Report on the Mini Project

Topic: Demonstration of sorting algorithms on an array

Algorithms: Merge sort, radix sort, counting sort

Team 19

Trần Thị Loan–20215080
Lê Phương Nam –20210628
Vũ Đình Linh – 20194606
Chu Anh Lợi – 20215280

Course: Object-oriented Programming 143577 – 2023.1
Dr. Nguyen Thi Thu Trang
06/01/2024

ACKNOWLEDGEMENT

We, the members of 19, come together to express our gratitude for the guidance and support received during the completion of our small project on "Demonstration of Sorting Algorithms on an Array".

First, we would like to thank Ms. Nguyen Thi Thu Trang, our teacher, for her profound advice and encouragement. Her expertise was instrumental in getting us on track and ensuring the success of our project.

We would also like to thank Mr. Le Thanh Giang, teaching assistant, for his valuable contributions and support throughout the project. His feedback and suggestions have played a vital role in improving the overall quality of our work.

We are grateful for the opportunity to work on this project as a team.

Together, we learned and grew through this project, and we would like to thank several of our classmates for their time and effort in helping us answer some of the questions that supported our report.

TABLE OF CONTENT

SUMMARY

Our project is focused on visualizing three sorting algorithms – Merge Sort, Counting Sort, and Radix Sort – through bar charts. The primary objectives are to fulfill our class assignments and develop a practical Java Swing application. This application aims to provide a useful tool for programmers, allowing them to visualize sorting algorithms in a colorful and interactive manner.

The report encapsulates the key aspects of our project, highlighting the contribution list, project overview, and design. The overview section delves into the project's requirements and includes a use case diagram illustrating the interaction between users and the application. The design section outlines the overall structure of the project, detailing the relationships between different components. It provides insights into each package, including classes, methods, attributes, and explanations of their interconnections.

In conclusion, our report will also showcase the Object-Oriented Programming (OOP) techniques employed throughout the project within the scope of our knowledge. This comprehensive summary encapsulates the main points, objectives, and technical aspects of our endeavor.

1. INTRODUCTION

In the realm of Computer Science, sorting is a foundational concept indispensable for programmers. It involves the arrangement of data sets sharing similar properties, and globally, diverse sorting algorithms with distinct implementations and strategies have been devised. Nevertheless, the comprehension of these algorithms can pose challenges, prompting the creation of a Java application as a viable solution. This application serves to visualize the steps of three specific sorting algorithms: Merge Sort, Counting Sort, and Radix Sort. The primary objective is to enhance accessibility and engagement in understanding sorting algorithms, aligning with the requirements of our class mini-project.

This report meticulously examines every facet of our project, encompassing team assignments, project descriptions, and design elements. The assignment segment acts as a grading guide, delineating individual contributions and providing essential identification information for team members. Furthermore, the project description elucidates features, requirements, and includes a use case diagram with detailed explanations and notes. The design section offers an encompassing perspective through a general class diagram and scrutinizes each package within it, presenting methods, attributes, and showcasing the application of Object-Oriented Programming (OOP) techniques employed in the project

2. ASSIGNMENT OF MEMBER

Member	Tran Thi Loan(Leader)	Le Phuong Nam	Vu Dinh Linh	Chu Anh Loi
Information	20215080 Loan.tt215080@sis.hust.edu.vn	20210628 Nam.lp210628@sis.hust.edu.vn	20194606 Linh.vd194606@sis.hust.edu.vn	20215280 Loi.ca215280@sis.hust.edu.vn
Work	- Package buttons - Package button - Package color - Package canvas - Class MergeSort - Report writing - *Pull request feedback: Assess whether coding following the agreed OO schema and conventions.*	- Package visualizer - Class Sort - Class MainFrame - UML Diagram - README.md (video) - *Pull request feedback: Run developing programs and perform actions to test and report uncovered issues.*	- Package bars - Package formatter - Class CountingSort - Presentation designer - *Pull request feedback:* *Assess whether new additions have conflicts with the old ones. (e.g. a package class diagram is different from general ones)*	- Package mainMenuApplication - Class RadixSort - Presentation design - *Pull request feedback: Professional knowledge is not good, making slides still needs a lot of editing*

3. MINI PROJECT DESCRIPTION

3.1. Project Overview

The mission of our project is to build an application that visualizes three sorting algorithms, namely merge sort, counting sort, and radix sort.

