8/29/2016

Name: Daniel Field

Student ID: 3107593512

Arithmetic Challenge Game

Advanced Data Structures and Algorithms

Contents

[Introduction 6](#_Toc460245342)

[Main.java 7](#_Toc460245343)

[Equation (Equation.java) 7](#_Toc460245344)

[Algorithms 7](#_Toc460245345)

[Constructor (datetime specified) 7](#_Toc460245346)

[Constructor (datetime NOT Specified) 8](#_Toc460245347)

[isCorrect 8](#_Toc460245348)

[toStringArray 8](#_Toc460245349)

[GetDayOfMonth 8](#_Toc460245350)

[Log (Log.java) 8](#_Toc460245351)

[Algorithms 8](#_Toc460245352)

[CreateExceptionLog 8](#_Toc460245353)

[AppendExceptionLog 9](#_Toc460245354)

[createArithmeticLog 9](#_Toc460245355)

[AppendArithmeticLog 9](#_Toc460245356)

[ImportArithmeticLog 9](#_Toc460245357)

[Instructor and Student Selection Screen (MainMenu.java) 10](#_Toc460245358)

[IPO Chart 10](#_Toc460245359)

[Algorithms 10](#_Toc460245360)

[main 11](#_Toc460245361)

[Constructor 11](#_Toc460245362)

[InitialiseComponents 11](#_Toc460245363)

[ActionPerformed 11](#_Toc460245364)

[Instructor Screen (Instructor.java) 13](#_Toc460245365)

[IPO Chart 13](#_Toc460245366)

[Algorithms 14](#_Toc460245367)

[Constructor 14](#_Toc460245368)

[initialiseComponents 15](#_Toc460245369)

[radPreOrder\_ItemStateChanged 16](#_Toc460245370)

[radInOrder\_ItemStateChanged 16](#_Toc460245371)

[radPostOrder\_ItemStateChanged 16](#_Toc460245372)

[displayHashSet 16](#_Toc460245373)

[ActionPerformed 16](#_Toc460245374)

[paintComponent 18](#_Toc460245375)

[drawStyledBox 18](#_Toc460245376)

[Connect 18](#_Toc460245377)

[Open 18](#_Toc460245378)

[Close 19](#_Toc460245379)

[InstructorHandle 19](#_Toc460245380)

[Run 19](#_Toc460245381)

[start 19](#_Toc460245382)

[stop 19](#_Toc460245383)

[findClient 20](#_Toc460245384)

[serverHandle 20](#_Toc460245385)

[remove 20](#_Toc460245386)

[addThread 20](#_Toc460245387)

[Student screen(Student.java) 20](#_Toc460245388)

[IPO Chart 21](#_Toc460245389)

[Algorithms 21](#_Toc460245390)

[Constructor 21](#_Toc460245391)

[initialiseComponents 21](#_Toc460245392)

[paintComponent 21](#_Toc460245393)

[actionPerformed 22](#_Toc460245394)

[Connect 22](#_Toc460245395)

[send 22](#_Toc460245396)

[Handle 22](#_Toc460245397)

[Open 22](#_Toc460245398)

[Close 23](#_Toc460245399)

[stop 23](#_Toc460245400)

[Sorting (Sort.java) 23](#_Toc460245401)

[algorithm Comparison 23](#_Toc460245402)

[testing 23](#_Toc460245403)

[Algorithm details 25](#_Toc460245404)

[Comparator 27](#_Toc460245405)

[JFreeChart (Third Party) (chart.java) 28](#_Toc460245406)

[Algorithms 28](#_Toc460245407)

[Constructor 28](#_Toc460245408)

[paint 28](#_Toc460245409)

[DisplayLineChart 28](#_Toc460245410)

[Utilities (Utilities.java) 29](#_Toc460245411)

[Algorithms 29](#_Toc460245412)

[CreateRadioButton 29](#_Toc460245413)

[stripzero 29](#_Toc460245414)

[Binary Tree (BinaryTree.java) 30](#_Toc460245415)

[BinaryTree Algorithms 30](#_Toc460245416)

[AddLeaves 30](#_Toc460245417)

[AddLeaf 30](#_Toc460245418)

[InOrderTraverseTree 31](#_Toc460245419)

[PreOrderTraverseTree 31](#_Toc460245420)

[PostOrderTraverseTree 31](#_Toc460245421)

[GetEquationAndStripZeroes 31](#_Toc460245422)

[getTreeAsString 31](#_Toc460245423)

[findLeaf 32](#_Toc460245424)

[Leaf (node) Algorithms 32](#_Toc460245425)

[Constructor 32](#_Toc460245426)

[DLinkedEquationList (DLinkedEquationList.java) 33](#_Toc460245427)

[Algorithms 33](#_Toc460245428)

[Constructor (no parameters) 33](#_Toc460245429)

[Constructor (Equation specified) 33](#_Toc460245430)

[Constructor (float, char, float, float) 33](#_Toc460245431)

[get 33](#_Toc460245432)

[getLength 33](#_Toc460245433)

[displayAsString 34](#_Toc460245434)

[ListNode (ListNode.java) 34](#_Toc460245435)

[Algorithms 34](#_Toc460245436)

[Constructor (No parameters) 34](#_Toc460245437)

