Abstract

The CLI Contact Manager is a console-based application developed using Java that provides users with a simple yet powerful tool to manage personal contact information efficiently. Designed with a focus on core software engineering principles, the project implements full CRUD (Create, Read, Update, Delete) functionalities for contact records. Each contact entry includes details such as name, phone number, blood group, email address, and place, which are initially stored in an ArrayList and persistently maintained in a MySQL database using JDBC (Java Database Connectivity).

This project aims to bridge the gap between in-memory data handling and real-world persistent storage, giving developers an integrated experience with object-oriented programming (OOP) concepts and database operations. The menu-driven interface allows users to interact with the system via the command line, offering options to insert new contacts, search for specific entries, update existing data, or delete unwanted records with ease. Additionally, the application emphasizes data consistency and ensures that all operations performed in the runtime memory are accurately reflected in the database.

From a developmental perspective, this project enhances skills in modular Java programming, JDBC API usage, exception handling, and database schema design. It provides a foundational understanding of building scalable and maintainable command-line applications, laying the groundwork for future expansion such as GUI integration or multi-user support. It is particularly suitable for beginners in software development looking to gain practical exposure to end-to-end application design and implementation.

Overall, the CLI Contact Manager represents a clean, efficient, and educational tool for managing contacts, while reinforcing essential programming techniques and offering a tangible demonstration of real-world Java-MySQL integration.

Objective

To build a user-friendly CLI-based Contact Manager that can:

- Add new contacts
- View and update saved contacts
- Search by name, number, place, blood group, or starting letter
- Store and fetch data from a MySQL database using arrays in Java
- Ensure data consistency by syncing changes between the array and the MySQL database during all operations.
- Implement a menu-driven interface that guides users through all contact management functions in a clear and intuitive manner.

Problem Statement

In today's digital era, managing contacts manually through diaries or unstructured notes is inefficient and error-prone. These traditional methods are time-consuming, lack standardization, and offer no easy way to search, update, or back up contact information.

This project solves these issues by introducing a CLI-based Contact Manager using Java and MySQL. It offers:

- Faster access and updates through a menu-driven interface
- Accurate and consistent data stored in arrays and a MySQL database

- In short, the Contact Manager replaces outdated practices with a fast, accurate, and maintainable digital solution.

Scope of the Project

- What our project includes and what it doesn't.

  * Supports only the admin user (no user login module)
  * Only tracks employee payroll (no leave/attendance module)
  * Focuses on CLI (no GUI)

- This project is designed to fulfill the core requirements of a contact management system within a console-based environment. It does not aim to implement advanced features like multi-user access, encryption, cloud integration, or UI frameworks, keeping its scope clear and achievable. The combination of in-memory operations and persistent database storage ensures both performance and reliability, making the CLI Contact Manager a practical tool for small to mid-sized contact databases.

Technology Stack

Layer/Component	Technology Used
Programming	Java (JDK 21)
Database Storage	Arrays (Java) + MySQL 8.0.36
DB Connectivity	JDBC
DB GUI Tool	MySQL Workbench 8.0
IDE	Eclipse IDE 2025-06
Version Control	Git 2.49.0
OS	Windows 11

Minimum Requirement

Layer/Component	Technology Needed
OS	Windows 7/Linus/macOS
Processor	Dual Core 1.5 GHz or above
RAM	2GB
DB GUI Tool	MySQL Workbench 8.0
JAVA	Java SE Development Kit (JDK) 8 or higher
Storage	250 MB free space (for JDK, DB, and project files)
Database	MySQL Server 5.7 or higher
Additional	MySQL Connector/J (JDBC driver)
IDE (Optional)	Eclipse / IntelliJ / VS Code

System Design & Architecture

Architecture as a top-down flow:

User Interface (CLI)

⇩

Service Layer (ContactService.java)

⇩

Application Logic (Java Arrays)

⇄

SQL Database (via JDBC Connector)

User Interface (CLI):
Provides a text-based interface for users to interact with the contact manager through a menu-driven system.

