```java
/**
graph.java
This is the main component of Graph Widget. It creates the main window with the
    equation box and canvas, and contains code to process user-entered equations
    and represent them graphically on the canvas. It references the
    settingsWindow.java application window, and the Settings.java class.
**/

import java.lang.reflect.Array;
import java.util.ArrayList;

import org.eclipse.swt.SWT;
import org.eclipse.swt.events.*;
import org.eclipse.swt.graphics.*;
import org.eclipse.swt.layout.*;
import org.eclipse.swt.program.Program;
import org.eclipse.swt.widgets.*;
import org.eclipse.swt.events.KeyAdapter;
import org.eclipse.swt.events.KeyEvent;

public class graph {

    //List of characters which cannot be used in the equations
    // given by the user, but may be used in the tokenised equation
    private char reservedChars[] = {'[',']','{','}'};
    //List of operators; must be organised in reverse order of operations
    private char operatorList[] = {'-','+','*','/','^'};
    //List of function names; must be organised longest to shortest
    private String functionNames[] = {"sin","cos","tan","abs","ln"};
    //List of variable (and constant) names; called variable names for brevity
    private String variableNames[] = {"pi","e","x","y"};
```

```
32
33      protected Shell shlGraphWidget;
34      private Text txtEquationBox;
35
36      settingsWindow settingsWindowInstance = new settingsWindow();
37      //The copy of the settings used in the main window.
38      //This is sent to the settings window, when it is opened,
39      //and then copied back when the settings window is closed.
40      Settings mainSettings = new Settings();
41
42      /**
43       * Launch the application.
44       * @param args
45       */
46      public static void main(String[] args) {
47          try {
48              graph window = new graph();
49              window.open();
50          } catch (Exception e) {
51              e.printStackTrace();
52          }
53      }
54
55      /**
56       * Open the window.
57       */
58      public void open() {
59          Display display = Display.getDefault();
60          createContents();
61          shlGraphWidget.open();
62          shlGraphWidget.layout();
```

```
63          while (!shlGraphWidget.isDisposed()) {
64              if (!display.readAndDispatch()) {
65                  display.sleep();
66              }
67          }
68      }
69
70      /**
71       * Create contents of the window.
72       */
73      protected void createContents() {
74          //All this code is generated automatically by Eclipse through its GUI design page,
75          //with the exception of the listeners, which execute commands upon interaction with the
    GUI.
76          shlGraphWidget = new Shell();
77          shlGraphWidget.setSize(596, 249);
78          GridLayout gl_shlGraphWidget = new GridLayout();
79          gl_shlGraphWidget.numColumns = 7;
80          shlGraphWidget.setLayout(gl_shlGraphWidget);
81          shlGraphWidget.setText("Graph Widget");
82
83          txtEquationBox = new Text(shlGraphWidget, SWT.MULTI);
84          GridData gridData_txtEquationBox = new GridData();
85          gridData_txtEquationBox.widthHint = 107;
86          gridData_txtEquationBox.heightHint = 186;
87          gridData_txtEquationBox.horizontalAlignment = GridData.FILL;
88          gridData_txtEquationBox.verticalAlignment = GridData.FILL;
89          gridData_txtEquationBox.grabExcessHorizontalSpace = true;
90          gridData_txtEquationBox.grabExcessVerticalSpace = true;
91          gridData_txtEquationBox.horizontalSpan = 1;
92          txtEquationBox.setLayoutData(gridData_txtEquationBox);
```

```
 93
 94          Canvas canvas = new Canvas(shlGraphWidget, SWT.BORDER);
 95          canvas.setBackground(mainSettings.backgroundColor);
 96          GridData gridData_canvas = new GridData();
 97          gridData_canvas.horizontalSpan = 6;
 98          gridData_canvas.widthHint = 234;
 99          gridData_canvas.horizontalAlignment = GridData.FILL;
100          gridData_canvas.verticalAlignment = GridData.FILL;
101          gridData_canvas.grabExcessVerticalSpace = true;
102          gridData_canvas.grabExcessHorizontalSpace = true;
103          canvas.setLayoutData(gridData_canvas);