[Constructor (Equation eq) 34](#_Toc460245438)

[Constructor (float operand one, operator, operand two, answer) 34](#_Toc460245439)

[Append 35](#_Toc460245440)

[Insert 35](#_Toc460245441)

[Remove 35](#_Toc460245442)

[toString (Override) 35](#_Toc460245443)

[TableModel (TableModel.java) 36](#_Toc460245444)

[Algorithms 36](#_Toc460245445)

[Constructor 36](#_Toc460245446)

[getRowCount 36](#_Toc460245447)

[getColumnCount 36](#_Toc460245448)

[getValueAt 36](#_Toc460245449)

[getColumnName 36](#_Toc460245450)

[Add 36](#_Toc460245451)

[InstructorThread (InstructorThread.java) 37](#_Toc460245452)

[Algorithms 37](#_Toc460245453)

[Constructor 37](#_Toc460245454)

[Open 37](#_Toc460245455)

[close 37](#_Toc460245456)

[Run (override) 37](#_Toc460245457)

[ServerThread (ServerThread.java) 38](#_Toc460245458)

[Algorithms 38](#_Toc460245459)

[Constructor 38](#_Toc460245460)

[Send 38](#_Toc460245461)

[run (override) 38](#_Toc460245462)

[open 38](#_Toc460245463)

[close 38](#_Toc460245464)

[StudentThread (StudentThread.java) 39](#_Toc460245465)

[Algorithms 39](#_Toc460245466)

[Constructor 39](#_Toc460245467)

[Open 39](#_Toc460245468)

[close 39](#_Toc460245469)

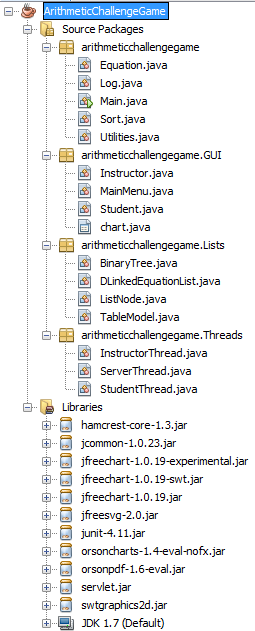
[Run (override) 39](#_Toc460245470)

[References 40](#_Toc460245471)

# Introduction

This is the technical documentation for the arithmetic challenge game, which is a network-based game where there is an instructor, and a student. The instructor sends an equation to the student to answer, and the student sends back the answer. If the answer given is correct, the instructor can send a new equation. If the answer is incorrect, the student is given another try to answer the equation, while the instructor stores the incorrectly answered equation in a linked list.

There are 16 java files used, and a number of jar libraries used. Here is the complete list of files included in the project:



These files are separated into separate packages to make it easier to navigate to the different files. All of the files that contain code to be visually displayed in JFrames, are stored in the “GUI” package. All of the files that are used for creating lists are stored in the “Lists” package.

The “.jar” files at the bottom are third party files used for creating charts. See the “JFreeChart” section for more information regarding the charting libraries.

# Main.java

Main.java is the file that runs first, and is used to call the main menu frame. It also contains the application’s name in a final string.

The reason behind using this file is to make it easier to start the programme. Due to the packages in the project, it would be harder to have to navigate to the correct package manually to find the main menu file, so the reasoning behind the use of this file was primarily for user friendliness.

Inside the main method, it first sets the look and feel to “nimbus”, which makes the program look prettier; it then declares and draws a JFrame which contains the MainMenu JPanel; and lastly it creates the exception log and the arithmetic log.

# Equation (Equation.java)

The equation class contains the following properties:

* DateTime (String)
* OperandOne (float)
* Operator (char)
* OperandTwo (float)
* AnswerGiven (float)
* CorrectAnswer (float)

Each contain methods for getting and setting the variables.

## Algorithms

### Constructor (datetime specified)

1. Set operandOne to the specified value
2. Set operator to the specified value
3. Set operandTwo to the specified value
4. Set answerGiven to the specified value
5. Set dateTime to the specified value
6. Set correct answer to the return value of Utilities.calculateAnswer(operandOne, operator, operandTwo)

### Constructor (datetime NOT Specified)

1. Set operandOne to the specified value
2. Set operator to the specified value
3. Set operandTwo to the specified value
4. Set answerGiven to the specified value
5. Set dateTime to the current date & time
6. Set correct answer to the return value of Utilities.calculateAnswer(operandOne, operator, operandTwo)

### isCorrect

Return true if answer given is correct, else false

### toStringArray

Return the equation as a string array, with unnecessary zeroes stripped from the number

### GetDayOfMonth

Returns the day of the month from the dateTime property.

# Log (Log.java)

The log class is used for logging arithmetic, as well is logging exceptions. It contains two final strings:



These strings determine from where the log data is read, and to where it is written.

