tempArg = (Argument)arguments.get(index);
893                 Argument evalArg = new Argument(tempArg);
894                 if (evalArg.getOperator() == '(')
895                     openP.add(0,index);
896             }
897
898             //If there are:
899             while (openP.size() > 0)
900             {
901                 //testing read arguments
902     //            for(int i = 0; i< arguments.size(); ++i)
903     //                System.out.print(arguments.get(i));
904     //            System.out.print("\n");
905     //
906     //            //testing openP values
907     //             for(int i = 0; i< openP.size(); ++i)
908     //                System.out.println("( at:"+openP.get(i));
909     //
910                 start = (int)openP.get(0);
911                 for (int index = start; index < arguments.size(); index++)
912                 {
913                     Argument tempArg = (Argument)arguments.get(index);
914                     Argument evalArg = new Argument(tempArg);
915                     if (evalArg.getOperator() == ')')
916                     {
917                         end = index;
918                         index = arguments.size()-1;
919                     }
920                     //testing closeP value
921                 //    System.out.println(end);
922                 }
923                 //delete parenthesis
924                 arguments.remove(start);
925                 arguments.remove(end-1);
926
927                 //read in arguments to evaluateArgs to be sent to evaluate method
928                 for (int index = start; index < (end-1); index++)
929                 {
930                     Argument tempArg = (Argument)arguments.get(index);
931                     Argument evalArg = new Argument(tempArg);
932                     evaluateArgs.add(evalArg);
933                 }
934
935                 //remove arguments added to evaluateArgs from arguments
936                 for (int index = (end-2); index >= (start); index--)
937                 {
938                     arguments.remove(index);
939                 }
940
941                 //send evaluateArgs into evaulate method
942                 Argument tempArg = evaluate(evaluateArgs);
943                 //make deep copy
944                 Argument evalArg = new Argument(tempArg);
945
946                 //Put arguement returned from evaluateArgs back in arguments Array list
947                 arguments.add(start, evalArg);
```

```
948                 openP.remove(0);
949
950                 //for debugging
951   //              answerString ="";
952   //              System.out.print("evaluateArgs:");
953   //              for (int index = 0; index < evaluateArgs.size(); index++)
954   //              {
955   //                  answerString += evaluateArgs.get(index).toString();//for testing only
956   //              }
957   //              System.out.println(""+answerString);//for testing only
958   //
959   //              answerString ="";
960   //              System.out.print("arguments:");
961   //              for (int index = 0; index < arguments.size(); index++)
962   //              {
963   //                  answerString += arguments.get(index).toString();//for testing only
964   //              }
965   //              System.out.println(""+answerString);//for testing only
966
967                 //reset evaluteArgs for next iteration
968                 evaluateArgs.clear();
969             }
970
971                 //if no/ no more parenthesis
972
973             //when there are no more calulations, return final argument
974             Argument tempArg = evaluate(arguments);
975             Argument evalArg = new Argument(tempArg);
976             returnArg = "" + evalArg.getValue();
977             //testing
978   //          answerString ="";
979   //          System.out.print("arguments in calculate method:");
980   //          for (int index = 0; index < arguments.size(); index++)
981   //          {
982   //              answerString += arguments.get(index).toString();//for testing only
983   //          }
984   //          System.out.println(""+answerString);//for testing only
985   //          //if not a number, should throw an exception
986             }
987             catch (Exception e)
988             {
989                 alert("information","Calculator","Error", "An Error occured. Please check
                    your input format",1);
990
991                 arguments.clear();
992                 arguments.add(new Argument(0.0,'#'));
993             }
994             finally
995             {
996                 return "" + returnArg;
997             }
998         }
999
1000        /**This method performs unary expressions (trig, log, abs) and basic 4
1001         * function arithmetic by accepting an ArrayList of arguments and evaluating
1002         * products, sums, etc in the order of operations then returning the answer.
1003         *
1004         * @param arguments
1005         * @return
1006         */
1007        public static Argument evaluate(ArrayList arguments)
1008        {
1009            Argument tempArg =
                (Argument)addSubtract(multiplyDivide(exponents(unaryOperator(arguments)))).get(0)
                ;
1010            Argument evaluation = new Argument(tempArg);
1011
1012            return evaluation;
1013        }
```

