**Name: Lenell Davis**

**Class: CMIS 242**

**Professor: Goode**

**Assignment: Project Three**

**Date: 4/24/16**

Problem Analysis

Create a GUI that allows the user to input a positive integer that will display a result according to a predetermined sequence. The sequence is: each term is twice the previous term plus the second previous term. For example, [0 1 2 5 12 29], if the user inputs 5, the value displayed should be 29.

The program will use either a recursive call or an iterative call to compute the proper value. The program will also display the efficiency of each method call (ie the number of times the method/loop is called before the final value is displayed.)

Design

The program requires two classes. My program will have 3 classes. One will hold the main method which will run the program. Another class will contain the GUI components. The final class will do contain the sequence computing methods and set the efficiency value for each method called.

Two inner classes will be used to handle the button action event as well as the window close event. The button will create the txt file as well as call the iterative and recursive methods.

I will utilize two different print methods to print the same file. The print format will be slightly different so that the iterative/recursive efficiency value will have its own column.

**Test Plan**

|  |  |  |  |
| --- | --- | --- | --- |
| Issue | Expected | Results | Passed |
| Test that the user can only enter positive whole numbers. | Please enter a whole number. Or Please enter a positive whole number | Shows correct dialog and text for situations where the user has no text, decimal, or negative input. | Y |
| Test that the sequence works for both methods. | N= 0 - 0  N= 1 - 1  N= 2 - 2  N= 3 - 5  N= 4 - 12  N= 5 - 29  N= 6 - 70 | N= 0 - 0  N= 1 - 1  N= 2 - 2  N= 3 - 5  N= 4 - 12  N= 5 - 29  N= 6 - 70 | Y. Correct values are displayed for both the iterative and recursive methods. |
| Test the efficiency the iterative method | N= 0 - 1  N= 1 - 1  N= 2 - 2  N= 3 - 3  N= 4 - 4  N= 5 - 5 | N= 0 - 1  N= 1 - 1  N= 2 - 2  N= 3 - 3  N= 4 - 4  N= 5 - 5 | Y |
| Test the efficiency for recursive method | N= 0 - 1  N= 1 - 1  N= 2 - 3  N= 3 - 4  N= 4 - 5  N= 5 - 6 | N= 0 - 1  N= 1 - 1  N= 2 - 3  N= 3 - 5  N= 4 - 9  N= 5 - 15 | N. This is due to how the equation is set up.   1. **int** num1 = computeRecursive(n-1); 2. **int** num2 = computeRecursive(n-2); 3. **int** newTerm = ((2 \* num1) + num2);   In order to get the correct number in the sequence to add to the multiple, I needed to call the method again at the correct position. |
| Test that file is outputted properly. | Output.txt with values separated by commas | Output.txt with values separated by commas | Y |

**Code**

**Main Program**

1. /\*
2. \* Lenell Davis
3. \* CMIS 242
4. \* Project 3
5. \* 4/24/16
6. \* TestDriver.java
7. \*/

10. **public** **class** TestDriver {
12. **public** **static** **void** main(String[] args) {
13. Gui myView = **new** Gui();
14. myView.setVisible(**true**);
16. }
17. }