104
105          //This lets the user zoom and pan using the arrow keys
106          canvas.addKeyListener(new KeyAdapter()
107          {
108              public void keyPressed(KeyEvent e)
109              {
110                  switch (e.keyCode) {
111                      //Arrow keys to pan
112                      case SWT.ARROW_UP:
113                          mainSettings.yPan=mainSettings.yPan+mainSettings.panSpeed;
114                          canvas.redraw();
115                          break;
116                      case SWT.ARROW_DOWN:
117                          mainSettings.yPan=mainSettings.yPan-mainSettings.panSpeed;;
118                          canvas.redraw();
119                          break;
120                      case SWT.ARROW_LEFT:
121                          mainSettings.xPan=mainSettings.xPan+mainSettings.panSpeed;;
122                          canvas.redraw();
123                          break;
```

```
124                     case SWT.ARROW_RIGHT:
125                         mainSettings.xPan=mainSettings.xPan-mainSettings.panSpeed;;
126                         canvas.redraw();
127                         break;
128                     case 61://+ button
129                         //Zoom in
130                         mainSettings.xZoom = mainSettings.xZoom *
    Math.pow(mainSettings.zoomSpeed,mainSettings.zoomRatio);
131                         mainSettings.yZoom = mainSettings.yZoom * mainSettings.zoomSpeed;
132                         canvas.redraw();
133                         break;
134                     case 45://- button
135                         //Zoom out
136                         mainSettings.xZoom = mainSettings.xZoom /
    Math.pow(mainSettings.zoomSpeed,mainSettings.zoomRatio);
137                         mainSettings.yZoom = mainSettings.yZoom / mainSettings.zoomSpeed;
138                         if (mainSettings.debugMode) {
139                             System.out.println("xZoom is:"+mainSettings.xZoom+" yZoom
    is:"+mainSettings.yZoom);
140                         }
141                         canvas.redraw();
142                         break;
143                     case 93://[ button
144                         //Decrease the sensitivity by decreasing the threshold
145                         mainSettings.accuracyThreshold *= 1.1;
146                         canvas.redraw();
147                         break;
148                     case 91://] button
149                         //Increase the sensitivity by decreasing the threshold
150                         mainSettings.accuracyThreshold /= 1.1;
151                         canvas.redraw();
```

```
152                         break;
153                 }
154             }
155         });
156
157         //Button to redraw the graph
158         Button btnGraph = new Button(shlGraphWidget, SWT.NONE);
159         btnGraph.setLayoutData(new GridData(SWT.RIGHT, SWT.CENTER, false, false, 1, 1));
160         btnGraph.addSelectionListener(new SelectionAdapter() {
161             @Override
162             public void widgetSelected(SelectionEvent e) {
163                 //Store the equations in the text box
164                 mainSettings.currentEquations = txtEquationBox.getText();
165                 //Remove the spaces from the equations e.g. "y = x" -> "y=x"
166                 mainSettings.currentEquations = mainSettings.currentEquations.replace(" ", "");
167                 //Redraw the graph
168                 canvas.redraw();
169             }
170         });
171         btnGraph.setBounds(10, 255, 65, 28);
172         btnGraph.setText("Graph");
173
174         //Open settings
175         Button btnSettings = new Button(shlGraphWidget, SWT.NONE);
176         btnSettings.addSelectionListener(new SelectionAdapter() {
177             @Override
178             public void widgetSelected(SelectionEvent e) {
179                 mainSettings.currentEquations = txtEquationBox.getText();
180                 //Open the settings window
181                 settingsWindowInstance.open(mainSettings);
182                 //When it is closed, update the settings to match and redraw the graph with the new
```

```
    settings
183                 mainSettings = settingsWindowInstance.currentSettings;