```
1014        /*
1015        ***************format Method (work into interpret method)
1016        -needs to account for decimals bad syntax (3.3.4 type of errors)
1017        //this needs work
1018                if (input.contains(".")) //check for decimals >1 in between operators
1019                    input = "ERROR";
1020                else
1021
1022        ***************
1023        */
1024        /* Deal with unary operators*/
1025        public static ArrayList unaryOperator(ArrayList arguments)
1026        {
1027            ArrayList<Integer> deleteIndex = new ArrayList<>();
1028            double temp = 0;
1029            char operator = '0';
1030            int deleteIndexSize = 0;
1031
1032            for (int index = 0; index<arguments.size(); ++index)
1033            {
1034                Argument tempArg = (Argument)arguments.get(index);
1035                Argument evalArg = new Argument(tempArg);
1036
1037                if (evalArg.getOperator() == 's' || evalArg.getOperator() == 'c' ||
1038                    evalArg.getOperator() == 't' || evalArg.getOperator() == 'L' ||
1039                    evalArg.getOperator() == 'l' || evalArg.getOperator() == 'f' )
1040                {
1041                    operator = evalArg.getOperator();
1042                    switch (operator)
1043                    {
1044                        case 's':
1045                        {
1046                            Argument tempArg2 = (Argument)arguments.get(index+1);
1047                            Argument evalArg2 = new Argument(tempArg2);
1048                            arguments.set(index, new
                                Argument(Math.sin(evalArg2.getValue()), '#'));
1049                            deleteIndex.add(0, index+1);
1050                            System.out.println("Argument size before delete:
                                "+arguments.size());
1051                            System.out.println("Delete index contents: "+deleteIndex.get(0));
1052                            break;
1053                        }
1054                        case 'c':
1055                        {
1056                            Argument tempArg2 = (Argument)arguments.get(index+1);
1057                            Argument evalArg2 = new Argument(tempArg2);
1058                            arguments.set(index, new
                                Argument(Math.cos(evalArg2.getValue()), '#'));
1059                            deleteIndex.add(0, index+1);
1060                            break;
1061                        }
1062                        case 't':
1063                        {
1064                            Argument tempArg2 = (Argument)arguments.get(index+1);
1065                            Argument evalArg2 = new Argument(tempArg2);
1066                            arguments.set(index, new
                                Argument(Math.tan(evalArg2.getValue()), '#'));
1067                            deleteIndex.add(0, index+1);
1068                            break;
1069                        }
1070                        case 'L':
1071                        {
1072                            Argument tempArg2 = (Argument)arguments.get(index+1);
1073                            Argument evalArg2 = new Argument(tempArg2);
1074                            arguments.set(index, new
                                Argument(Math.log10(evalArg2.getValue()), '#'));
1075                            deleteIndex.add(0, index+1);
1076                            break;
1077                        }
```

```
                    case 'l':
                    {
                        Argument tempArg2 = (Argument)arguments.get(index+1);
                        Argument evalArg2 = new Argument(tempArg2);
                        arguments.set(index, new
                        Argument(Math.log(evalArg2.getValue()), '#'));
                        deleteIndex.add(0, index+1);
                        break;
                    }
                    case 'f':
                    {
                        double answer = 1;
                        Argument tempArg2 = (Argument)arguments.get(index-1);
                        Argument evalArg2 = new Argument(tempArg2);
                        //need to check for remainder, throw exception if not integer
                        int factorial = (int)evalArg2.getValue();
                        if (factorial > 0)
                        {


                            for (int i = 1; i <= factorial; ++i)
                            {
                                answer *= i;
                            }
                        }
                        else if (factorial == 0)
                            answer = 1;
                        arguments.set(index, new Argument(answer, '#'));
                        deleteIndex.add(0, index-1);
                        break;
                    }
                    default:
                    {

                         break;
                    }
                }
            }
        }

        for (int index = 0; index < deleteIndex.size(); index++ )
        {
            int deleteArg = deleteIndex.get(index);
            arguments.remove(deleteArg);
        }
        System.out.println("Argument size after delete: "+arguments.size());
        deleteIndex.clear();

        return arguments;
    }

    //evaluate exponents, roots r -> l (need to fix, also problems with multiple
    operators)***************START HERE**************
    public static ArrayList exponents(ArrayList arguments)
    {
        ArrayList<Integer> exponentIndex = new ArrayList<>();
        ArrayList<Integer> deleteIndex = new ArrayList<>();
        char operator = '0';

        for (int index = 0; index < arguments.size(); index++)
        {
            Argument tempArg = (Argument)arguments.get(index);
            Argument evalArg = new Argument(tempArg);
            if (evalArg.getOperator() == '^' || evalArg.getOperator() == 'S' ||
                evalArg.getOperator() == 'X')
                exponentIndex.add(0,index);
        }

        System.out.print("Argument size before loop:" + arguments.size()+"\n^ iteration
```