## Algorithms

### CreateExceptionLog

1. Create File object with the string EXCEPTION\_LOG
2. If the file doesn’t exist
   1. Get the parent file and make the directories
   2. Create a new file
3. Return true
4. If an exception is thrown, return false

### AppendExceptionLog

The logging format is:

[<Date> <Time>][<Exception class name>]

Message: <Exception message>

Additional Info: <Custom information (User friendly info)>

1. Instantiate a new FileWriter object
2. Append file with specified data
3. Close FileWriter object
4. Return true
5. If there is an error, return false

### createArithmeticLog

1. Create File object with the string ARITHMETIC\_LOG
2. If the file doesn’t exist
   1. Get the parent file and make the directories
   2. Create a new file
3. Return true
4. If an exception is thrown, return false

### AppendArithmeticLog

The format for the arithmetic log is:

[<Date> <Time>] <Equation>

There is a space after the tab in between the DateTime and the equation because it makes it easier to substring.

1. Instantiate a new FileWriter object
2. Append file with specified data
3. Close FileWriter object
4. Return true
5. If there is an error, return false

### ImportArithmeticLog

1. Create array list of equations called “array”
2. Create a BufferedReader object
3. Initialize string “line”
4. While there is another line to be read
   1. Set line to the current line being read
   2. Split the line into temporary string array
   3. Create equation object
   4. Pass temporary array data into the equation constructor
   5. Add the equation to the array list
5. Return the array list

# Instructor and Student Selection Screen (MainMenu.java)

## IPO Chart

|  |  |  |
| --- | --- | --- |
| **Input** | **Process** | **Output** |
| Host name text field | Stores the text in RAM | Displays the text |
| Port number spinner | Changes the number depending on what you typed or depending on whether you clicked up or down. | Displays the value |
| 'Host' button | Gets the local host and port number from the fields above and passes it into the instructor constructor. | Displays the instructor frame.  Displays information regarding the status of the server on the console. |
| 'Join' button | Gets the host name and port number from the fields above and passes it into the student constructor. | Displays the client student frame |
| 'Exit' button | Terminates the currently running Java Virtual Machine. | Closes the window. |

## Algorithms

### main

1. Set look and feel to “com.sun.java.swing.plaf.nimbus.NimbusLookAndFeel”
2. Instantiate JFrame
3. Set the default close operation to exit on close
4. Set the title to “Arithmetic challenge game”
5. Set visible
6. Set resizable to false
7. Set size to 265 x 140
8. Centre the frame
9. Create arithmetic and exception logs, if necessary

### Constructor

1. Set layout to null
2. Call InitialiseComponents method
3. Call addComponents method

### InitialiseComponents

1. Create a JSpin­ner to allow the user to choose the port number
2. Create a text field to allow the user to set the host name
3. Create labels
4. Create host button
5. Create join button
6. Create exit button

### ActionPerformed

1. If action is btnHost
   1. If fields are not empty:
      1. Store the field values in RAM
      2. Initialise instructor frame
      3. Pass field values into instructor constructor
      4. Draw frame
      5. Centre frame
   2. If fields are empty:
      1. Set host text to the PC’s host name.
      2. Throw NullPointerException to catch and store in exception log
   3. Catch exceptions and append the exception log with custom message.
   4. Display custom message in message dialog
2. If action is btnJoin
   1. If fields are not empty
      1. Get the values and store in RAM
      2. Create Student JFrame
      3. Centre the frame
   2. If fields are empty
      1. Set host text to the PC’s host name
      2. Throw NullPointerException to catch and store in exception log
   3. Catch exceptions and append the exception log with custom message.
   4. Display custom message in message dialog
3. If action btnExit:
   1. Terminate the currently running virtual machine.

# Instructor Screen (Instructor.java)

## IPO Chart

|  |  |  |
| --- | --- | --- |
| **Input** | **Process** | **Output** |
| Operand one text field | Stores the text in RAM | Displays the text |
| Operator drop down list | Stores the selection in RAM | Displays the selection |
| Operand two text field | Stores the text in RAM | Displays the text |
| Answer text field | Calculates the text from the first text field and the second text field after converting them into integers and uses the selected operator. | Displays the answer in the text field. |
| 'Send' button | 1. Stores the equation in an equation object and puts it in the table. 2. Puts the equation in a binary tree 3. Appends arithmetic log 4. Clears text fields 5. Sends the equation to the socket. 6. Disable send button | 1. Displays the equation in the JTable. 2. Displays the equation in a binary tree displayed as text in a text area. |
| 'Bubble Sort' button | Sort by answer using the bubble sort algorithm in asc/desc order (toggles asc/desc) | Displays the sorted array list in the table. |
| 'Insertion Sort' button | Sort by answer using the insertion sort algorithm in asc/desc order (toggles asc/desc) | Displays the sorted array list in the table. |
| 'Selection Sort' button | Sort by answer using the selection sort algorithm in asc/desc order (toggles asc/desc) | Displays the sorted array list in the table. |
|  |  |  |
| Binary tree text area | Stores the text in RAM | Displays the text |
| Pre-order traverse tree | Does a pre-order traversal of the tree. | Displays the tree data. |
| In-order traverse tree | Does an in-order traversal of the tree | Displays the tree data. |
| Post-order traverse tree | Does a post-order traversal of the tree | Displays the tree data. |
| Save button (binary tree) | Gets the data from the text area and prepares to put it into a file. The file name depends on which radio button is selected. | Writes the data to the file, creating the file if it does not yet exist, else it will overwrite it. |
| Display Line Chart button | Generate a line chart with the 10 bottom-most equations from the array list | Display in a new JFrame |

## Algorithms

### Constructor

* 1. Store port number in class variable
  2. Set layout to null
  3. Import the arithmetic log
  4. Store in array list
  5. Add array list items to binary tree
  6. Add equations to a HashSet
  7. Print HashSet data
  8. Call the initialiseComponents method
  9. Call addComponents
  10. Instantiate server socket with specified port no.
  11. Start the server.
  12. Connect to the server

### initialiseComponents

This method is to define and draw the components on the frame. This has been done to reduce the amount of code in the constructor method.