Due to the main purpose of visualization is to help users get a better insight into how an algorithm works, we have put some restrictions on our application:

1. Only non-negative (>0) numbers are allowed to be an array's element.

2. The array size used for visualization has a max size of 200 elements.

3. A valid array in Radix Sort, Merge Sort, Counting Sort only has the max value is form 30 to 350.

3.2. Project Requirement

For a better understanding and visualization, we have added some features to our application:

1. On the main menu: The main menu is designed to provide users with a simple and intuitive way to navigate through the sorting visualizer application. The components of the interface include:

- Title Label:Positioned at the top of the interface, it prominently displays the application's title, "Sorting Visualizer," in a bold and large font.

- Sort Button:Labeled "Sort," this button allows users to initiate the sorting demonstration. Upon clicking, it opens a new frame (MainFrame) for the selected sorting algorithm, and the current menu frame is disposed of.

- Help Button:Labeled "Help," this button triggers the display of a help dialog providing users with essential information about sorting algorithms and the functionality of the application.

- Quit Button:Labeled "Quit," this button prompts users with a confirmation dialog before exiting the application. If the user confirms, the application is terminated using System.exit(0).

2. In the demonstration:

In the sorting demonstration interface, users are presented with various components allowing for interactive exploration and visualization of sorting algorithms.

- Back to Menu Button: Located at the top-left corner, this button enables users to return to the main menu, offering seamless navigation between the sorting demonstration and the main application.

- Buttons Panel: Positioned on the left side, the buttons panel provides a set of controls for the user to interact with the sorting algorithms. Options include creating a random array and selecting different sorting algorithms such as Radix Sort, Counting Sort, Merge Sort, and a back button to return to the main menu.

- Canvas: Serving as the visualization area for the sorting algorithms. It dynamically adjusts its size based on the available space.

- Input Fields and Sliders: Include input fields for adjusting the capacity of the array and a slider for controlling the frames per second (FPS) to regulate the speed of the sorting visualization.

- Statistics Panel: Displays real-time statistics such as elapsed time, the number of comparisons, and the number of swaps during the sorting process. These statistics provide valuable insights into the efficiency and performance of the chosen sorting algorithm.

* User Interaction: The sorting demonstration interface encourages user engagement and exploration. Users can create random arrays, select different sorting algorithms, and observe the step-by-step visualization of the sorting process. The interactive components, including buttons and sliders, provide users with control over the demonstration's parameters, enhancing the learning experience.

3.3. Use case diagram and explanation

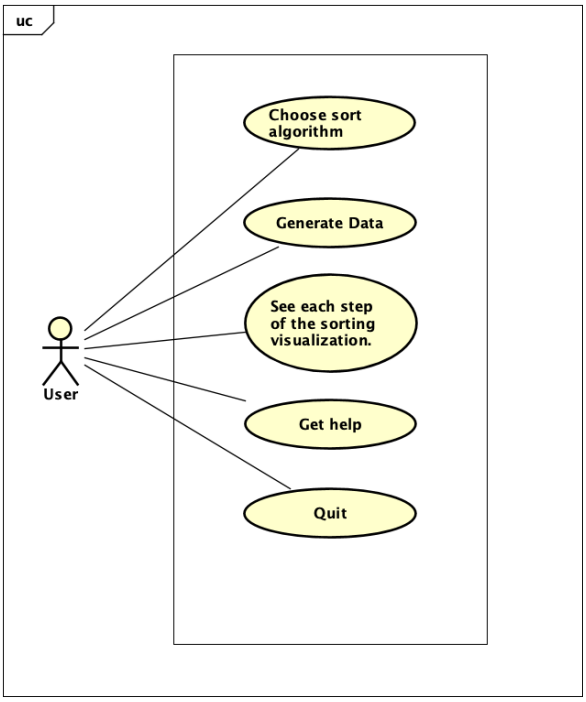

Figure 1 Use case diagram

Based on all the requirements we decided to develop five use cases (as shown in the figure for our application.)

To be more specific:

- Choose sort algorithm: Allow the user to choose an algorithm to sort with an array.