184                 //Set the background colour
185                 canvas.setBackground(mainSettings.backgroundColor);
186                 //Remove empty lines from the equations list
187                 mainSettings.currentEquations = mainSettings.currentEquations.replace("\n\n",
    "\n");
188                 //Redraw the graph
189                 canvas.redraw();
190                 //Print the equations used back into the text box
191                 txtEquationBox.setText(mainSettings.currentEquations);
192             }
193         });
194         btnSettings.setBounds(143, 255, 76, 28);
195         btnSettings.setText("Settings");
196 
197         Button btnHelp = new Button(shlGraphWidget, SWT.NONE);
198         btnHelp.addSelectionListener(new SelectionAdapter() {
199             @Override
200             public void widgetSelected(SelectionEvent e) {
201                 Program.launch("https://sites.google.com/view/graph-widget/");
202             }
203         });
204         btnHelp.setText("Help");
205 
206         //Zoom into the graph
207         Button btnZoomIn = new Button(shlGraphWidget, SWT.NONE);
208         btnZoomIn.addSelectionListener(new SelectionAdapter() {
209             @Override
210             public void widgetSelected(SelectionEvent e) {
211                 mainSettings.xZoom = mainSettings.xZoom *
```

```
   Math.pow(mainSettings.zoomSpeed,mainSettings.zoomRatio);
212                 mainSettings.yZoom = mainSettings.yZoom * mainSettings.zoomSpeed;
213                 canvas.redraw();
214             }
215         });
216         btnZoomIn.setBounds(313, 255, 40, 28);
217         btnZoomIn.setText("+");
218 
219         //Zoom out of the graph
220         Button btnZoomOut = new Button(shlGraphWidget, SWT.NONE);
221         btnZoomOut.addSelectionListener(new SelectionAdapter() {
222             @Override
223             public void widgetSelected(SelectionEvent e) {
224                 mainSettings.xZoom = mainSettings.xZoom /
   Math.pow(mainSettings.zoomSpeed,mainSettings.zoomRatio);
225                 mainSettings.yZoom = mainSettings.yZoom / mainSettings.zoomSpeed;
226                 if (mainSettings.debugMode) {
227                     System.out.println("xZoom is:"+mainSettings.xZoom+" yZoom
   is:"+mainSettings.yZoom);
228                 }
229                 canvas.redraw();
230             }
231         });
232         btnZoomOut.setText("-");
233         btnZoomOut.setBounds(359, 255, 37, 28);
234 
235         //Reset the zoom and pan
236         Button btnCenterView = new Button(shlGraphWidget, SWT.NONE);
237         btnCenterView.addSelectionListener(new SelectionAdapter() {
238             @Override
239             public void widgetSelected(SelectionEvent e) {
```

```
240                 mainSettings.xZoom = 50;
241                 mainSettings.yZoom = 50;
242                 mainSettings.xPan = 0;
243                 mainSettings.yPan = 0;
244                 canvas.redraw();
245             }
246         });
247         btnCenterView.setText("Center view");
248         btnCenterView.setLayoutData(new GridData(SWT.RIGHT, SWT.CENTER, false, false, 1, 1));
249 
250         //Reset the sensitivity
251         Button btnResetSensitivity = new Button(shlGraphWidget, SWT.NONE);
252         btnResetSensitivity.addSelectionListener(new SelectionAdapter() {
253             @Override
254             public void widgetSelected(SelectionEvent e) {
255                 mainSettings.accuracyThreshold = 3;
256                 canvas.redraw();
257             }
258         });
259         btnResetSensitivity.setText("Reset Sensitivity");
260 
261         //This runs when the canvas is redrawn, for example, when the "graph" button is pressed.
262         canvas.addPaintListener(new PaintListener() {
263               public void paintControl(PaintEvent onGraph) {
264             //This object stores the height and width of the canvas.
265             Rectangle rect = ((Canvas) onGraph.widget).getBounds();
266 
267             //Set the colours for the background and foreground
268             onGraph.gc.setForeground(mainSettings.penColor);
269             onGraph.gc.setBackground(mainSettings.backgroundColor);
270 
```

```
271             //Draws the x and y axes, if the settings say they should be drawn.