The method returns true if successful so that if necessary, code could be written to deal with the issue where the components didn’t initialise correctly.

|  |  |
| --- | --- |
| Equation form | * + 1. Create label which displays information to instruct the user.     2. Create text fields to store the operands, and the answer     3. Create operator combo box (drop-down list)     4. Create send button     5. Add components to frame |
| Arithmetic history table | * + 1. Instantiate TableModel class, passing in an array list of equations.     2. Create a scroll pane and add it to the frame     3. Create table and put it in the scroll pane     4. Set the table border to a solid blue line |
| Sorting buttons | * + 1. Create bubble sort button     2. Create Insertion sort button     3. Create selection sort button     4. Add buttons to frame |
| Doubly linked list | * + 1. Create text area to display the list     2. Create a scroll pane     3. Put text area in scroll pane     4. Add scroll pane to frame |
| Binary tree | * + 1. Create text area     2. Create scroll pane     3. Put text area in scroll pane     4. Create save button     5. Create radio buttons to change ordering (pre-order, in-order, post-order)     6. Put radio buttons in a button group |
| Status bar | * + 1. Create status label     2. Add label to frame |
| Line Chart button | * + 1. Create line chart button     2. Display line chart button |

### radPreOrder\_ItemStateChanged

1. Return a new Callable<Integer>
   1. Override call method with:
      1. Call the method to traverse the binary tree in pre-order
      2. Set the text of the binary tree text area to display the pre-ordered traversal
      3. Return 0

### radInOrder\_ItemStateChanged

1. Return a new Callable<Integer>
   1. Override call method with:
      1. Call the method to traverse the binary tree in order
      2. Set the text of the binary tree text area to display the in-order traversal
      3. Return 0

### radPostOrder\_ItemStateChanged

1. Return a new Callable<Integer>
   1. Override call method with:
      1. Call the method to traverse the binary tree in post-order
      2. Set the text of the binary tree text area to display the post-ordered traversal
      3. Return 0

### displayHashSet

1. Print “Displaying contents of hash set (Equation answer is the key, equation is the value):”
2. Create Iterator<Equation> object called temp
3. Set temp to hashset object’s (hs) iterator
4. For each item in the hashset
   1. Create temporary object to store the next equation in the iteration
   2. Print the equation
5. Print “Successfully displayed all <number of items in hs> items.\r\n”

### ActionPerformed

* + - 1. Initialise **status** string
      2. If the action is from the **combo box, first operand text field, or second operand text field**:
         1. Store first operand as a float
         2. Store second operand as a float
         3. Calculate the answer
         4. Store the answer to the equation as a float
         5. Set the text of the answer field to the answer, with ‘.0’ stripped from the string if it ends with ‘.0’.
         6. Set status to “Calculated successfully. Click 'send' (hotkey: Alt + S) when you are ready.”
      3. If the action is from the **send** button:
         1. If the fields are not empty

Store operands as floats in RAM

Calculate and store answer in RAM

Create equation object to store the equation in RAM

Add the equation object to the table model object

Add a new node with the equation to the doubly linked list

Set the text of the linked list text area to the contents of the doubly linked list, in string format

Add equation to binary tree

Add equation to hash set

Display hash set

Append arithmetic log with the equation object

Clear fields

Call the server handling method and pass in the equation data

Disable send button

Repaint JPanel

* + - 1. If the action is from the **Bubble sort** button
         1. Set boolean to the opposite value.
         2. Call bubble sort method to sort by answer, passing in the array list, and Boolean that determines whether it is ascending or descending (allows for toggle functionality)
         3. Update the table data
      2. If the action is from the **Insertion sort** button
         1. Set boolean to the opposite value.
         2. Call insertion sort method to sort by answer, passing in the array list, and Boolean that determines whether it is ascending or descending (allows for toggle functionality)
         3. Update the table data
      3. If the action is from the **Selection sort** button
         1. Set boolean to the opposite value.
         2. Call selection sort method to sort by answer, passing in the array list, and Boolean that determines whether it is ascending or descending (allows for toggle functionality)
         3. Update the table data
      4. If the action is from the **save** button
         1. Write to file with the data from the binary tree text area

If pre-order, file name is “PreOrder.txt”

If in-order, file name is “InOrder.txt”

If post-order, file name is “PostOrder.txt”

* + - 1. If the action is from the **chart** button
         1. Create a new instance of the chart class, passing the array list into the constructor
         2. Set the size to 650, 490