- Generate Data: Allow the user to create random array by the leftside button

- In the visualize algorithms use cases, the application will:

    + Run the visualization based on user command and see the steps in the sorting process

- View help menu uses case: Show the user manual and application restriction to the user under.
- Back to the main menu and quit : Backward to the main menu and exit the application.

1. DESIGN

1.1. General class diagram

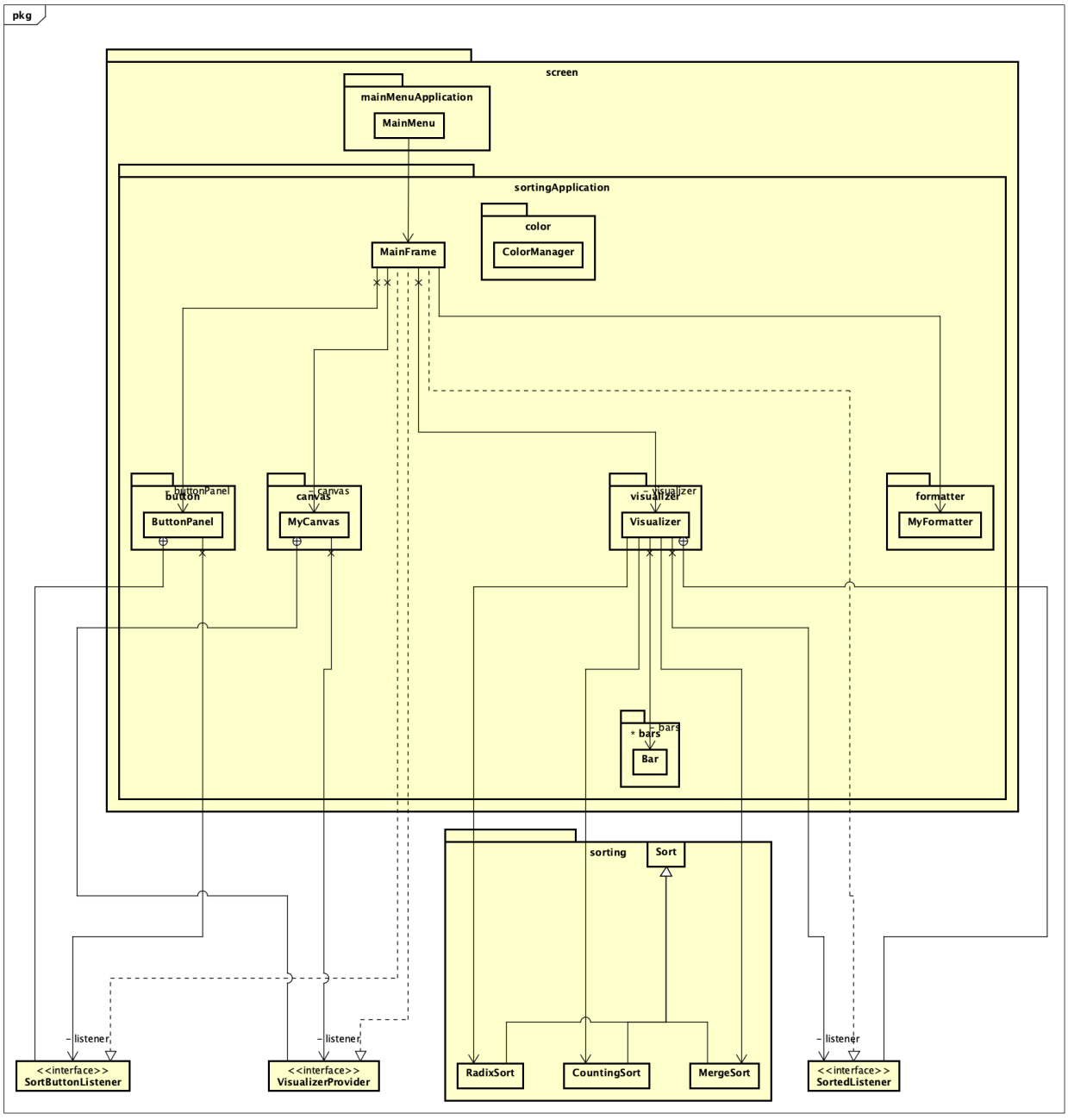

Figure 2: General Class Diagram

Package Name	Package Usage	Inside Class

sorting	Store three sorting algorithms.	Sort, MergeSort, RadixSort, CountingSort
mainMenuApplication	Store the main menu GUI application	MainMenu
button	Store the buttonPanel to generate all of button	ButtonPanel
canvas	Store the new canvas to override paint method of java.awt.Canvas	MyCanvas
color	Store all the colors to style programming	colorManager
bars	Store the bar of graph to visualize value of each element array	Bar
formatter	Store the format for capacity to override stringToValue method	MyFormatter
visualizer	Visualize sorting algorithms.	Visualizer
buttons	Store all assets of button icons.	