272             if (mainSettings.showAxes) {
273                 onGraph.gc.drawLine(mainSettings.xPan+rect.width/2, 0, mainSettings.xPan
    +rect.width/2, rect.height);
274                     onGraph.gc.drawLine(0, mainSettings.yPan+rect.height/2, rect.width,
    mainSettings.yPan+rect.height/2);
275                 }
276 
277             //Gets the equations from the text box and splits them at each newline into an array.
278                 String equations[] = mainSettings.currentEquations.split("\\r?\\n");
279 
280                 //Runs all the unit tests on the methods found in the graph and node classes.
281                 if (mainSettings.debugMode) {
282                     testGraphMethods();
283                 }
284 
285                 String tokenisedEquation;   //Stores the tokenised form of the current equation.
286                 Node parsedEquation;//Stores the parsed tree representation of the current
    equation.
287 
288                 boolean containsInvalidEquation=false;  //True if there is at lease one invalid
    equation in the list, in which case a warning is displayed.
289 
290                 //Loops through all the equations and points and checks if each point satisfies
    each equation.
291                 for (int equationIndex = 0; equationIndex <
    Array.getLength(equations);equationIndex++) {
292                     //Make sure the line isn't a comment; if it isn't, try to graph it
293                     if (!equations[equationIndex].startsWith("//")) {
294                         String currentEquation = equations[equationIndex];
295 
```

```
296                         //Turn all functions of x into equations in x & y
297                         //This allows expressions like sin(x) to be interpreted as y=sin(x)
298                         if (!(currentEquation.contains("="))){
299                             currentEquation="y="+currentEquation;
300                         }
301 
302                         //Check if the equation is valid and if so, graphs it on the canvas
303                         if (isValidExpression(currentEquation)) {
304                             //Tokenises the equation
305                             tokenisedEquation = tokenise(currentEquation);
306                             try {
307                                 //Turns the equation into an expression tree
308                                 parsedEquation = parse(tokenisedEquation);
309                                 //Set the background
310                                 onGraph.gc.setBackground(mainSettings.backgroundColor);
311 
312                                 //Loop through every pixel of the canvas and colour the pixel in if
    it matches the equation
313                                 for (int xCoord = 0; xCoord < canvas.getBounds().width; xCoord++) {
314                                     for (int yCoord = 0; yCoord < canvas.getBounds().height; yCoord
    ++) {
315                                         //The coordinates need to be adjusted based on zoom and pan
316                                         double adjustedX = (xCoord-rect.width/2-mainSettings.xPan)/
    mainSettings.xZoom;
317                                         double adjustedY = (-yCoord+rect.height/
    2+mainSettings.yPan)/mainSettings.yZoom;
318 
319                                         //The coordinates are checked against the equation and then
    plotted if they match
320                                         try {
321                                             if
```

```
    (substitute(adjustedX,adjustedY,parsedEquation).boolValue) {
322                                                     onGraph.gc.drawPoint(xCoord, yCoord);
323                                                 }
324                                             } catch (Exception e){
325                                                 if (mainSettings.debugMode) {
326                                                     System.out.println("Substitution failure on
    x="+xCoord+", y="+yCoord+", "+currentEquation);
327                                                 }
328                                             }
329                                         }
330                                     }
331                                 } catch (Exception e) {
332                                     containsInvalidEquation = true;
333                                     if (mainSettings.debugMode) {
334                                         System.out.println("Parse failure on "+currentEquation);
335                                     }
336                                 }
337                             } else {
338                                 containsInvalidEquation = true;
339                             }
340                         }
341                 }
342                 //If there is an invalid equation, display a warning
343                 if (containsInvalidEquation && !mainSettings.currentEquations.isBlank()) {
344                     MessageBox warningText = new MessageBox(shlGraphWidget);
345                     warningText.setMessage("One or more of your equations/expressions is invalid.
