This project implements the Model View Controller architecture pattern as well as the Blackboard pattern. In addition, it uses the observer design pattern to update the GUI and logic parts of the program as new input comes in. It also uses Singleton to maintain and work with the data in one instance. More details about these patterns are listed below:

Model-View-Controller:

Model View Controller is an architectural design that splits the implementation of the logic, the user interface, and user actions. This pattern fits perfectly with the requirements of the project. The project requires a user interface for the user to input commands (mouse clicks), a controller to control what to do with those clicks, and the logic to create new data points/lines.

The View in this program is the Main and Canvas class. They are in charge of creating what the users see. Specifically, the Main class is the starting point for the program and it creates the frame of the GUI and connects dependencies for the whole system. The class Canvas fills this GUI with dots according to the user clicks and predefined rules to draw the lines. paintComponent is called whenever there is a new drawing that needs to be updated. To know when this should be called, the class implements the Observer class and waits for the observable to call update.

The Controller of this program is the class Reporter. It implements MouseListener to listen for the user clicked and perform an action. In this case, it calls KnowledgeSource to add the new point and lines.

The Model of this program is the collection of classes and they also formed the BlackBoard architectural pattern. It is in charge of processing the information in the background of the call update on the GUI to print their new processed data.

BlackBoard:

BlackBoard is an architectural design pattern that has a central place for information, or the blackboard, and uses other classes such as control and knowledge source to add/modify/delete information from this blackboard. It clearly simplifies and divides the responsibilities for each class.

In this program, the central place for information storing is the class Repository. This class is a Singleton class because many classes use it and it needs to maintain only 1 global copy of the data. Specifically, the data here is the points and lines list that need to be shown. Furthermore, this class is the Observable class because when new points/lines appear, it will notify the observers to calculate the rules of the lines and update the GUI with the correct information.

The KnowledgeSouce class is in charge of adding new points to the blackboard. Then, it calculates the closest already existing point and draws a line between them. Overall, this class takes care of new incoming information.

The Control class is in charge of making sure the new information follows the line rules. The rule states that when a point reaches 5 lines, it should be deleted and the lines to the old points should be redrawn. In order to know when to check, this class implements Observer. The goal for this is to let the blackboard call this check whenever it is needed. And since this rule can be violated when a new point comes in, it is called whenever the users click on the screen.

This program has implemented the technique of refinement, information hiding, and modularity by dividing each responsibility into its own classes and hiding unnecessary information. Large tasks are broken into many smaller methods for maintainability and reuse.