### paintComponent

1. **Title**
   1. Display blue rectangle
   2. Display “instructor” text with 20 font size and white colour
2. Draw white string with size 12, displaying the **number of equations** loaded from the arithmetic log.
3. **Equation form**
   1. Call the drawStyledBox method to draw a box around equation form
4. **Linked list**
   1. Call the drawStyledBox method to draw a box around linked list
   2. Draw black string “linked list (of all incorrectly answered exercises):”
5. **Binary tree**
   1. Call the drawStyledBox method to draw a box around binary tree
   2. Draw black string “Binary Tree:”

### drawStyledBox

1. Draw filled rectangle with specified size and colour
2. Draw rectangle with specified size and colour

### Connect

1. Display text “Establishing connection. Please wait ...” in the status label
2. Instantiate socket with the Socket class, passing the server name and port into the constructor
3. Set status text to say “Connected to <host name>”
4. Call open method

### Open

1. Create an instance of the DataOutputStream class, passing in the return value of the Socket.getOutputStream method
2. Instantiate the instructor thread, passing in this class object, and the socket

### Close

1. If streamOut is not null
   1. Call streamOut.close
2. If socket is not null
   1. Call socket.close
3. Call instructorThread.close
4. Call instructorThread.stop

### InstructorHandle

1. If message equals “.bye”
   1. Set status text to “good bye. Press EXIT button to exit …”
   2. Call close method
2. Else
   1. Get last equation in array list
   2. If message equals the correct answer
      1. Set status to display “Student answered correctly! You can now send a new question.”
      2. Disable send button
   3. Else if message does not contain “/” (This is only the case if it is a full equation with the date time stamp)
      1. Set status to “Student answered incorrectly. Waiting to recieve correct answer...”
      2. Add to linked list
      3. Update text area to display the current data in the linked list

### Run

1. Print “Waiting for a client ...”
2. Call addThread method, passing in the return value of server.accept

### start

1. If thread is null
   1. Instantiate thread, passing in this
   2. Start thread

### stop

1. If thread isn’t null
   1. Stop thread
   2. Set thread to null

### findClient

1. For each client
   1. If client id equals the specified id
      1. Return the index
2. Return -1

### serverHandle

1. If input equals “.bye”
   1. Find the client and send “.bye”
   2. Remove specified client
2. Else
   1. For each client
      1. Send input

### remove

1. Set pos to the return value of findClient(<specified id>)
2. If post is greater than or equal to 0
   1. Initialise and define ServerThread object “ToTerminate” as the item in the client thread array at pos
   2. Print “Removing client thread <id> at <pos>
   3. If pos is less than the client count -1
      1. For each client
         1. Set client array item at the current index -1 to the client array item at the current index
   4. Decrement client count
   5. Call toTerminate.close
   6. Call toTerminate.stop

### addThread

1. If there is room in server thread array
   1. Print “client accepted: <socket>”
   2. Create a new server thread in array
   3. Call server thread’s open method
   4. Call server thread’s start method
   5. Increment client count
2. Else print “Client refused: maximum <server thread array length> reached.”

# Student screen(Student.java)

## IPO Chart

|  |  |  |
| --- | --- | --- |
| **Input** | **Process** | **Output** |
| Question text field | Stores the text in RAM | Displays the text |
| Answer text field | Stores the text in RAM | Displays the text |
| Submit button | Sends answer to server  Checks whether answer is correct  call showMessageDialog method | Displays pop up message saying whether you answered correctly or not |
| Status label | sets the text to the text specified by the program | Displays the text |

## Algorithms

### Constructor

1. Store server host string in RAM
2. Store port number in RAM
3. Set layout to null
4. Call initialiseComponents
5. Add components
6. Connect to server

### initialiseComponents

1. Create submit button
2. Create equation text field
3. Disable editing of equation text field
4. Create answer text field
5. Create status label

### paintComponent

1. Draw filled blue rectangle for title
2. Draw white string “Student” with 20 font size
3. Draw black string “Question: “
4. Draw black string “Answer: “

### actionPerformed

1. If btnSubmit is the action event
   1. Call send method
   2. If answer is correct
      1. Display message “Correct!”
      2. Disable button
   3. Else
      1. Display message “Incorrect; Please try again”
      2. Enable button
   4. Clear answer text field

### Connect

1. Display text “Establishing connection. Please wait ...” in the status label
2. Instantiate socket with the Socket class, passing the server name and port into the constructor
3. Set status text to say “Connected to <host name>”
4. Call open method

### send

1. Call DataOutputStream object’s writeUTF method, and pass in the answer, with “.0” stripped if necessary.
2. Call DataOutputStream object’s flush method

### Handle

1. If message equals “.bye”
   1. Display message “good bye. Press EXIT button to exit…”
   2. Call close method
2. Else if message split by spaces, has a length of 6
   1. Store split message in temp string array
   2. Get values from array and store in separate variables
   3. Current question equals new equation with the values
   4. Set the equation text field’s text to show the equation, without the answer
   5. Enable btnSubmit

### Open

1. Create an instance of the DataOutputStream class, passing in the return value of the Socket.getOutputStream method
2. Instantiate the student thread, passing in this class object, and the socket

### Close

1. If streamOut is not null
   1. Call streamOut.close
2. If socket is not null
   1. Call socket.close
3. Call studentThread.close
4. Call studentThread.stop

### stop

1. If thread isn’t null
   1. Stop thread
   2. Set thread to null

# Sorting (Sort.java)

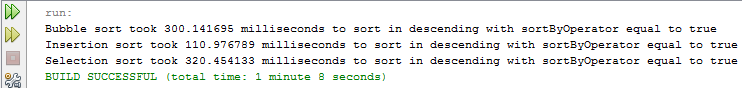
## algorithm Comparison

### testing

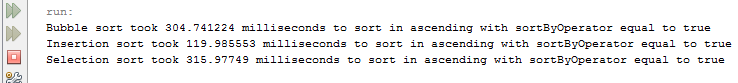
The three sorting algorithms were timed in milliseconds to determine their speeds when sorting the ArrayList<Equation> which contained 9,236 elements. (Note: This was tested on a computer which has a lot of other processes running in the background, which may make the results inaccurate).

