***A3. demonstrate an understanding of object-oriented programming concepts and practices in the design and creation of computer programs***

**1:**

Being able to use a class multiple times is quite helpful. For example say you are making a user storage system. You would want to be able to structure a person efficiency and receive their data quick. You would usually have a Person class and within that class you would have methods that can be initialized through the constructor, received through get methods and set from set methods.

1. **public** **class** Person {
3. **private** String firstName;
4. **private** String lastName;
5. **private** **int** age;
7. **public** Person (String firstName, String lastName, **int** age) {
8. **this**.firstName = firstName;
9. **this**.lastName = lastName;
10. **this**.age = age;
11. }
13. **public** String getFirstName() {
14. **return** **this**.firstName;
15. }
17. **public** String getLastName() {
18. **return** **this**.lastName;
19. }
21. **public** **int** getAge() {
22. **return** **this**.age;
23. }
25. **public** **void** setFirstName(String firstName) {
26. **this**.firstName = firstName;
27. }
29. **public** **void** setLastName(String lastName) {
30. **this**.lastName = lastName;
31. }
33. **public** **void** setAge(**int** age) {
34. **this**.age = age;
35. }
36. }

**2:**

Classes are like containers for specific data. For example the code above, shows a Person class, the person class stores the user’s first and last name along with their age. This class can be initialized as much as the developer wants, to create a Person. That allows for the creation of an infinite amount of unique person classes. The constructor requires 3 parameters, there are also 3 getters to receive the private variables (firstName, lastName, age) and there are also 3 setter methods to set the value of either 3 of the variables (firstName, lastName, age).

**3:**

The following code will show how the Person class can be used within another class to create multiple unique users, without needing to create an entirely new class for each user. This makes everything more dynamic, and easier to use.

1. **public** **class** Core {
3. **private** List<Person> users = **new** ArrayList<>();
5. **public** Core() {
6. users.add(**new** Person("Kiran", "Hart", 17));
7. users.add(**new** Person("Megan", "Rogers", 16));
8. users.add(**new** Person("Jonathan", "Johnson", 17));
9. users.add(**new** Person("Arbaaz", "Rana", 16));
11. users.forEach(user -> System.out.println("User: " + user.getFirstName() + " " + user.getLastName() + " is " + user.getAge()))
12. }
14. **public** **static** **void** main(String[] args) {
15. **new** Core();
16. }
17. }

***A1. use data structures in the design and creation of computer programs***

*(A1.3):*

A one dimensional array is used to store single row elements, for example you make an array called String[] treeTypes = new String[3] with a single value in [] is a one dimensional array. You would be able to add 3 values at different indexes starting at 0 to the max value assigned minus 1.

treeTypes[0] = “Birch”;

treeTypes[1] = “Oak”;

treeTypes[2] = “Pine”

You would be able to get the value at the index using treeTpyes[index#], the **index#** is the spot of a specific element in the array, the **element** is the actual variable / value that is set in the specific point in the array. So if I wanted to get the element that contains “Oak” I would call treeTypes and pass in the index where it is, since It’s hard-coded I know it’s 1 so I’d pass 1 it (treeTypes[1]).

A two dimensional array works pretty much in the same way when it comes to the indexing, but you just have two. Think of it as a grid with an x and y but the 0, 0 starts at the top left. Let’s make a 2 dimensional array called int[][] board = new int[3][3];

1. **public** **class** Core {
3. **private** **static** **int**[][] board = **new** **int**[3][3];
5. **public** **static** **void** main(String[] args) {
6. **for** (**int** x = 0; x < 3; x++)
7. **for** (**int** y = 0; y < 3; y++)
8. board[x][y] = 0;
9. }
10. }

That code will fill in all the available indexes of the array [0-2][0-2] with 0. Let’s say I want to want to set the middle of the board to a 1.

|  |  |  |
| --- | --- | --- |
| 0 | 1 | 2 |
| 1 | **SET VALUE TO 1** |  |
| 2 |  |  |

I would do so by using this code, since java starts at 0 and we have 3 elements, the 2nd one in the array would technically be 1 since 0 is first. This is the same case for the x and y.

1. **public** **class** Core {
3. **private** **static** **int**[][] board = **new** **int**[3][3];
5. **public** **static** **void** main(String[] args) {
6. board[1][1] = 1;
7. System.out.println(board[1][1]);
8. }
9. }

***B3. design user-friendly graphical user interfaces (GUIs) that meet user requirements***

**1/4:**

The following code, is the code I made for the current project we’re working on (3D Tic Tac Toe) as it includes the usage of constructors, private variables, inner classes, loops to add buttons, list, mouse listeners. And layout management according to the x and y value to format everything nicely. Although it doesn’t exactly include labels or boxes, but eh, the point of designing the UI is there with this.