**Gui.java**

1. /\*\*
2. \* Lenell Davis
3. \* CMIS 242
4. \* Project 3
5. \* 4/24/16
6. \* Gui.java
7. \*/
9. **import** javax.swing.\*;
10. **import** java.awt.\*;
11. **import** java.awt.event.\*;
12. **import** java.io.\*;
14. **public** **class** Gui **extends** JFrame{
15. **private** **final** Font font = **new** Font ("Arial", Font.BOLD, 18);
16. **private** **final** JFrame frame = **new** JFrame();
18. **private** **final** ButtonGroup userSelection = **new** ButtonGroup();
19. **private** **final** JLabel enterLabel = **new** JLabel("Enter a number: ");
20. **private** **final** JLabel resultLabel = **new** JLabel("Result");
21. **private** **final** JLabel effLabel = **new** JLabel("Efficiency");
23. **private** **final** JRadioButton iterativeBtn = **new** JRadioButton("Iterative");
24. **private** **final** JRadioButton recursiveBtn = **new** JRadioButton("Recursive");
26. **private** **final** JTextField enterField = **new** JTextField(15);
27. **private** **final** JTextField resultField = **new** JTextField();
28. **private** **final** JTextField effField = **new** JTextField();
30. **private** **final** JButton calcBtn = **new** JButton("Compute");
32. **private** **final** JPanel choicePanel = **new** JPanel();
33. **private** **final** JPanel enterPanel = **new** JPanel();
34. **private** **final** JPanel buttonPanel = **new** JPanel();
35. **private** **final** JPanel resultPanel = **new** JPanel();
36. **private** **final** JPanel mainPanel = **new** JPanel();
38. **private** **final** String fileName = "output.txt";
39. **private** PrintWriter output;
40. **private** FileWriter fw;
41. **private** BufferedWriter bw;
43. /\*
44. \* Constructor for the MyGui View
45. \*/
46. **public** Gui(){
47. setFrame();
48. setPanels();
49. setAttributes();
50. addWindowListener(**new** CloseApp());
51. }

54. /\*
55. \* Sets the attributes for the frame
56. \*/
57. **private** **void** setFrame() {
59. setSize(600, 300);
60. setVisible(**true**);
61. setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);
62. setTitle("Recursive vs Iterative");
63. add(mainPanel);
64. }
66. /\*
67. \* Sets the attributes for the components
68. \*/
69. **private** **void** setAttributes(){
70. mainPanel.setLayout(**new** GridLayout(4,1));
71. resultPanel.setLayout(**new** GridLayout(2,2));
72. userSelection.add(iterativeBtn);
73. userSelection.add(recursiveBtn);
74. iterativeBtn.setFont(font);
75. recursiveBtn.setFont(font);
76. enterLabel.setFont(font);
77. resultLabel.setFont(font);
78. effLabel.setFont(font);
79. calcBtn.setFont(font);
80. resultField.setEditable(**false**);
81. effField.setEditable(**false**);
82. iterativeBtn.setSelected(**true**);
83. calcBtn.addActionListener(**new** buttonClick());
84. }
86. /\*
87. \* Adds the components to the Panels
88. \* Adds the panels to the mainPanel
89. \*/
90. **public** **void** setPanels(){
91. choicePanel.add(iterativeBtn);
92. choicePanel.add(recursiveBtn);
94. enterPanel.add(enterLabel);
95. enterPanel.add(enterField);
97. buttonPanel.add(calcBtn);
99. resultPanel.add(resultLabel);
100. resultPanel.add(resultField);
101. resultPanel.add(effLabel);
102. resultPanel.add(effField);
104. mainPanel.add(choicePanel);
105. mainPanel.add(enterPanel);
106. mainPanel.add(buttonPanel);
107. mainPanel.add(resultPanel);
108. }
110. /\*
111. \* Method that calls the Sequence class and passes in the integer
112. \*/
113. **private** **void** callSequence(**int** a){
114. **if**(iterativeBtn.isSelected()){
115. **int** iterativeValue = Sequence.computeIterative(a);
116. **long** itEff = (Sequence.counter + 1);
117. resultField.setText(Integer.toString(iterativeValue));
118. effField.setText(Long.toString(itEff));
119. //Print to file
120. iPrintToFile(a, itEff);
122. Sequence.resetCounter(0);
123. }
125. **else** **if** (recursiveBtn.isSelected()){
126. **int** recursiveValue = Sequence.computeRecursive(a);
127. **long** recEff = Sequence.counter;
128. resultField.setText(Integer.toString(recursiveValue));
129. effField.setText(Long.toString(recEff));
130. //Print the values to the file.
131. rPrintToFile(a, recEff);
133. Sequence.resetCounter(0);
134. }
135. }
137. /\*
138. \* Test method that parses the text to an Integer and ensures it's not negative.
139. \*/
140. **private** **int** testMethod(String enterText){
141. **int** a = 0;
142. **int** b = 0;
143. **try**{
144. a = Integer.parseInt(enterText);
145. }
146. **catch**(NumberFormatException e){
147. JOptionPane.showMessageDialog(frame, "Please enter a whole number.");
148. }
150. **if** (a >= 0){
151. b = a;
152. }
153. **else** {
154. JOptionPane.showMessageDialog(frame, "Please enter a positive whole number.");
155. }
157. **return** b;
158. }
160. /\*
161. \* Print method that takes in the values and prints them to a file
162. \*/
163. **private** **void** iPrintToFile(**int** sequenceValue, **long** effValue){
164. output.println(sequenceValue + "," + effValue);
165. output.close();
166. }
168. **private** **void** rPrintToFile(**int** sequenceValue, **long** effValue){
169. output.println(sequenceValue + "," + "," + effValue);
170. output.close();
171. }