1.2. Detailed class diagram of each package

1.2.1. Package sorting

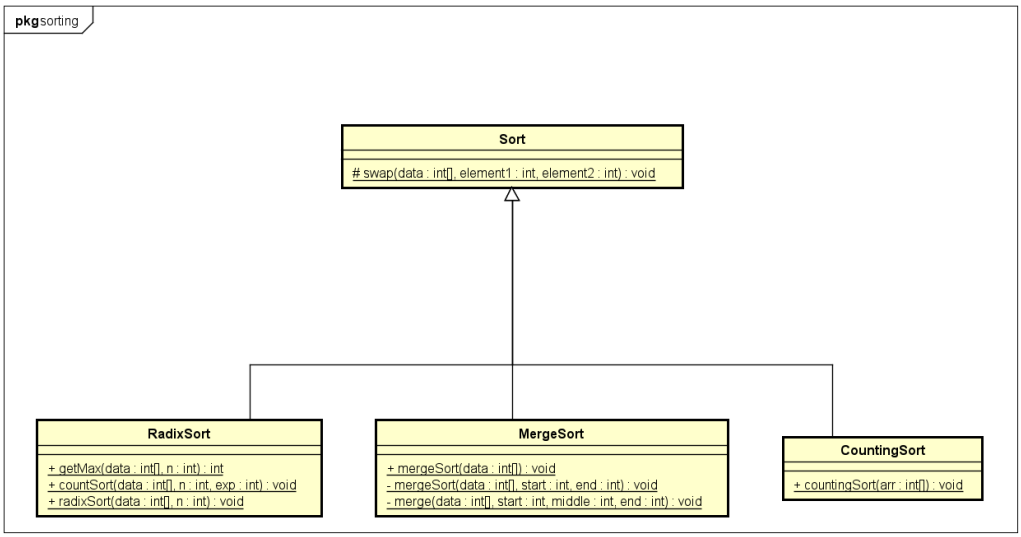

Figure 3 Sorting Package Class Diagram

Class Name	Class Usage	Method	OOP Technique
Sort	An abstract class, generalization of RadixSort, CountingSort and MergeSort.	# swap(data : int[],element1: int element2: int) : void	
RadixSort	Radix Sort Algorithm	+ getMax(data : int[], n : int) : int + countSort(data : int[], n : int, exp : int) : void + radixSort(data : int[], n : int) : void	Inherit from Sort (Inheritance)
CountingSort	Counting Sort Algorithm	+ countingSort(arr : int[]) : void	Inherit from Sort (Inheritance)
MergeSort	Merge Sort Algorithm	+ mergeSort(data : int[]) : void - mergeSort(data : int[], start : int, end : int) : void - merge(data : int[], middle : int, end : int) : void	Inherit from Sort (Inheritance)