    Please see the help document for more information.");
346                     warningText.open();
347                 }
348             }
349         });
```

```
350     }
351
352     public boolean isValidExpression(String equationString) {
353         boolean returnValue = true;
354
355         //Check if the string is empty
356         if (equationString.length()==0) {
357             returnValue = false;
358         }
359
360         //Check for illegal characters by comparing each character of string to each element of
    reservedChars
361         boolean isLegalChar;
362         for (int charIndex = 0; charIndex < equationString.length(); charIndex++) {
363             isLegalChar = true;
364             for (int illegalCharIndex = 0; illegalCharIndex < Array.getLength(reservedChars);
    illegalCharIndex++) {
365                 if (reservedChars[illegalCharIndex]==equationString.charAt(charIndex)) {
366                     isLegalChar = false;
367                 }
368             }
369             if (!isLegalChar) {
370                 returnValue = false;
371             }
372         }
373
374         //Check that brackets are paired by comparing the number of '(' and ')' brackets
375         int bracketCheckSum = 0;
376         for (int charIndex = 0;charIndex < equationString.length();charIndex++) {
377             switch(equationString.charAt(charIndex)) {
378               case '(':
```

```
379                 bracketCheckSum++;
380                 break;
381               case ')':
382                 bracketCheckSum--;
383                 break;
384             }
385             //If a ')' is found without a pair, e.g. in "())", then brackets are not paired.
386             if (bracketCheckSum < 0) {
387                 returnValue = false;
388             }
389         }
390 
391         //If the number of brackets of each type are not equal, then brackets are not paired.
392         if (!(bracketCheckSum == 0)){
393             returnValue = false;
394         }
395 
396         if (mainSettings.debugMode) {
397             System.out.println("isValidExpression returns " + Boolean.toString(returnValue) + " on
    "+ equationString);
398         }
399         return returnValue;
400     }
401     private String tokenise(String equationString) {
402         if (mainSettings.debugMode) {
403             System.out.println("Entered tokenise");
404         }
405 
406         //Make the equation lowercase for ease of comparison with tokens
407         String lowercaseEquation = equationString.toLowerCase();
408 
```

```
409         //Loop through the string and compare portions of it with each of the function and variable
    names
410         String subString;   //Stores a section of the equation string
411         char previousChar = ' ';//Stores the character before the current one
412         int charIndex = 0;  //Stores the index of the current character
413         boolean exitLoop = false;   //Will cause the loops through the token names to be terminated
    if set to true
414         while (charIndex < lowercaseEquation.length()) {//Note: the length of newString will likely
    change during the loop
415             if (!Character.isLetter(previousChar)) {//Check if the previous character is a letter
416                 //Compare a part of the string at and after the current character with the name of
    each function
417                 exitLoop = false;
418                 int functionIndex = 0;
419                 do {
420                     subString = lowercaseEquation.substring(charIndex,Math.min(charIndex
    +functionNames[functionIndex].length(),lowercaseEquation.length()));
421                     if (subString.equals(functionNames[functionIndex])) {
422                         lowercaseEquation = lowercaseEquation.substring(0,charIndex) + "{" +
    functionNames[functionIndex] + "}" + lowercaseEquation.substring(charIndex
    +functionNames[functionIndex].length());
423                         charIndex = charIndex + subString.length() + 1;
424                         exitLoop = true;
425                     }
426                     functionIndex++;
427                 } while (!exitLoop && functionIndex < functionNames.length);
428 
429                 //Compare a part of the string at and after the current character with the name of
    each variable (and constants e.g. pi, e)
430                 exitLoop = false;
431                 int variableIndex = 0;
```

```
432                 do {
433                     subString = lowercaseEquation.substring(charIndex,Math.min(charIndex
    +variableNames[variableIndex].length(),lowercaseEquation.length()));
434                     if (subString.equals(variableNames[variableIndex])) {
435                         lowercaseEquation = lowercaseEquation.substring(0,charIndex) + "[" +
    variableNames[variableIndex] + "]" + lowercaseEquation.substring(charIndex
    +variableNames[variableIndex].length());
436                         charIndex = charIndex + subString.length() + 1;
437                         exitLoop = true;
438                     }
439                     variableIndex++;
440                 } while (!exitLoop && variableIndex < variableNames.length);
441             }
442             charIndex++;
443         }
444 
445         //Regular expression to insert multiplication symbols between brackets and numbers where
    appropriate, e.g.:
446         // "...)[..." -> "...)*[..."  "...2(..." -> "...2*(..."
