

**PROJECT TITLE: PHONE BOOK APPLICATION**

|  |  |
| --- | --- |
| **Course** | **Data Structure and Algorithm (LAB)** |
| **Submitted to** | **Sir Rizwan Ali** |
| **Submitted by** | **Avinash Talreja** |
| **Course Code** | **CMC251** |
| **Date** | **June 12, 2025** |

**Concept:**

This project implements a phone book application using JavaFX for the graphical user interface (GUI) and emphasizes the use of appropriate data structures and algorithms to optimize performance and functionality. The application allows users to manage contacts (name and number), search, update, delete, export, and engage in simple chat functionality. The choice of data structures like LinkedHashMap and Deque, along with algorithms for searching and sorting, are crucial for the application's efficiency.

**Motivation:**

The application serves as a practical demonstration of how data structures and algorithms impact the performance and scalability of a real-world application. It allows developers to understand how choosing the right data structure (e.g., LinkedHashMap for efficient key-based access and insertion order) and implementing efficient algorithms (e.g., optimized search and filtering) can significantly improve the user experience. The inclusion of features like password protection and undo/redo highlights the importance of efficient data management and algorithmic design.

**Problem Statement:**

The project addresses the problem of creating a user-friendly and performant phone book application with a GUI. It provides functionalities for adding, updating, deleting, searching, and exporting contacts. The application's performance is optimized by selecting appropriate data structures and algorithms for each operation. The goal is to create an application that is both responsive and scalable, even with a large number of contacts.

**Design / Ways & Means:**

**Introduction and Requirements:**

* The program is a JavaFX-based phone book application with features for managing contacts, searching, exporting, and chatting.
* The application provides a GUI for user interaction.

**Functional Requirements:**

* Add, update, and delete contacts (name and number).
* Search contacts based on name, number, or both.
* Export contacts to a CSV file (all or selected).
* Implement undo/redo functionality for contact management actions.
* Provide a simple chat interface for communication with contacts.
* Password protection for application access.

**Non-Functional Requirements:**

* User-friendly GUI.
* Secure password handling using SHA-256 hashing.
* Efficient contact searching and filtering.
* Robust error handling and validation.

**Data Structures and Algorithms Design:**

**Data Structures:**

* LinkedHashMap: Used to store contacts (name-number pairs). LinkedHashMap provides O(1) average time complexity for get, put, and remove operations, while also maintaining insertion order, which is useful for displaying contacts in the order they were added.
* ObservableList: Used to hold contact data for the TableView. ObservableList allows the TableView to automatically update when the underlying data changes.
* Deque (ArrayDeque): Used to implement the undo/redo functionality. Deque provides O(1) average time complexity for push, pop, and peek operations, making it efficient for managing the history of actions.

**Algorithms:**

* **Searching:** The filterContacts method implements a search algorithm that filters contacts based on a search query. The algorithm iterates through the contactMap and checks if the name or number contains the search query. The time complexity of this algorithm is O(n), where n is the number of contacts. Optimization can be achieved using indexing techniques (e.g., using a Trie or inverted index) for faster lookups, especially with a large dataset.
* **Sorting:** The refreshContacts method sorts the contacts alphabetically by name using Map.Entry.comparingByKey(String.CASE\_INSENSITIVE\_ORDER). This uses a comparison-based sorting algorithm (likely a variant of merge sort or quicksort in Java's implementation), which has an average time complexity of O(n log n), where n is the number of contacts.
* **Hashing:** The hashPassword method uses SHA-256 hashing to securely store passwords. Hashing provides O(1) average time complexity for password verification.

**Basic Implementation:**

* The core functionalities (add, update, delete, search, export) are implemented using JavaFX components and data structures.
* CRUD operations are performed on the contactMap and reflected in the TableView.
* The undoStack and redoStack are used to implement undo/redo functionality.
* Password protection is implemented using SHA-256 hashing and file storage.
* The chat functionality allows users to send and receive messages, which are stored in separate files.

**Performance Testing and Analysis:**

* The application's performance can be analyzed by measuring the time taken to perform CRUD operations, search contacts, and export data.
* The efficiency of the search algorithm can be evaluated by measuring the time taken to filter contacts based on different search criteria.
* The memory usage of the application can be monitored to identify potential memory leaks or inefficiencies.
* The choice of LinkedHashMap provides efficient key-based access, while the Deque data structure enables efficient undo/redo operations. The sorting algorithm used for displaying contacts impacts the overall performance, and alternative sorting algorithms could be explored for optimization.

**Optimization and Advanced Features:**

* **User Interaction:** The GUI provides a user-friendly interface for interacting with the application.
* **Input Handling:** User input is handled through TextFields and validated before being processed.
* **Input Validation:** Phone numbers are validated to ensure they contain only digits, spaces, +, or -.
* **Search Modes:** The application supports different search modes (name, number, or both) using radio buttons.
* **Undo/Redo:** The undo/redo functionality allows users to revert or reapply changes.
* Chat Functionality: The chat functionality allows users to communicate with contacts.
* **Password Protection:** The password protection feature ensures that only authorized users can access the application.
* **Error Handling:** The application includes error handling to prevent crashes and provide informative messages to the user.

**Extensions and Creativity**:

The application can be extended to include more advanced features, such as:

* Adding support for multiple phone numbers per contact.
* Implementing a more sophisticated search algorithm (e.g., using a Trie or inverted index) for faster lookups.
* Adding support for importing contacts from other sources (e.g., CSV files, Google Contacts).
* Implementing a more robust chat interface with features like file sharing and group chat.
* Adding support for cloud storage and synchronization.
* Implementing a more secure password storage mechanism (e.g., using salting).

**Conclusion:**

This project demonstrates the effective use of data structures and algorithms to create a performant and user-friendly phone book application. The choice of LinkedHashMap for efficient key-based access, Deque for undo/redo, and the implementation of optimized search and sorting algorithms contribute to the application's overall efficiency. The project provides a valuable learning experience for developers interested in applying DSA principles to real-world problems. Further optimization can be achieved by exploring more advanced data structures and algorithms for specific functionalities, such as searching and sorting.