DataGuru App

A GUI Application for data analysis

Created By:

Ram Giri

TABLE OF CONTENTS

DataGuru GUI App building Overview

Introduction:

This report provides a concise explanation of the Data analysis GUI application and its Java code. The application allows users to load and analyze data from CSV files, create different types of charts, and draw custom graphics. This application contains number of classes, the application's uniqueness compared to other data analysis tools, and the strengths of the chatbot feature.

Number of Classes:

The "DataGuru" consists of the following classes:

1. **DataAnalysisApp:** The main class that handles the application's GUI, data loading, chart creation, custom graphics, and chatbot functionality. For chart generation, I use Jfree Chart Factory- a opensource library.
2. **CustomGraphicsPanel:** An inner class responsible for drawing custom graphics such as lines and text.
3. **Chatbot:** An inner class that simulates a chatbot for user interaction.

The application has three classes in total, with the CustomGraphicsPanel and Chatbot classes being inner classes within the main DataAnalysisApp class.

Features:

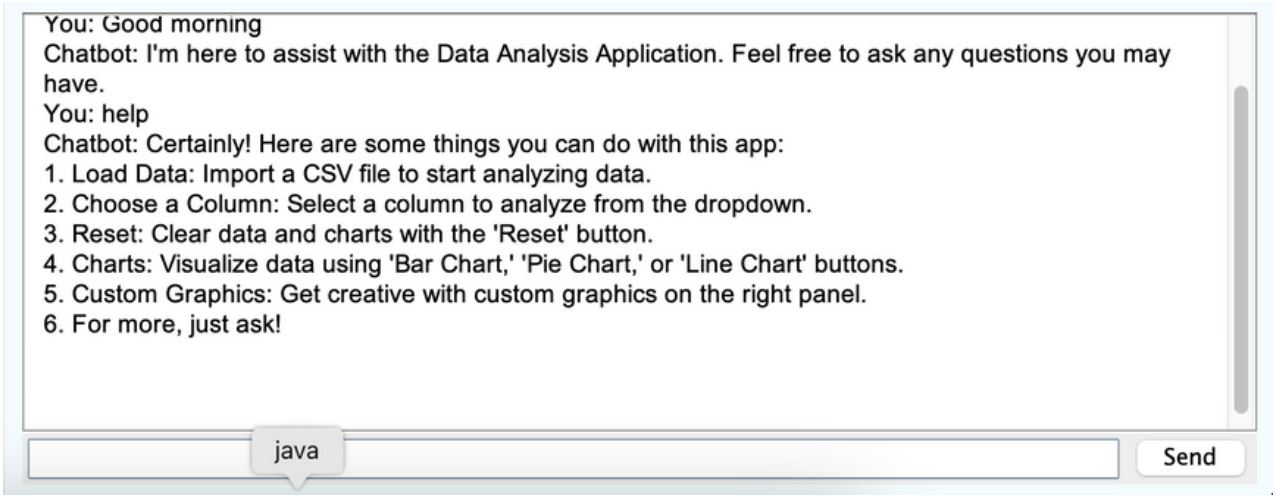
The "DataGuru" offers several unique features and characteristics compared to other data analysis tools:

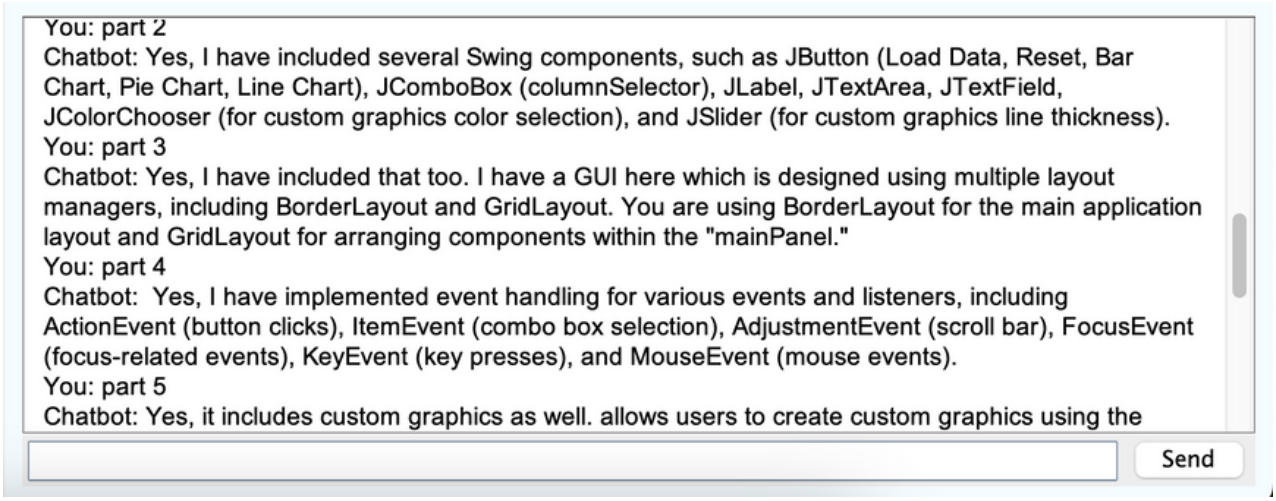
1. **Custom Graphics:** Unlike many data analysis tools, this application allows users to draw custom graphics directly on the interface. Users can draw lines and add text to create custom annotations, enhancing the visual representation of data.
2. **Interactive Chatbot:** The inclusion of a chatbot provides an interactive and user-friendly experience. Users can ask questions and receive responses, making the application more accessible and educational.
3. **Simple and Lightweight:** The application is designed to be lightweight and straightforward. It focuses on essential data analysis and visualization features, making it suitable for users who need a quick and easy way to analyze data without the complexity of more advanced tools.
4. **Educational Value:** The chatbot feature not only provides assistance but also offers information about data analysis, the application's features, and system requirements. This educational aspect sets it apart from many other data analysis tools.
5. **Customization:** While the customization options are currently limited, users can choose the color and line thickness for custom graphics. This basic level of customization adds a personal touch to the graphics.

Strengths of the Chatbot Feature:

The chatbot feature in the "Data Analysis Application" has several strengths:

1. **Interactive Assistance:** The chatbot allows users to interact with the application in a conversational manner. Users can ask questions, seek help, and receive responses in a friendly and approachable way.
2. **User Guidance:** The chatbot provides guidance on how to use the application, making it more user-friendly, especially for those who are new to data analysis.
3. **Educational Information:** The chatbot can provide information about data analysis, the application's features, and related topics. This feature adds educational value to the application, making it a learning tool as well.
4. **Queries Compliance:** The chatbot responds to specific queries related to the assignment requirements, demonstrating features of app and helpdesk functionalities.
5. **Convenience:** Users can receive quick answers to common questions, reducing the learning curve and improving the overall user experience.

