Design pattern assessment

Question #1	**Write the NAME of one of the controller classes (or class that contains a controller). Copy and paste a code segment of the controller that calls the mutator of the model.**
Controller Class	`TicTacToeBoard`
Mutator of model	`// Only update the model if the winner is not found` `if (but.getText().equals("") && model.getWinner() == false) {` `// set the text with the corrrect value "0 or X"` `// Update the model` `(but).setText(model.getCurrPlayer());` `for (int i = 0; i < b.length; i++) {` `if (e.getSource() == b[i]) {` `model.update(i);` `}` `}` `// Check for the winner or tie` `model.isWinner();` • **In the above highlighted in yellow, when the player makes a move , Controller identifies the selected square and updates the model.** `else if(model.getMoves() == 9)` `{` `JOptionPane.showMessageDialog(null, "The game has been tied");` `undo.setEnabled(false);` `}` `// change the player because now it turn of other player` `model.changePlayer();` `// Display which player turn is this` `display.setText("Player " + model.getCurrPlayer() + " turn");` `// Get the current player and is used above to avoid multiple undos in a row` `curr = model.getCurrPlayer();` • **In the above highlighted yellow , the controller informs the model about the player who shall make next move.**

Question #2 Write the NAME of the model class. Copy and paste a code segment of a mutator of the model that modifies data and also notifies view(s). Give me the name of mutator as well.

Name	`Model`
Mutator code	`public void isWinner() {` `moves++;` `if (board[0].equals(currPlayer) && board[1].equals(currPlayer) && board[2].equals(currPlayer)) {` `isWinner = true;` `} else if (board[0].equals(currPlayer) && board[3].equals(currPlayer) && board[6].equals(currPlayer)) {` `isWinner = true;` `} else if (board[6].equals(currPlayer) && board[7].equals(currPlayer) && board[8].equals(currPlayer)) {` `isWinner = true;` `} else if (board[2].equals(currPlayer) && board[5].equals(currPlayer) && board[8].equals(currPlayer)) {` `isWinner = true;` `} else if (board[3].equals(currPlayer) && board[4].equals(currPlayer) && board[5].equals(currPlayer)) {` `isWinner = true;` `} else if (board[0].equals(currPlayer) && board[4].equals(currPlayer) && board[8].equals(currPlayer)) {` `isWinner = true;` `} else if (board[1].equals(currPlayer) && board[4].equals(currPlayer) && board[7].equals(currPlayer)) {` `isWinner = true;` `} else if (board[2].equals(currPlayer) && board[4].equals(currPlayer) && board[6].equals(currPlayer)) {` `isWinner = true;` `}` `}` **The above method isWinner is invoked for every move. This method checks and compares historic moves by the player and**

	rules to win. Based on the outcome of the comparison, it sets the Boolean whether the current player is winner. **Controller/View class invokes the method getWinner() to read the value of isWinner as shown below.**
Question #3	**Write the NAME of the view class. Copy and paste a code the notification method of the view and show me how the notification method paints the view using the data from the model.**
Name of View Class	`TicTacToeBoard`
	1) Below code depicts that based on the value returned from model undo(Button) is enabled. `// Initially the undo button is disable so we need to enable it` `// but since we don't allow multiple undos in a row we` `// use a curr variable to check if the player has changed` `if(curr.equals(model.getCurrPlayer()))` `{` `  undo.setEnabled(true);` `}` **2) In the below code snippet , depending on the value returned by getWinner() method View notifies the players who has won the game.** `if(model.getWinner())` `{` `  JOptionPane.showMessageDialog(null, "Player with " +` `  model.getCurrPlayer() + " has won");` `  undo.setEnabled(false);` `}` `else if(model.getMoves() == 9)` `{` `  JOptionPane.showMessageDialog(null, "The game has been tied, please`

	reset for another game"); undo.setEnabled(false); } 3) Below code snippet updates whose turn it is based on the value provided by model from getCurrentPlayer() // Display which player turn is this display.setText("Player " + model.getCurrPlayer() + " turn");
Question#4	**Write the NAME of a strategy and copy the code.**
Name of Strategy	**BoardStyle** · Strategy is an interface providing the specifications of the board style . The concrete classes shall extend this class and provide the specific behaviour intended by each concrete class. · There can be N number of concrete classes extended by implementing this interface. · This is focussed and presenting different styles according to the user selection. * In this project it is chosen to be 2 board styles implemented with 2 * strategies 1. First Strategy with FirstBoardStyle.java implementing this * interface 2. Second Strategy with SecondBoardStyle.java implementing this * interface * * @author Admin * */ public interface BoardStyle { /** * This method is to specific operation on defined variables. This method will * be useful when the user starts playing the game This will be implemented as * part of MVC model to handle the user interaction Sub classes need to overide * the logic of implementation according their needs. */ public void doOperation(); /** * This method allows to show the color of board. Each subclass can define their * own color * * @return

	`` */ public java.awt.Color getBtnColor(); /** * This method defines the skin layout of the board Subclasses implementing this * method can define the specific theme depending on style requirement that * shall be presented. */ public void setBoardTheme(); } ```
Question#5	**Write the name of two concrete strategies. (Just names required).**
Names of the concrete classes	**FirstBoardStyle** **SecondBoardStyle**

Question#6	**Copy and paste the code segment where you create a concrete strategy and plug-in into the context program.**
Create concrete Strategy and plugin	**Below code snippet provides the information of creation of concrete strategy.** 1. **Player from view shall select the board style before starting the game(s)** 2. **Based on the selection Controller captures the selected board style and crates the appropriate Concrete Strategy class and plugs into the View.** `@Override` `public void actionPerformed(ActionEvent e) {` `JButton check = (JButton) (e.getSource());` `if (check.getText() == "Board Style 1") {` `strategyType = "style1";` `boardStrategy = new FirstBoardStyle();` `} else if (check.getText() == "Board Style 2") {` `strategyType = "style2";` `boardStrategy = new SecondBoardStyle();` `}` `boardStrategy.doOperation();` `frame.remove(strategy1);` `frame.remove(strategy2);` `frame.repaint(0, 0, 330, 450);` `undo.setEnabled(false);` `reset.setEnabled(false);` `}` **After plugging into the context, concrete class is later utilized to paint the view.** `b[i].setBackground(boardStrategy.getBtnColor());`