```
            arguments before loop: ");
1145            for (int i = 0; i < arguments.size(); ++i)
1146            {
1147                Argument tempArg = (Argument)arguments.get(i);
1148                System.out.print("" + tempArg.getValue());
1149            }
1150            System.out.println("");
1151
1152            System.out.print("^ iteration operators before loop: ");
1153            for (int i = 0; i < arguments.size(); ++i)
1154            {
1155                Argument tempArg = (Argument)arguments.get(i);
1156                System.out.print("" + tempArg.getOperator());
1157            }
1158            System.out.println("");
1159
1160        for (int index = 0; index < exponentIndex.size(); index++)
1161        {
1162            //read in operators at the marked indexes
1163            Argument tempArg = (Argument)arguments.get(exponentIndex.get(index));
1164            Argument evalArg = new Argument(tempArg);
1165
1166            System.out.print("^ iteration arguments: ");
1167            for (int i = 0; i < arguments.size(); ++i)
1168            {
1169                System.out.print("" + evalArg.getValue());
1170            }
1171            System.out.println("");
1172
1173            //determine operator at that index
1174            operator = evalArg.getOperator();
1175
1176            //process operator
1177            switch (operator)
1178            {
1179                case '^':
1180                {
1181                    Argument tempArgY =
                        (Argument)arguments.get(exponentIndex.get(index)+1);
1182                    Argument evalArgY = new Argument(tempArgY);
1183                    Argument tempArgX =
                        (Argument)arguments.get(exponentIndex.get(index)-1);
1184                    Argument evalArgX = new Argument(tempArgX);
1185                    arguments.set(exponentIndex.get(index), new
                        Argument(Math.pow(evalArgX.getValue(),evalArgY.getValue()), '#'));
1186                    arguments.remove(exponentIndex.get(index)+1);
1187                    arguments.remove(exponentIndex.get(index)-1);
1188                    break;
1189                }
1190                case 'S':
1191                {
1192                    Argument tempArg2 =
                        (Argument)arguments.get(exponentIndex.get(index)+1);
1193                    Argument evalArg2 = new Argument(tempArg2);
1194                    arguments.set(exponentIndex.get(index), new
                        Argument(Math.pow(evalArg2.getValue(), 1.0/2.0 ), '#'));
1195                    arguments.remove(exponentIndex.get(index)+1);
1196                    break;
1197                }
1198                case 'X':
1199                {
1200                    Argument tempArgRoot =
                        (Argument)arguments.get(exponentIndex.get(index)+1);
1201                    Argument evalArgRoot = new Argument(tempArgRoot);
1202                    Argument tempArgX =
                        (Argument)arguments.get(exponentIndex.get(index)-1);
1203                    Argument evalArgX = new Argument(tempArgX);
1204                    double xValue = evalArgX.getValue();
1205                    arguments.set(exponentIndex.get(index), new
```

```
                    Argument(Math.pow(evalArgRoot.getValue(),(1.0/xValue)), '#'));
                    arguments.remove(exponentIndex.get(index)+1);
                    arguments.remove(exponentIndex.get(index)-1);
                    break;
                }

                default:
                {
                     break;
                }
            }
        }

        deleteIndex.clear();
        exponentIndex.clear();

        return arguments;
    }

     /*Multiply divide, l > r, all multiplications are implicit*/
    public static ArrayList multiplyDivide(ArrayList arguments)
    {
        ArrayList<Integer> deleteIndex = new ArrayList<>();
        char operator0 = '0';
        char operator1 = '0';
        int updateArgumentSize = 0;

        for (int index = 0; index < arguments.size(); index++)
        {
            Argument tempArg = (Argument)arguments.get(index);
            Argument evalArg = new Argument(tempArg);

            operator0 = evalArg.getOperator();
            if (operator0 == '*')
                deleteIndex.add(0,index);
        }

        //testing
        System.out.print("deleteItems:");
        for (int index = 0; index < deleteIndex.size(); index++)
        {
            System.out.print("" + deleteIndex.get(index));
        }
        System.out.println("");
        int deleteSize = deleteIndex.size();
        for (int index = 0; index < deleteSize; index++)
        {
            int tempIndex = deleteIndex.get(index);
            arguments.remove(tempIndex);
        }

        System.out.print("arguments after running delete:");
        for (int index = 0; index < arguments.size(); index++)
        {
            System.out.print("" + arguments.get(index));
        }
        System.out.println("");

        deleteIndex.clear();
        //
        //
        updateArgumentSize = arguments.size();
        for (int index = 0; index < arguments.size(); index++)
        //while (arguments.size() > 1)       /********************rewrite
        loop********************/
        {
            //testing
        System.out.println("Arguments.size: " + arguments.size());
            System.out.print("iteration arguments: ");
```