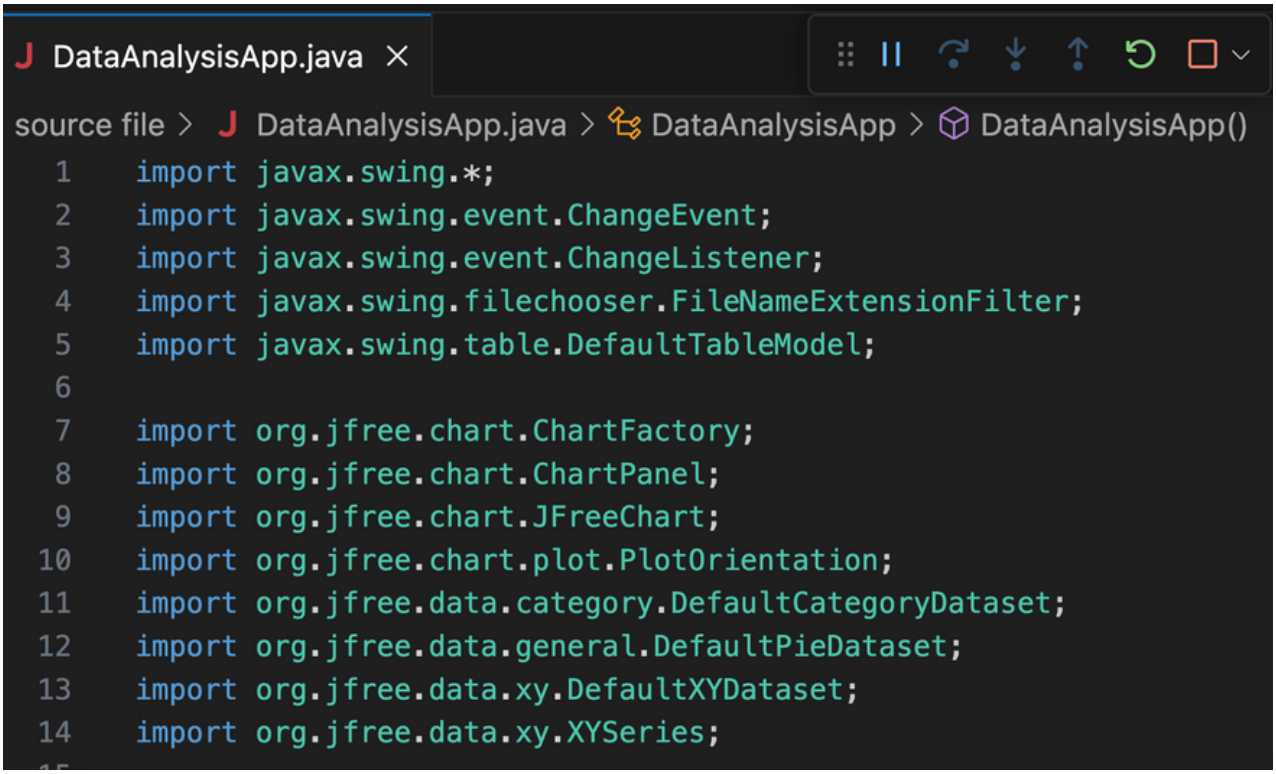


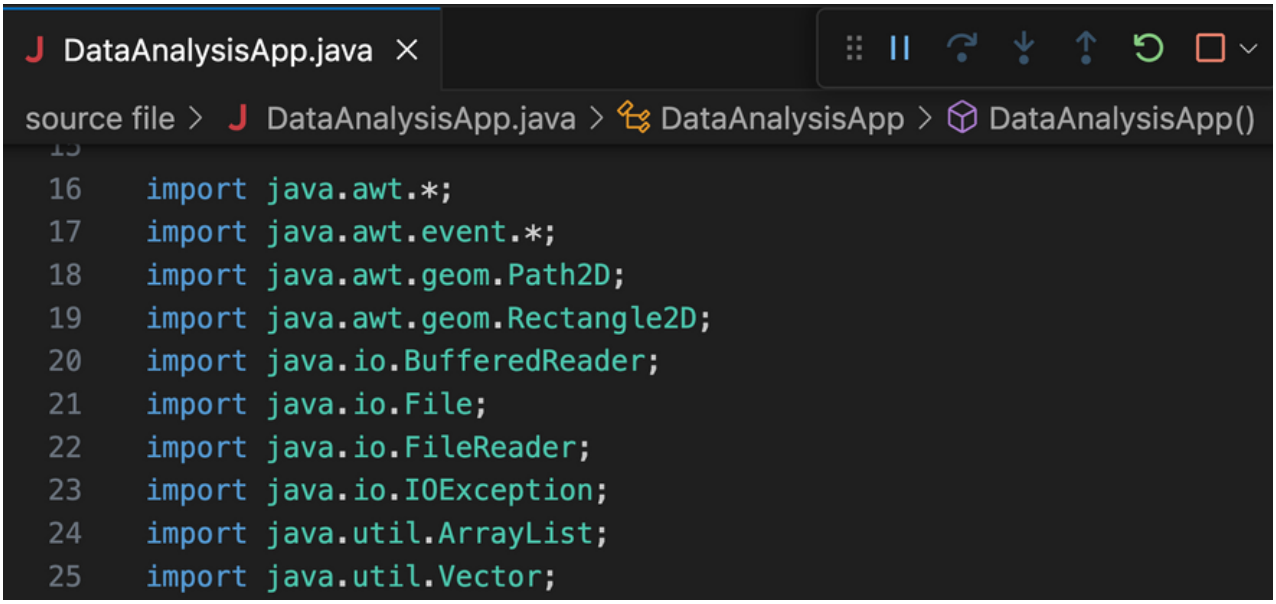


Part 1: Application Initialization :

I have successfully created a Java Swing application with a graphical user interface (GUI) window, achieving the first requirement. After imported the required libraries and functionalities, the application prominently displays a GUI window bearing the title "DataGuru" offering a user-friendly interface for data analysis.