Here are the results of the tests:

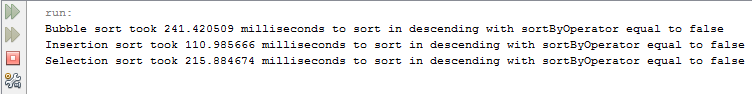
#### Sort by operator in Descending order



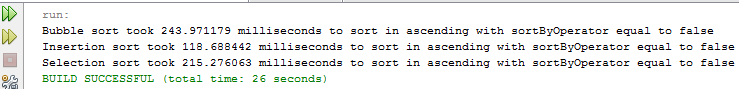
#### Sort by operator in ascending order



#### sort By answer in descending order



#### Sort By Answer In Ascending Order



### Algorithm details

As shown on the previous page, the bubble sort, selection sort, and insertion sort all have fairly different speeds. The difference in speed is enough for the average person to notice.

The speeds depend on many factors, the primary factors are the datatype being sorted, whether it is sorting in ascending order or descending order, and of course the size of the array. There are other little factors that make a small difference, for example: other processes happening on the computer at the same time, or other programs using the memory.

The methods used to calculate the speeds may not be too accurate, considering the time it may take to run System.nanoTime(), which has not been tested.

Insertion sort may have won because it only has to read through the array once, unlike the selection sorting, and the bubble sorting algorithms. This would have a clear advantage on most large arrays, and it has the benefit of being a very simple algorithm to learn and understand.

The pseudo-code for the bubble sort algorithm has been written to be easily read, whereas the code itself is much shorter but more confusing. Ternary operators were heavily used to write the methods so that the bubble sort and selection sort only require one if statement, and the insertion sort does not require any if statements.

Writing a single method that has the ability to sort multiple datatypes in ascending or descending order requires a lot of code to determine how it needs to compare the values. Generally, the easy way would be to use lots of if statements as shown in the pseudo-code, but using ternary operators, while ultimately doing the same thing, made the code much shorter than what it would have been with if statements.

In hindsight it would have been better to just make lots of different methods, but it was an interesting exercise nonetheless.

#### Selection Sort

This sorting algorithm was the slowest on average of the three, when dealing with an ArrayList of equations. Selection sort is faster than bubble sort at sorting by the answer (float), beating it by 25.535916 milliseconds in descending order, and 28.695116 milliseconds in ascending order.

Here is the pseudo-code for the selection sort:

|  |
| --- |
| For int i = 0; i < arr.size - 1; i++  int index equals i  For int j = i + 1; j < arr.size; j++  If sortByOperator is false  if desc is true  if num1 > num2  index = j  else  if num1 < num2  index = j  end if  end else  end if  else  if desc is true  if operator1 is lexicographically greater than operator2  index = j  end if  end if  else  if operator1 is lexicographically less than operator2  index = j  end if  end else  end else  end for  Equation temp = get array item at i  set array item at i to array item at index  set array item at index to temp  end for |

#### Bubble Sort

This algorithm came second in terms of the average speed. It beat the selection sort at sorting by operators, though still far behind insertion sort.

1. j = 0
2. for j < array size; j++
   1. i = j + 1
   2. for I < array size; i++
      1. if the return value of comparator(array item j, array item i, sort by operator/sort by answer, asc/desc) is true
         1. swap them around
3. return array

#### Insertion sort

Insertion sort is quite obviously the fastest of the three, even noticeable to the human eye. Its slowest speed was 119.985553, which is still much faster than the other two sorting algorithms.

1. Initialize key
2. Initialize i
3. j = 1
4. For j < array size; j++
   1. Key = array item j
   2. For i = j - 1; (i >= 0) && comparator(arr.get(i), key, sort by operator/sort by answer, asc/desc); i--
      1. Set array item i+1 to array item i
   3. Set array item i+1 to key
5. Return array

### Comparator

1. If sortByOperator is false
   1. If desc is true
      1. Return true if eq’s answer is greater than eqTwo’s answer
   2. If desc is false
      1. Return true if eq’s answer is less than eqTwo’s answer
2. If sortByOperator is true
   1. If desc is true
      1. Return true if the return value of comparing eq’s operator with eqTwo’s operator using the Character.compareTo method is less than zero
   2. If desc is false
      1. Return true if the return value of comparing eq’s operator with eqTwo’s operator using the Character.compareTo method is greater than zero

# JFreeChart (Third Party) (chart.java)

JFreeChart is a free chart library for Java that can generate a wide variety of charts for use in both client (Swing and JavaFX) and server (web) applications.