174. /\*\*
175. \*Inner class that handles the Compute Button
176. \* Opens the filewriter, the bufferedwriter and the printwriter
177. \*/
178. **private** **class** buttonClick **implements** ActionListener{
180. @Override
181. **public** **void** actionPerformed(ActionEvent e) {
182. **int** retrievedValue = testMethod(enterField.getText());
183. **try**{
184. fw = **new** FileWriter(fileName, **true**);
185. bw = **new** BufferedWriter(fw);
186. output = **new** PrintWriter(bw);
187. }
188. **catch**(IOException g){
189. JOptionPane.showMessageDialog(frame, "File was not found.");
190. }
192. callSequence(retrievedValue);
193. }
195. }
197. /\*\*
198. \*Inner class that handles the WindowClose Event
199. \* Closes the filewriter and the bufferedwriter
200. \*/
201. **private** **class** CloseApp **extends** WindowAdapter{
202. **public** **void** windowClosing(WindowEvent e)
203. {
204. **try**{
205. fw.close();
206. bw.close();
207. }
208. **catch**(IOException f){}
209. }
211. **public** **void** windowClosed(WindowEvent e){
212. System.exit(0);
213. }
214. }
215. }

**Sequence.java**

1. /\*\*
2. \* Lenell Davis
3. \* CMIS 242
4. \* Project 3
5. \* 4/24/16
6. \* Sequence.java
7. \*/
9. **public** **final** **class** Sequence {
10. **public** **static** **long** counter;
12. **private** Sequence(){
13. counter = 0;
14. }
16. /\*\*
17. \*Computes the sequence values iteratively
18. \*/
19. **public** **static** **int** computeIterative(**int** n){
20. **int** previous = 0;
21. **int** current = 1;
23. **if**(n==0){
24. **return** previous;
25. }
27. **for**(**int** i = 1; i < n; i++){
28. **int** result = (2 \* current) + previous;
29. previous = current;
30. current = result;
31. getEfficiency();
32. }
34. **return** current;
35. }
37. /\*\*
38. \*Computes the sequence values recursively
39. \*/
40. **public** **static** **int** computeRecursive(**int** n){
41. **if**(n == 0){
42. **int** result = 0;
43. getEfficiency();
44. **return** result;
45. }
47. **if**(n == 1){
48. **int** result = 1;
49. getEfficiency();
50. **return** result;
51. }
53. **int** num1 = computeRecursive(n-1);
54. **int** num2 = computeRecursive(n-2);
55. **int** newTerm = ((2 \* num1) + num2);
56. getEfficiency();
57. **return** newTerm;
58. }
60. /\*\*
61. \*Calculates the efficiency for each method
62. \*/
63. **public** **static** **void** getEfficiency(){
64. counter ++;
65. }
67. /\*
68. \* Resets the counter variable
69. \*/
70. **public** **static** **void** resetCounter(**int** a){
71. counter = a;
72. }
74. }

**Lessons Learned**

I had the opposite problem compared to Project 2. In that I couldn’t get the class data to persist, but this time it did. Needed to add a counter reset as it would not reset with each button click. That may have required creating a new object in order to have the counter variable reset. However, creating multiple objects would add unnecessary load to the user system increasing the odd of a crash.

While recursive methods add simplicity to code, they are not necessarily best when taking into account system resources. With n = 10, the number of calls increased 17x. Iterative methods require more thought when setting up, but they are faster- particularly as the sequence becomes more complex. This may be remedied with a better formula, however I failed to notice one.