To structure the application effectively, I utilized three Java classes, as specified in the requirements. The main class, ***DataAnalysisApp***, serves as the heart of the application, the creation and management of the GUI. The other two classes, ***CustomGraphicsPanel*** *and* ***Chatbot***, have been designed to manage custom graphics and chatbot interactions, respectively. This structured approach not only enhances code organization but also makes it more maintainable.





DataAnalysisApp.java

source file > DataAnalysisApp.java > DataAnalysisApp > DataAnalysisApp()

```
27  public class DataAnalysisApp {
28      private JFrame frame;
29      private JTable dataTable;
30      private JButton loadButton;
31      private JButton resetButton;
32      private JButton barChartButton;
33      private JButton pieChartButton;
34      private JButton lineChartButton;
35      private JComboBox<String> columnSelector;
36      private ChartPanel chartPanel;
37      private CustomGraphicsPanel customGraphicsPanel;
38      private JPanel chatPanel;
39      private JTextArea chatTextArea;
40      private JTextField chatInputField;
41      private Chatbot chatbot = new Chatbot();
```

DataAnalysisApp.java

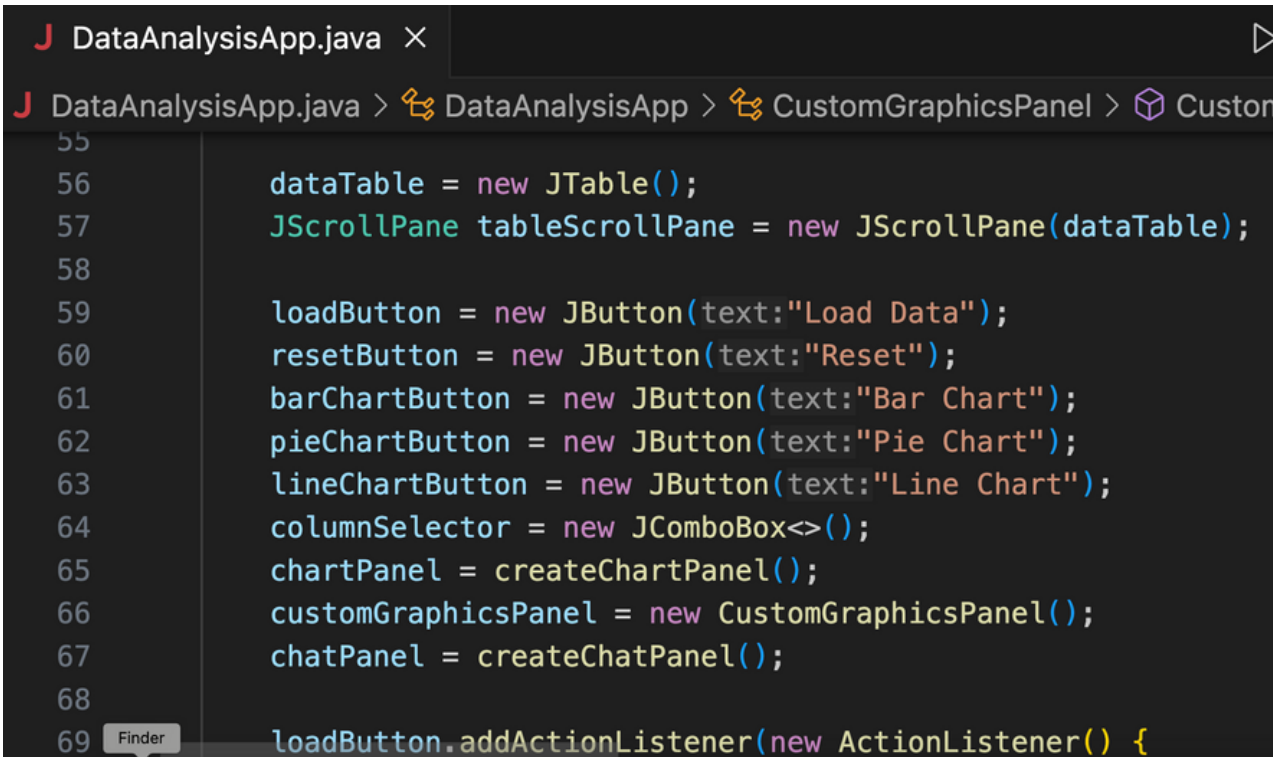
DataAnalysisApp.java > DataAnalysisApp > CustomGraphicsPanel > CustomGraphic

```
479
480     class CustomGraphicsPanel extends JPanel {
481         private ArrayList<Shape> customShapes;
482         private Path2D currentPath;
483
484         public CustomGraphicsPanel() {
485             customShapes = new ArrayList<>();
486             setPreferredSize(new Dimension(width:750, height:150));
487
488             addMouseListener(new MouseAdapter() {
489                 @Override
490                 public void mousePressed(MouseEvent e) {
491                     currentPath = new Path2D.Double();
492                     currentPath.moveTo(e.getX(), e.getY());
```