1.2.2. Package mainMenuApplication

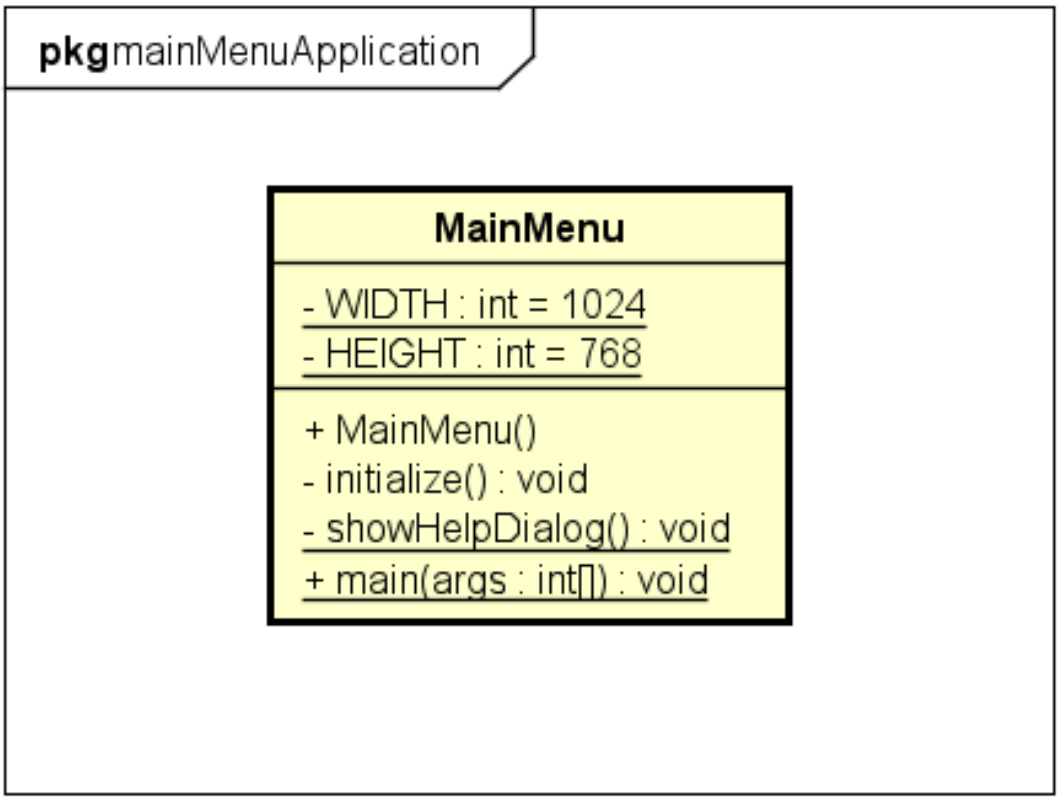

Figure 4 mainMenuApplication Package Class Diagram

Class Name	Class Usage	Attribute	Method	OOP Technique
MainMenu	A controller for BaseScreen	- WIDTH : int 1024 - HEIGHT : int = 768	+ MainMenu() - initialize() : void - showHelpDialog() : void + main(args : int[]) : void	