447         lowercaseEquation = lowercaseEquation.replaceAll("([\\)\\][0-9])([\\(\\[\\{])", "$1*$2");
448 
449         if (mainSettings.debugMode) {
450             System.out.println("tokenise returns " + lowercaseEquation + " on "+ equationString);
451         }
452         return lowercaseEquation;
453     }
454 
455     //If the parse function is given a string, put it in a node and then call the parse function
    with a node.
456     private Node parse(String tokenisedExpression) {
457         Node nodeToParse = new Node(tokenisedExpression);
```

```
458         return parse(nodeToParse);
459     }
460 
461     //Parses a node containing a tokenised expression into an expression tree, which can then be easily evaluated by the substitute function
462     private Node parse(Node tokenisedNode){
463 
464         //The output of the parse function; it is "current" because the parse function is recursive.
465         Node currentNode = tokenisedNode;
466 
467         //If an expression begins with a '-', such as in "-2", make it start with "0-", e.g. "0-2".
468         if (currentNode.contents.charAt(0)=='-') {
469             currentNode.contents = '0'+currentNode.contents;
470         }
471 
472         //Search for an '=' character in the string. Returns -1 if not found.
473         int indexInString = -1;
474         for (int currentIndex=0;currentIndex<currentNode.contents.length();currentIndex++) {
475             if (currentNode.contents.charAt(currentIndex)=='=') {
476                 indexInString = currentIndex;
477             }
478         }
479 
480         //If an '=' character is found, split the string there and create two new child nodes to represent the LHS and RHS of the equation.
481         //These are to be parsed.
482         if (!(indexInString == -1)) {
483             Node lhs = new Node(currentNode.contents.substring(0, indexInString));
484             currentNode.newChild(parse(lhs));
485             Node rhs = new Node(currentNode.contents.substring(indexInString+1));
```

```
486             currentNode.newChild(parse(rhs));
487             currentNode.contents = "=";
488         } else {//If the '=' character is not found (the more common outcome), continue with the next part of the parse function.
489             //Firstly, remove any brackets that enclose the entire string, e.g. "(2)" -> "2". This is done in a similar manner to how
490             // the isValidExpression function checks that brackets are paired.
491             boolean exitLoop = false;
492             int bracketCheckSum;
493             int charIndex;
494             while (currentNode.contents.startsWith("(") && !exitLoop) {
495                 charIndex = 1;
496                 bracketCheckSum = 1;
497                 while ((charIndex <= currentNode.contents.length()) && !(bracketCheckSum==0)) {
498                     switch (currentNode.contents.charAt(charIndex)) {
499                         case ')':
500                             bracketCheckSum--;
501                             break;
502                         case '(':
503                             bracketCheckSum++;
504                             break;
505                     }
506                     charIndex++;
507                 }
508                 if (charIndex == currentNode.contents.length()) {
509                     currentNode.contents = currentNode.contents.substring(1,currentNode.contents.length()-1);
510                 } else {
511                     exitLoop = true;
512                 }
513             }
```

```
514
515             //Check if the string is enclosed by a function token, e.g. "{sin}(2)",
516             // and then create an expression tree with the function token as the root node.
517             //Firstly, the string must begin with '{' and end with ')'.
518             boolean exitParse = false;
519             if (currentNode.contents.startsWith("{") && currentNode.contents.endsWith(")")) {
520
521                 charIndex = 0;
522                 while (!(currentNode.contents.charAt(charIndex)=='}')) {
523                     charIndex++;
524                 }
525
526                 //Secondly, it must not be of the form "{...}(...)...(...)"; that is,
527                 // the last bracket must be paired with the bracket after the function token.