Part 2: Creating GUI Components using Swing:

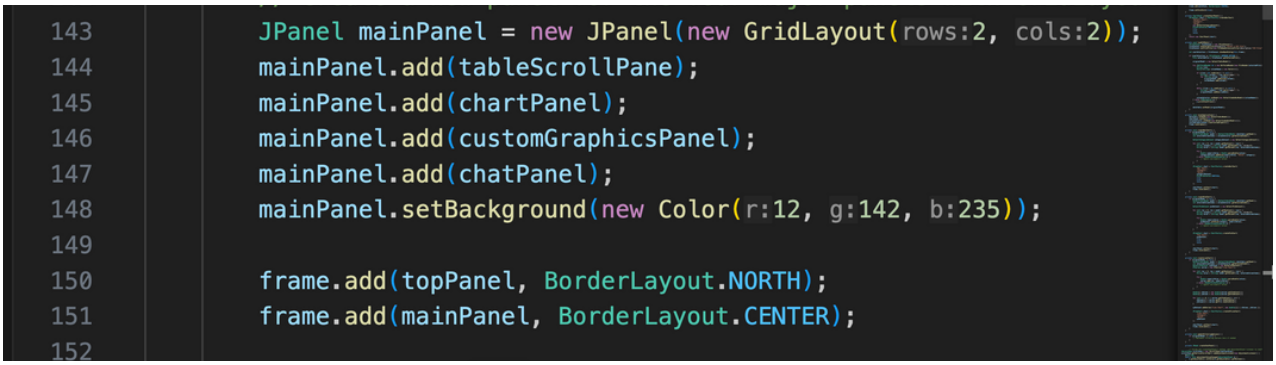
In response to the second requirement, I thoughtfully integrated a range of Swing components into the application's interface. The primary components employed include ***JTable***, ***JButton***, and ***JComboBox***. These components play a pivotal role in enabling users to ***load, analyze, and visualize data*** effortlessly.

All these components have been seamlessly integrated into the ***GUI window***, providing a cohesive and intuitive user experience. From the data table presented using ***JTable to the buttons for data loading and chart creation (implemented using JButton),*** these elements are visible, functional, and enhance the application's usability.



Part 3: Layout Managers :

To structure the GUI and align components optimally, I chose to use the ***BorderLayout*** layout manager for the main application layout. Additionally, the ***GridLayout manager*** was adopted to arrange components within the "mainPanel." The combination of these layout managers ensures that the GUI remains well-organized and responsive, adapting to various screen sizes and orientations. I have also used some other layout managers.



Part 4: Implementing Event Handling

Event handling is a critical aspect of the application, and I have successfully implemented it for various types of events and listeners, aligning with the fourth requirement.

1. **ActionEvent:** The application responds to button clicks, such as those for loading data, resetting the interface, and generating different types of charts. This allows users to interact with the application seamlessly.
2. **AdjustmentEvent:** The scroll bar in the chat panel responds to user interactions. The scrollbar ensures that users can navigate through chat history efficiently.
3. **FocusEvent:** To enhance user experience, I implemented focus-related events. For example, the chatInputField receives focus when the chat panel is accessed, simplifying user interactions.
4. **ItemEvent:** Users can select a column for analysis using the columnSelector component, demonstrating the handling of item selection events.
5. **KeyEvent:** The chat functionality is optimized to respond to key presses, primarily detecting the Enter key for sending chat messages.
6. **MouseEvent:** The application allows users to create custom graphics by responding to mouse events such as mouse presses and drags.

```
414             public void actionPerformed(ActionEvent e) {
415                 customGraphicsColor = JColorChooser.showDialog(component
416                 repaintCustomGraphics();
417             }
```

```
430             public void stateChanged(ChangeEvent e) {
431                 customGraphicsStroke = new BasicStroke(thicknessSlider.g
432                 repaintCustomGraphics();
433             }
```

```
490             public void mousePressed(MouseEvent e) {
491                 currentPath = new Path2D.Double();
492                 currentPath.moveTo(e.getX(), e.getY());
493             }
```