1.2.3. Class MainFrame of Package sortingApplication

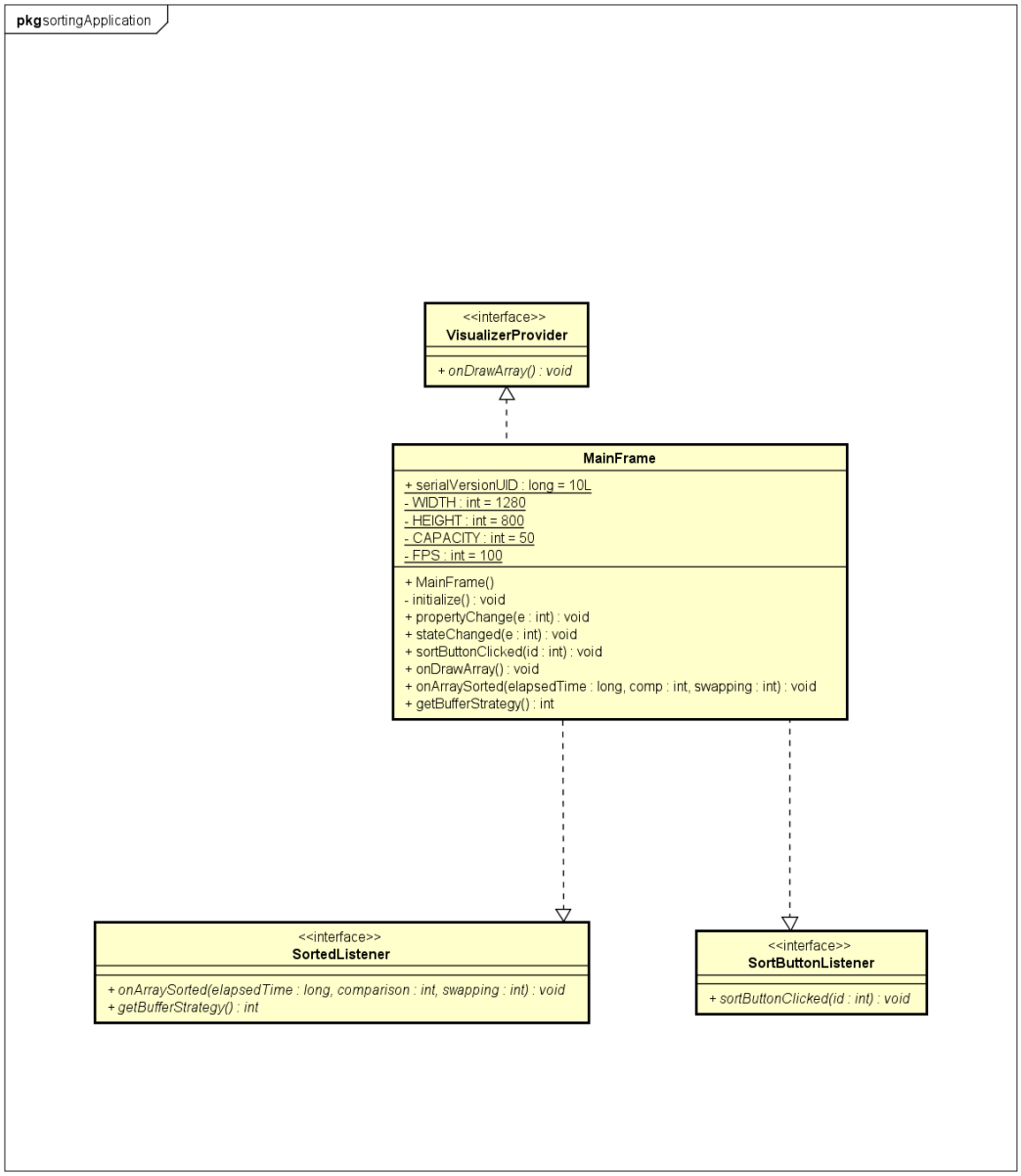

Figure 5 MainFrame Package Class Diagram

Class Name	Class Usage	Method	OOP Technique
MainFrame	Screen to visualize sorting algorithms	+ MainFrame() - initialize() : void + propertyChange(e : int) : void + stateChange(e : int) : void + sortButtonClicked(id : int) : void + onDrawArray() : void +onArraySorted(elapsedTime: long, comp : int, swapping : int) : void + getBufferStrategy() : int	Override method onArraySorted, getBufferStrategy, sortButtonClicked and onDrawArray
SortedListener	Displays real-time statistics such as elapsed time, the number of comparisons, and the number of swaps during the sorting process.	+ onArraySorted(elapsedTime : long, comparison : int, swapping : int) : void + getBufferStrategy() : int	
SortButtonListener	Listen command of user to sort	+ sortButtonClicked(id : int) : void	
VisualizerProvider	Re-draw bar array when user click minimize button	+ onDrawArray() : void	