528                 // An example of a string not satisfying this is "{sin}(2)*(1+1)".
529                 int braceIndex = charIndex;
530                 charIndex++;
531                 bracketCheckSum = 1;
532                 while (!(bracketCheckSum==0)) {
533                     charIndex++;
534                     switch (currentNode.contents.charAt(charIndex)) {
535                         case '(':
536                             bracketCheckSum++;
537                             break;
538                         case ')':
539                             bracketCheckSum--;
540                             break;
541                     }
542                 }
543
544                 //Create the tree with the function token as the root node.
```

```
545                 if (charIndex == currentNode.contents.length()-1) {
546                     currentNode.newChild(parse(currentNode.contents.substring(braceIndex+1)));
547                     currentNode.contents = currentNode.contents.substring(0,braceIndex+1);
548                     exitParse = true;
549                 }
550             }
551 
552             //If no tokens have been dealt with so far, check for infix-style operators such as +, -, /, *.
553             if (exitParse==false) {
554                 int operatorIndex = 0;
555                 bracketCheckSum = 0;
556                 //The operators should be checked in reverse order of operations. This should be reflected in the operatorList array.
557                 while (!exitParse && (operatorIndex < operatorList.length)) {
558                     charIndex = currentNode.contents.length()-1;
559                     while ((charIndex >= 0) && !exitParse) {
560                         switch (currentNode.contents.charAt(charIndex)) {
561                             case '(':
562                                 bracketCheckSum++;
563                                 break;
564                             case ')':
565                                 bracketCheckSum--;
566                                 break;
567                         }
568                         //The operator must not be enclosed by any brackets. If it is to be parsed, it becomes the root node
569                         // of a tree with two children representing each of its arguments, e.g. "1+2" would have root "+" and children "1" and "2".
570                         // The children should also be parsed.
571                         if ((bracketCheckSum == 0) && (currentNode.contents.charAt(charIndex) ==
```

```
    operatorList[operatorIndex])) {
572                             Node leftArg = parse(currentNode.contents.substring(0,charIndex));
573                             if (mainSettings.debugMode) {
574                                 System.out.println("Parsing "+leftArg.contents);
575                             }
576                             currentNode.newChild(leftArg);
577                             Node rightArg = parse(currentNode.contents.substring(charIndex+1));
578                             if (mainSettings.debugMode) {
579                                 System.out.println("Parsing "+rightArg.contents);
580                             }
581                             currentNode.newChild(rightArg);
582 
583                             currentNode.contents =
    Character.toString(currentNode.contents.charAt(charIndex));
584                             exitParse = true;
585                         }
586                         charIndex--;
587                     }
588                     operatorIndex++;
589                 }
590             }
591         }
592 
593         //If no tokens are detected, the input and output are the same.
594         // For example, parsing a node containing "2" should just return a node containing "2".
595         if (mainSettings.debugMode) {
596             System.out.println("parse returns " + currentNode.contents + " as parent on " +
    tokenisedNode.contents);
597         }
598         return currentNode;
599     }
```

```
600
601     //Substitute coordinates into an expression tree. If the tree has a '=' as its root, then the
    output will have a boolean value of "true",
602     // otherwise it will have a numerical value representing the value of the expression at the two
    coordinates.
603     private ReturnValues substitute(double xCoord, double yCoord, Node expressionTree) {
604         //This output has both a boolean and numerical value, as both are acceptable outputs of
    this function.