This library was used to draw a line graph displaying the student’s improvement in the 10 most recent equations.

## Algorithms

### Constructor

Parameter: ArrayList<Equation> eqs

1. Set visibility to true
2. Set layout to null
3. If eqs is null
   1. Return
4. Call displayLineChart(eqs)
5. Set size to 640, 480
6. Call initComponents (Auto generated with netbeans)

### paint

1. Define new file “LineChart.jpeg”
2. Initialize bufferedImage and define it as null
3. Image equals the return value of ImageIO.read, passing the file in as a parameter
4. Draw the image

### DisplayLineChart

1. Instantiate DefaultCategoryDataSet object
2. Initialize num
3. Define rowKey as “Number of correct answers out of the last 10 equations”
4. For i = eqs.size()-10; i < eqs.size(); i++
   1. If eqs item I’s answer equals eqs item I’s correct answer
      1. Num++
      2. Add value to the data set
   2. Else
      1. Num--
      2. Add value to the data set
5. Call ChartFactory.createLineChart
6. Image size equals 640, 480
7. Save chart as jpg file

# Utilities (Utilities.java)

This class contains potentially re-usable methods that are loosely coupled and highly cohesive.

The class mostly contains methods for creating components like labels, text fields, and frames. This reduces the amount of repetition in the code, and reduces the file size.

Below are some of the noteworthy methods.

## Algorithms

### CreateRadioButton

This method takes the parameters: text (string); rect (rectangle); defaultState (Boolean); font (Font); and event\_itemStateChanged (Callable<Integer>).

This is noteworthy because it uses reflection to call the event\_itemStateChanged

1. Instantiate JRadioButton with the constructor arguments: text and defaultState
2. Set font
3. Set location
4. Set size
5. Add ItemListener with a new instance of ItemListener
   1. Override itemStateChanged method
      1. Call event\_itemStateChanged

### stripzero

This method strips “.0” off of a string, and returns it. This is useful for when a float, double, or decimal is converted to a string because if it is a whole number, it will have “.0” on the end, which may be unnecessary.

For example, “5256.0” would become “5256”.

# Binary Tree (BinaryTree.java)

The binary tree class is used to store the equations in the form of a binary tree. It is designed to be outputted as a string in a text field in Instructor.java.

## BinaryTree Algorithms

### AddLeaves

Parameter(s): ArrayList<Equation> eqs

1. For each equation in eqs
   1. Add leaf (node) to the tree.

### AddLeaf

Parameter(s): Equation e

1. Define new Leaf object with specified equation
   1. If the root node is null
      1. Root equals the new Leaf
   2. If the root is NOT null
      1. Set the focus leaf to root
      2. Initialize leaf “parent”
      3. Infinite loop
         1. Set parent to focus leaf
         2. If e’s answer is less than the answer in the focus leaf
            1. Focus leaf equals the left child of focus leaf
            2. If focus leaf is null

Parent’s left child equals new leaf

Return void

* + - 1. If e’s answer is greater than the answer in the focus leaf
         1. Focus leaf equals the right child of focus leaf
         2. If focus leaf is null

Parent’s right child equals new leaf

Return void

### InOrderTraverseTree

Parameter(s): Leaf focusLeaf

1. If focus leaf is not null
   1. Call inOrderTraverseTree(focus leaf’s left child)
   2. Append global variable “treeString” with the focus leaf’s equation
   3. Call inOrderTraverseTree(focus leaf’s right child)

### PreOrderTraverseTree

Parameter(s): Leaf focusLeaf

1. If focus leaf is not null
   1. Append global variable “treeString” with the focus leaf’s equation
   2. Call preOrderTraverseTree(focus leaf’s left child)
   3. Call preOrderTraverseTree(focus leaf’s right child)

### PostOrderTraverseTree

Parameter(s): Leaf focusLeaf

1. If focus leaf is not null
   1. Call postOrderTraverseTree(focus leaf’s left child)
   2. Call postOrderTraverseTree(focus leaf’s right child)
   3. Append global variable “treeString” with the focus leaf’s equation

### GetEquationAndStripZeroes

Parameter(s): Leaf focusLeaf

1. Get focusLeaf’s answer
2. Get focusLeaf’s first operand
3. Get focusLeaf’s operator
4. Get focusLeaf’s second operand
5. Convert answer, operand one, operator, and operand two to strings
6. Strip ‘.0’ from numbers if necessary.
7. Return string in the format: “<answer> (<operand one><operator><operand two>), ”

### getTreeAsString

1. Set temp string to treeString without the comma at the end
2. Clear treeString
3. Return temp string

### findLeaf

Parameter(s): float answer

1. Set focus leaf to root
2. While focus leaf’s answer is not equal to answer
   1. If answer is less than focus leaf’s answer
      1. Focus leaf equals focus leaf’s left child
   2. Else
      1. Focus leaf equals focus leaf’s right child
   3. If focus leaf is null
      1. Return null
3. Return focus leaf

## Leaf (node) Algorithms

This class is used for storing a single node of the binary tree

### Constructor

Return the equation contained in the node

# DLinkedEquationList (DLinkedEquationList.java)

This class is a doubly/double linked list that stores Equation objects.