1. **import** java.util.ArrayList;
2. **import** java.util.List;
4. **import** org.eclipse.swt.SWT;
5. **import** org.eclipse.swt.events.MouseEvent;
6. **import** org.eclipse.swt.events.MouseListener;
7. **import** org.eclipse.swt.graphics.Point;
8. **import** org.eclipse.swt.widgets.Button;
9. **import** org.eclipse.swt.widgets.Display;
10. **import** org.eclipse.swt.widgets.Shell;
12. **public** **class** Core {
14. //The display and shell variables used to setup the window.
15. **private** Display display;
16. **private** Shell shell;
18. /\*
19. \* The List to store the all of the buttons
20. \* that can be clicked in the game.
21. \*/
22. **private** List<Button> gridButtons = **new** ArrayList<>();
24. /\*
25. \* Determine which player turn it is, start with X
26. \* then swap to player O, then back, vis-versa
27. \*/
28. **private** **boolean** playerXTurn = **true**;
30. /\*
31. \* This is where everything will be initialized, setting up the display
32. \* creating the 3x3 X3 grids, and any other methods to setup the game.
33. \*/
34. **public** Core() {
35. setupDisplay();
36. }
38. /\*
39. \* This method will perform everything that is needed
40. \* in order to setup the game and display the game.
41. \* Initialize the display and shell, generate the buttons
42. \* add the click listeners, set the size, and code for disposal
43. \*/
44. **private** **void** setupDisplay() {
45. display = **new** Display();
46. shell = **new** Shell(display);
48. // Set the title
49. shell.setText("3D Tic Tac Toe");
51. // Setup the grids / buttons
52. generateGridButtons();
54. // Add Listeners
55. addClickListeners();
57. shell.pack();
58. shell.setSize(196, 637);
59. shell.setVisible(**true**);
61. // Check for if the x button is closed.
62. **while** (!shell.isDisposed()) {
63. **if** (!display.readAndDispatch())
64. display.sleep();
65. }
67. shell.dispose();
69. }
71. /\*
72. \* This method will generate all of the grid buttons onto the shell and
73. \* display them.
74. \*/
75. **private** **void** generateGridButtons() {
76. **int** x = 0;
77. **int** y = 0;
79. //Loop through 27 times, 3x3 -- 3 times
80. **for** (**int** i = 0; i < 27; i++) {
81. /\*
82. \* Create an empty button with the size 60x60 pixels
83. \* then set the location according the the x and y.
84. \*/
85. Button button = **new** Button(shell, SWT.PUSH);
86. button.setSize(**new** Point(60, 60));
87. button.setLocation(**new** Point(x, y));
88. button.setVisible(**true**);
90. //Add 60 to x
91. x += 60;
92. //If x is 180; so 3 buttons on a row, set
93. //x to 0 to start on the left then increase y
94. //by 60 to move down a row.
95. **if** (x == 180) {
96. x = 0;
97. y += 60;
99. /\*
100. \* Add Spacer in between the grids
101. \*/
102. **if** (y == 180 || y == 380 || y == 580) {
103. Button filler = **new** Button(shell, SWT.PUSH);
104. filler.setSize(**new** Point(180, 20));
105. filler.setLocation(**new** Point(x, y));
106. filler.setVisible(**true**);
107. //add 20 to keep formatting nice
108. y += 20;
109. }
110. }
112. //Finally add the buttons to the list for later use.
113. gridButtons.add(button);
114. }
115. }
117. /\*
118. \* This method will loop through each of the buttons in the gridButtons
119. \* array and add the mouse listener class to the button, by getting the
120. \* button by index I.
121. \*/
122. **private** **void** addClickListeners() {
123. **for** (**int** i = 0; i < gridButtons.size(); i++)
124. gridButtons.get(i).addMouseListener(**new** MouseListeners(gridButtons.get(i)));
125. }
127. /\*
128. \* This is an inner class that implements the mouse listener this helps
129. \* minimize the code as much as possible, so I can loop through each button
130. \* and add the listener, and perform any action from there. Gotta make it
131. \* efficient you know.
132. \*/
133. **class** MouseListeners **implements** MouseListener {
135. **private** Button button;
137. **public** MouseListeners(Button button) {
138. **this**.button = button;
139. }
141. /\*
142. \* Check when the player clicks on the button, if they
143. \* click perform any check needed to place the O or X
144. \* character, and check wins.
145. \*/
146. @Override
147. **public** **void** mouseDown(MouseEvent e) {
149. /\*
150. \* Swap between player turns, if it was player
151. \* x's turn, switch to O, same thing for o to x.
152. \*/
153. **if** (playerXTurn) {
154. button.setText("X");
155. playerXTurn = **false**;
156. } **else** {
157. button.setText("O");
158. playerXTurn = **true**;
159. }
160. }
162. @Override
163. **public** **void** mouseDoubleClick(MouseEvent e) {
164. }
166. @Override
167. **public** **void** mouseUp(MouseEvent e) {
168. }
169. }
171. /\*
172. \* This is the main method, the program will call this method and run any
173. \* code within this class upon pressing start
174. \*/
175. **public** **static** **void** main(String[] args) {
176. **new** Core();
177. }
178. }

***B1. design standard algorithms according to specifications***

***A2. demonstrate the ability to use standard algorithms in the design and creation of computer programs***

***C2. demonstrate the ability to use software development tools to design and write a computer program.***

***B4. participate in a large student-managed project, using proper project management tools and techniques to manage the process effectively.***