605         ReturnValues output = new ReturnValues();
606         if (expressionTree.contents == "=") {
607             double lhs = substitute(xCoord,yCoord,expressionTree.children.get(0)).numValue;
608             double rhs = substitute(xCoord,yCoord,expressionTree.children.get(1)).numValue;
609             if (Math.abs(lhs-rhs)<2*mainSettings.accuracyThreshold/(mainSettings.xZoom
    +mainSettings.yZoom)) {
610                 output.boolValue = true;
611             } else {
612                 output.boolValue = false;
613             }
614         } else {
615             double args[] = new double[expressionTree.getNumChildren()];
616             for (int childrenIndex = 0; childrenIndex < Array.getLength(args); childrenIndex++) {
617                 args[childrenIndex] =
    substitute(xCoord,yCoord,expressionTree.children.get(childrenIndex)).numValue;
618             }
619             switch (expressionTree.contents) {
620                 case "[x]":
621                     output.numValue = xCoord;
622                     break;
623                 case "[y]":
624                     output.numValue = yCoord;
625                     break;
```

```
626                 /*case "[t]":
627                     output.numValue =
628                     break;*/
629                 case "[e]":
630                     output.numValue = Math.E;
631                     break;
632                 case "[pi]":
633                     output.numValue = Math.PI;
634                     break;
635                 case "{sin}":
636                     output.numValue = Math.sin(args[0]);
637                     break;
638                 case "{cos}":
639                     output.numValue = Math.cos(args[0]);
640                     break;
641                 case "{tan}":
642                     output.numValue = Math.tan(args[0]);
643                     break;
644                 case "{abs}":
645                     output.numValue = Math.abs(args[0]);
646                     break;
647                 case "{ln}":
648                     output.numValue = Math.log(args[0]);
649                     break;
650                 case "+":
651                     output.numValue = args[0]+args[1];
652                     break;
653                 case "-":
654                     output.numValue = args[0]-args[1];
655                     break;
656                 case "/":
```

```
657                     output.numValue = args[0]/args[1];
658                     break;
659                 case "*":
660                     output.numValue = args[0]*args[1];
661                     break;
662                 case "^":
663                     output.numValue = Math.pow(args[0],args[1]);
664                     break;
665                 default:
666                     output.numValue = Double.valueOf(expressionTree.contents);
667                     break;
668                 }
669         }
670         return output;
671     }
672     private void testGraphMethods() {
673         //TEST isValidExpression
674         assert isValidExpression("x=y");
675         assert isValidExpression("y=sin(x)");
676         assert isValidExpression("x^2");
677         assert !isValidExpression("x=(y");
678         assert !isValidExpression("x=)(y");
679         assert !isValidExpression("y=[x]");
680         assert !isValidExpression("y={x}");
681 
682         //TEST tokenise
683         assert tokenise("x=y")=="[x]=[y]";
684         assert tokenise("y=sin(x)")=="[y]={sin}([x])";
685         assert tokenise("x^2")=="[x]^2";
686 
687         //TEST parse
```

```
688         assert parse("[y]=[x]").contents=="=";
689         assert parse("[y]+[x]").contents=="+";
690 
691         //TEST substitute
692         assert substitute(12,12,parse("x=y")).boolValue;
693         assert substitute(0,0,parse("1=1")).boolValue;
694         assert substitute(0,0,parse("1+1=2")).boolValue;
695         assert substitute(5,7,parse("x+y")).numValue==12;
696         assert !substitute(0,0,parse("2+2=5")).boolValue;
697         assert !substitute(200,-200,parse("x=y")).boolValue;
698 
699         System.out.println("All tests passed!");
700     }
701 }
702 
703 class Node {
704     String contents;
705     ArrayList<Node> children = new ArrayList<Node>();
706 
707     //Make a node containing newContents
708     public Node(String newContents){
709         contents = newContents;
710     }
711 
712     //Add a child to the node from a string
713     public void newChild(String newContents) {
714         Node childNode = new Node(newContents);
715         children.add(childNode);
716     }
717 
718     //Add a child to the node from a node
```

```
719     public void newChild(Node childNode) {
720         children.add(childNode);
721     }
722 
723     //Get the number of children of the node
724     public int getNumChildren() {
725         return children.size();
726     }
727 }
728 
729 class ReturnValues {
730     boolean isVoid = false;
731     double numValue;
732     boolean boolValue;
733 }
```