```
1273                for (int i = 0; i < arguments.size(); ++i)
1274                {
1275                    Argument tempArg = (Argument)arguments.get(i);
1276                    System.out.print("" + tempArg.getValue());
1277                }
1278                System.out.println("");
1279                //read in operators from left to right
1280
1281                if(arguments.size()-1 != index)
1282                {
1283                    Argument tempArg0 = (Argument)arguments.get(index);
1284                    Argument evalArg0 = new Argument(tempArg0);
1285
1286                    Argument tempArg1 = (Argument)arguments.get(index+1);
1287                    Argument evalArg1 = new Argument(tempArg1);
1288
1289                    operator0 = evalArg0.getOperator();
1290                    operator1 = evalArg1.getOperator();
1291
1292                    if (operator1 == '/')
1293                    {
1294                        Argument tempArg2 = (Argument)arguments.get(index+2);
1295                        Argument evalArg2 = new Argument(tempArg2);
1296                        arguments.set(index, new
                            Argument((double)evalArg0.getValue()/(double)evalArg2.getValue(),
                            '#'));
1297                        arguments.remove(index+1);
1298                        arguments.remove(index+1);
1299                       //updateArgumentSize = updateArgumentSize - 2;
1300                    }
1301                    else if (operator0 == '#' &&  operator1 == '#')
1302                    {
1303                        arguments.set(index, new
                            Argument((double)evalArg0.getValue()*(double)evalArg1.getValue(),
                            '#'));
1304                        arguments.remove(index+1);
1305                        if (arguments.size() > 1)
1306                            index--;
1307                    }
1308                }
1309            }
1310
1311            deleteIndex.clear();
1312
1313            return arguments;
1314        }
1315
1316        public static ArrayList addSubtract(ArrayList arguments)
1317        {
1318            char operator = '0';
1319            int testArgSize = arguments.size();
1320
1321            while (arguments.size() > 1)
1322            {
1323
1324                //testing
1325                System.out.print("+/- iteration arguments: ");
1326                for (int i = 0; i < arguments.size(); ++i)
1327                {
1328                    Argument tempArg = (Argument)arguments.get(i);
1329                    System.out.print("" + tempArg.getValue());
1330                }
1331                System.out.println("");
1332                //read in operators from left to right
1333                Argument tempArg0 = (Argument)arguments.get(0);
1334                Argument evalArg0 = new Argument(tempArg0);
1335
1336                Argument tempArg1 = (Argument)arguments.get(1);
1337                Argument evalArg1 = new Argument(tempArg1);
```

```
1338                testArgSize = arguments.size();
1339                if (arguments.size() > 2)
1340                {
1341                    Argument tempArg2 = (Argument)arguments.get(2);
1342                    Argument evalArg2 = new Argument(tempArg2);
1343                    operator = evalArg1.getOperator();
1344
1345                    if (operator == '-')
1346                    {
1347
1348                        arguments.set(0, new
                            Argument(evalArg0.getValue()-evalArg2.getValue(), '#'));
1349                        arguments.remove(1);
1350                        arguments.remove(1);
1351                    }
1352                    else
1353                    {
1354                        arguments.set(0, new
                            Argument(evalArg0.getValue()+evalArg2.getValue(), '#'));
1355                        arguments.remove(1);
1356                        arguments.remove(1);
1357                    }
1358                testArgSize = arguments.size();
1359                }
1360            }
1361
1362            return arguments;
1363        }
1364
1365        public static void alert(String type, String title, String header, String content,
            int result)
1366        {
1367            /* alert text format
1368                     * Alert alert = new Alert(AlertType.CONFIRMATION);
1369                alert.setTitle("Action Failed");
1370                alert.setHeaderText("Undo Function Works Only From \"Make A Transaction\"
                    and \"History\" Window");
1371                alert.setContentText("Press \"OK\" to go to \"Make A Transaction\" window");
1372                alert.setX(DashboardStage.getX() + 60);
1373                alert.setY(DashboardStage.getY() + 170);
1374                Optional<ButtonType> result = alert.showAndWait();
1375    */
1376
1377                Alert alertPopUp = new Alert(AlertType.CONFIRMATION);
1378                alertPopUp.setTitle(title);
1379                alertPopUp.setHeaderText(header);
1380                alertPopUp.setContentText(content);
1381                alertPopUp.showAndWait();
1382
1383        }
1384    }
1385
1386
```