## Algorithms

### Constructor (no parameters)

Sets the head to a new instance of ListNode

### Constructor (Equation specified)

Sets the head to a new instance of ListNode(equation)

### Constructor (float, char, float, float)

Sets the head to a new instance of ListNode(float, char, float, float)

### get

Parameter(s): int i

1. Current equals head
2. If i is less than zero or current equals null
   1. Throw new ArrayIndexOutOfBoundsException
3. While i > 0
   1. Decrement i
   2. Current equals node after current
   3. If current is null
      1. Throw new ArrayIndexOutOfBoundsException
4. Return current

### getLength

1. n equals head
2. i equals 0
3. while node after n is not null
   1. increment i
4. return i

### displayAsString

1. str equals “HEAD <-> “
2. current equals head’s next node
3. For current is not equals to head and current is not null; current equals current’s next node
   1. String array arr equals the return value of current’s equation’s toStringArray method
   2. str += arr[0] + " " + arr[1] + " " + arr[2] + " = " + arr[3] + " <-> "
4. return str + “TAIL”

# ListNode (ListNode.java)

This class was originally inside the DLinkedEquationList.java file, but it was taken out because I needed to make it public. It would have been better to have it as a sub-class, though.

The purpose of this class is to store a single node of the doubly linked list.

## Algorithms

### Constructor (No parameters)

1. previous node equals null
2. next node equals null
3. equation equals null

### Constructor (Equation eq)

1. previous equals null
2. next equals null
3. equation equals eq

### Constructor (float operand one, operator, operand two, answer)

1. previous equals null
2. next equals null
3. equation equals new instance of Equation with the arguments specified

### Append

Parameter(s): float operandOne, char operator, float operandTwo, float answer

1. n equals new ListNode(operandOne,operator,operandTwo,answer)
2. n’s previous node equals this
3. n’s next node equals next
4. if next is not null
   1. next’s previous node equals n
5. next equals n

### Insert

Parameter(s): ListNode newNode

1. newNode’s previous node equals prev
2. newNode’s next node equals this
3. prev’s next node equals newNode
4. prev equals newNode

### Remove

1. next’s previous node equals prev
2. prev’s next node equals next

### toString (Override)

Returns the equation as a string in this format: “<num1> <operator> <num2> = <answer>”.

# TableModel (TableModel.java)

This class extends AbstractTableModel, and is used to store an ArrayList of Equation objects and a string array containing the table column headings.

This is implemented in the instructor class as the model for the JTable.

## Algorithms

### Constructor

Parameter(s): ArrayList<Equation> eqs, String[] header

1. this instance’s header equals header
2. this instance’s array list equals eqs

### getRowCount

Returns the size of the array list

### getColumnCount

Returns the length of the header

### getValueAt

Parameter(s): int rowIndex, int columnIndex

Returns cell from array list at rowIndex, columnIndex

### getColumnName

Parameter(s): int index

Return header’s value at position: index

### Add

Parameter(s): Equation e

1. if e’s answer is correct
   1. add equation
2. call fireTableDataChanged method

# InstructorThread (InstructorThread.java)

This class extends Thread, and is used in the instructor class.

## Algorithms

### Constructor

Parameter(s): Instructor \_instructor, Socket \_socket.

1. Instructor equals \_instructor
2. Socket equals \_socket
3. Call open method
4. Call start method

### Open

Open the input stream

### close

Close the input stream

### Run (override)

1. Infinite loop
   1. Call the instructorHandle method with the data read from the input stream.

# ServerThread (ServerThread.java)

This class extends Thread, and is used in the instructor class. The server could have been put into a separate file so that the instructor can run on a different computer to the server, but it isn’t quite necessary for this project.

## Algorithms

### Constructor

1. Call super()
2. server = \_server
3. socket = \_socket
4. id = the socket’s port

### Send

Parameter(s): String msg

Writes the msg to the DataOutputStream and flush the stream.

### run (override)

Call the serverHandle method with the parameters: ID and the data from the DataInputStream.

### open

Opens the input stream, and the output stream

### close

1. if socket isn’t null, close it
2. if input stream isn’t null, close it
3. if output stream isn’t null, close it

# StudentThread (StudentThread.java)

This class extends Thread, and is used in the student class.

## Algorithms

### Constructor

Parameter(s): student \_student, Socket \_socket.

1. student equals \_student
2. Socket equals \_socket
3. Call open method
4. Call start method

### Open

Open the input stream

### close

Close the input stream

### Run (override)

1. Infinite loop
   1. Call the student handle method with the data read from the input stream.

# References

JFreeChart: <https://sourceforge.net/projects/jfreechart/files/>

JFreeChart JavaDoc: <http://www.jfree.org/jfreechart/api/javadoc/index.html>

Sorting: <http://www.java2novice.com/java-sorting-algorithms>, MathBits\_Java\_InsertionSort.pdf, MathBits\_Java\_BubbleSort.pdf, MathBits\_Java\_SelectionSort.pdf

Ternary operators: <http://alvinalexander.com/java/edu/pj/pj010018>

JRadioButton: <http://www.tutorialspoint.com/swing/swing_jradiobutton.htm>

Pass function as parameter: <http://stackoverflow.com/questions/4685563/how-to-pass-a-function-as-a-parameter-in-java>