1.2.4. Package bars

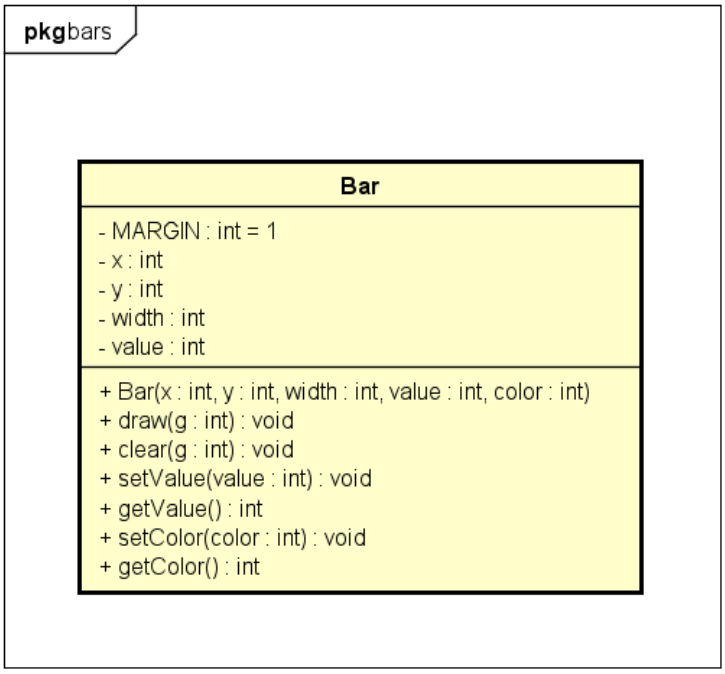

Figure 6 bars Package Class Diagram

Class Name	Class Usage	Attribute	Method	OOP Technique
Bar	Draw the bar in the screen	- MARGIN : int = 1 - x : int - y : int - width : int - value : int	+ Bar(x : int, y : int, width : int, value : int, color : int) + draw(g : int) : void + clear(g : int) : void + setValue(value : int) : void + getValue() : int + setColor(color : int) : void + getColor() : int	Inherit from awt.ActionListener (Inheritance)

1.2.5. Package button

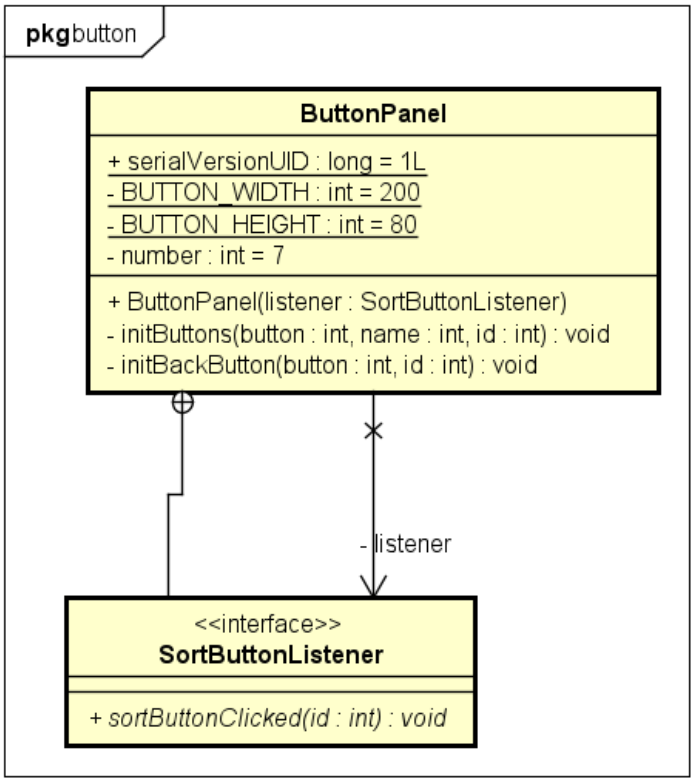

Figure 7 button Package Class Diagrams

Class Name	Class Usage	Attribute	Method
ButtonPanel	Generate button	+ serialVersionUID : long = 1L - BUTTON_WIDTH : int = 200 - BUTTON_HEIGHT : int = 80 - number : int = 7	+ ButtonPanel(listener : SortButtonListener) - initButtons(button : int, name : int, id : int) : void -initBackButton(button : int, id : int) : void - initBackButton(button : int, id : int) : void
SortButtonListener	Listen command of user to sort		+ sortButtonClicked(id : int) : void