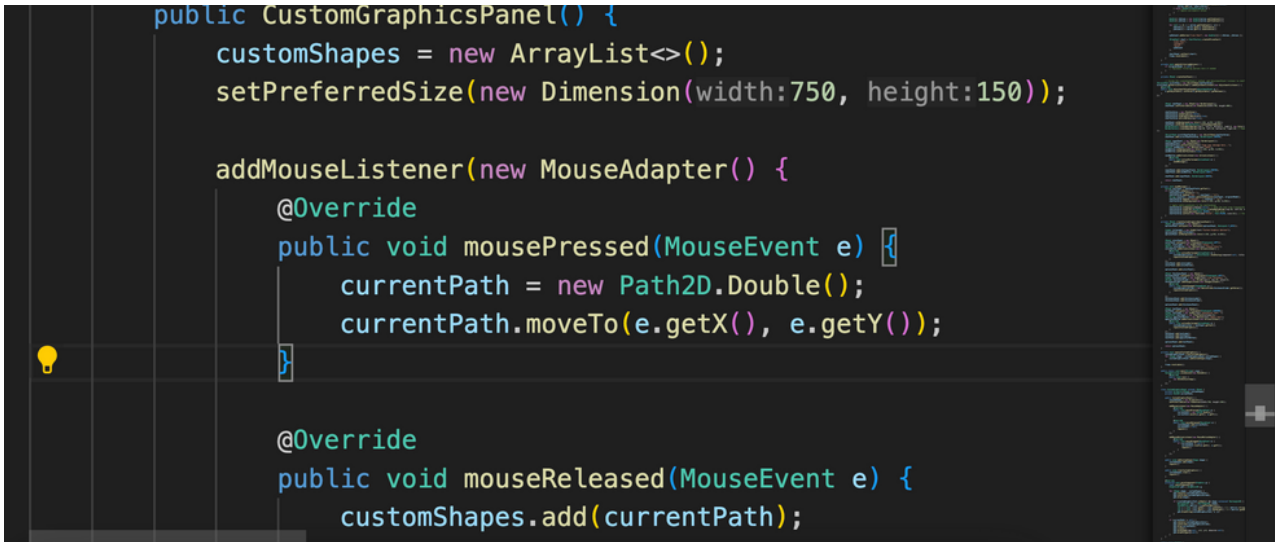
```
104         columnSelector.addItemListener(new ItemListener() {
105             @Override
106             public void itemStateChanged(ItemEvent e) {
107                 if (e.getStateChange() == ItemEvent.SELECTED) {
108                     updateFilteringOptions();
109                 }
```

```
120 chatInputField.addFocusListener(new FocusAdapter() {
121     @Override
122     public void focusGained(FocusEvent e) {
123         chatInputField.requestFocus();
124     }
```

```
335     public void adjustmentValueChanged(AdjustmentEvent e) {
336         e.getAdjustable().setValue(e.getAdjustable().getMaximum());
```

Part 5: Creating Custom Graphics

In this Part I have added Custom Graphic features that enables users to engage with custom graphics, fulfilling the requirements and enhancing the application's capabilities.
I incorporated a ***CustomGraphicsPanel class*** that empowers users to ***draw shapes, lines, and add custom text annotations*** within the custom graphics panel. This feature introduces a creative dimension to data analysis and visualization. Users can interact with different types of ***graphical elements***, including ***freehand writing,*** **lines and text annotations**, thereby providing an additional layer of expression and contextualization to their analyses.



```
public CustomGraphicsPanel() {
    customShapes = new ArrayList<>();
    setPreferredSize(new Dimension(width:750, height:150));

    addMouseListener(new MouseAdapter() {
        @Override
        public void mousePressed(MouseEvent e) {
            currentPath = new Path2D.Double();
            currentPath.moveTo(e.getX(), e.getY());
        }

        @Override
        public void mouseReleased(MouseEvent e) {
            customShapes.add(currentPath);
```

Conclusion:

In conclusion, the Data Analysis Application "DataGuru" provides a user-friendly and feature-rich environment for data analysis specially generating charts, complemented by the incorporation of custom graphics and interactive event handling. This application stands as a testament to effective Java Swing application development, demonstrating a structured approach to GUI design and coding.