1.2.6. Package visualizer

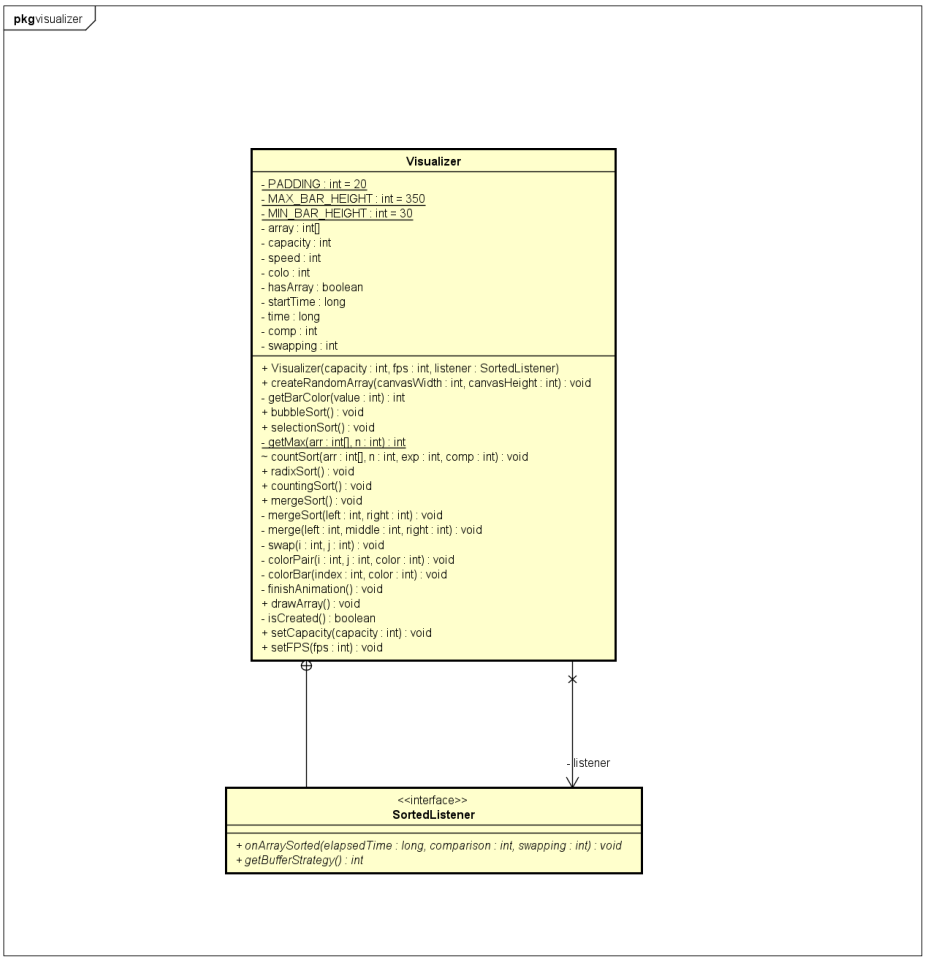

Figure 8 visualizer Package Class Diagram

Class Name	Class Usage	Attribute	Method
Visualizer	Visualize sorting	- PADDING : int = 20 - MAX_BAR_HEIGHT : int = 350 - MIN_BAR_HEIGHT : int = 30 - array : int[]	+ Visualizer(capacity : int, fps : int, listener : SortedListener) + createRandomArray(canvasWidth : int, canvasHeight : int) : void - getBarColor(value : int) : int + selectionSort() : void - getMax(arr : int[], n : int, exp : int, comp : int) : void

		- capacity : int - speed : int - colo : int - has Array : Boolean - startTime : long - time : long - comp : int - swapping : int	+ radixSort() : void + coutingSort() : void + mergeSort(): void - merge(left : int, middle : int, right : int) : void - mergeSort(left : int, right : int) : void - swap(i : int, j : int) : void - colorPair(i : int, j : int, color : int) : void - colorBar(index : int, color : int) : void - finishAnimation() : void + drawArray() : void - isCreated() : boolean + setCapacity(capacity : int) : void + setFPS(fps : int) : void
SortedListener	An exception called if input array is null or empty		+ onArraySorted(elapsedTime : long, comparison : int, swapping : int) : void + getBufferStrategy() : int

CONCLUSION

The Java Swing-based application has successfully met all the project requirements within the class. Our team has thoroughly tested its functionality, affirming its runnability. The visualization aspect, particularly for Merge Sort, Counting Sort, and Radix Sort, has been implemented effectively. Visualization, being a versatile form of expression, can adopt various styles to convey ideas. In our case, the project successfully delivered a unique perspective on sorting algorithms through a Java application, offering insights beyond mere lines of code and syntax.

The application exhibits a well-organized design structured into five main packages. Throughout the implementation, several Object-Oriented Programming (OOP) techniques such as Inheritance, Polymorphism, Aggregation, and Composition have been employed within the scope of our knowledge.

Despite its success, the project still presents ample opportunities for enhancement. Areas for improvement include User Interface (UI) design, accessibility, application logic, performance optimization, and tool integration. Visualization serves as a crucial means of communication between algorithms and programmers. The overarching goal now is to further develop the project, aiming to attract a broader audience and facilitate a deeper understanding of these abstract algorithmic concepts.