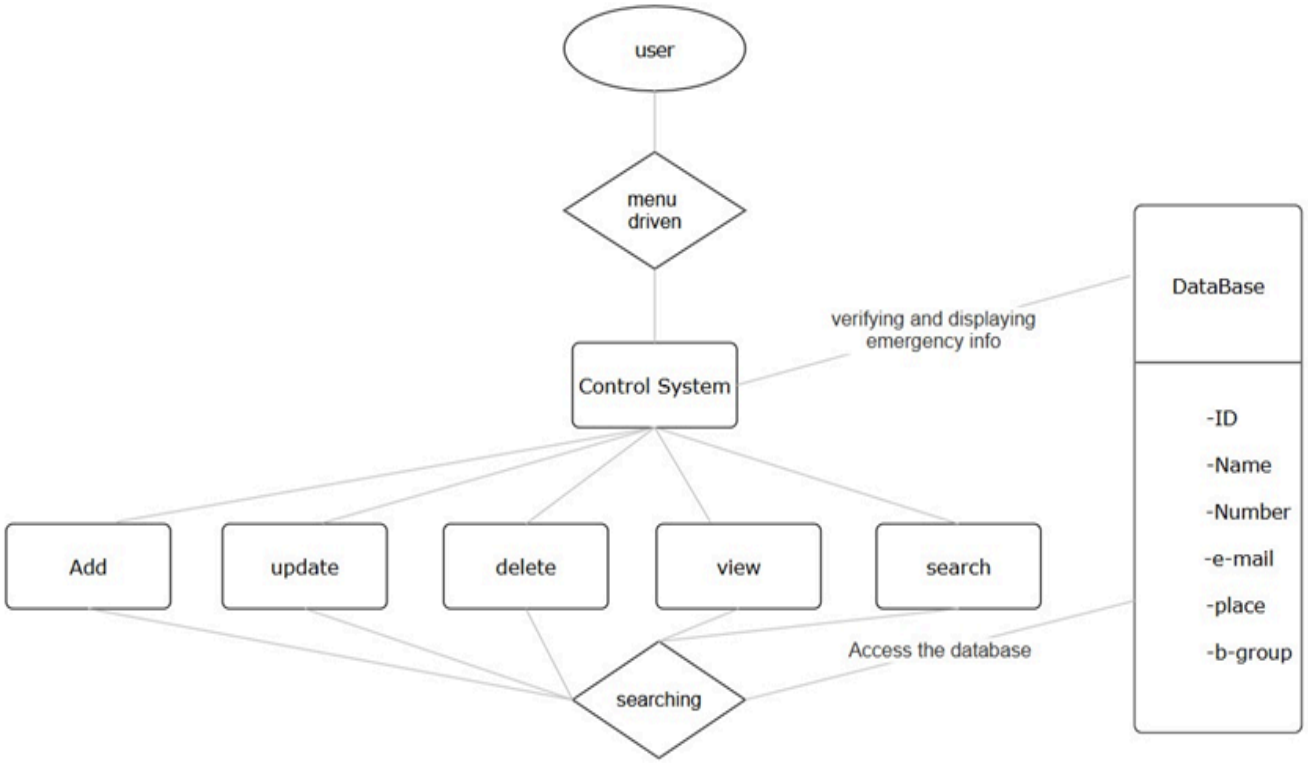
Service Layer (ContactService.java):
Acts as the bridge between the user interface and application logic, handling input validation and coordination of operations.

Application Logic (Java Arrays):
Stores and manages contact data temporarily in memory for fast access and manipulation during program execution.

SQL Database (via JDBC Connector):

Persists contact data permanently and communicates with the Java application using JDBC for all CRUD operations.

DataFlow Diagram



Modules Description

The CLI-Based Contact Manager application is structured around several functional modules that provide an interactive, efficient, and reliable way to manage contact information using Java arrays and a MySQL database. Each module corresponds to a specific task and is accessible through a user-friendly menu displayed in the command-line interface (CLI). Below is a detailed description of each key module:

1. Add Contact

The Add Contact module enables users to insert new contact records into the system. When selected from the menu, it prompts the user to enter the contact's Name, Phone Number, Email, Blood Group, and Place. Each input is validated to check for correctness—for example, ensuring the phone number is numeric and required fields are not left empty. A Contact object is then created and stored in a Java array to maintain the in-memory copy of data. Simultaneously, the contact is inserted into the MySQL database using an SQL INSERT query executed via JDBC. After insertion, the contact list is refreshed by reloading all data from the database, ensuring synchronization between the array and the persistent storage.

2. View All Contacts

This module provides a complete list of all stored contacts. When this option is chosen, the system iterates through the array and displays each contact's details—such as ID, Name, Phone, Email, Blood Group, and Place—in a clear format on the terminal. It allows users to review their records and is particularly useful for identifying the ID numbers needed for update or delete operations. This module ensures that all data shown is consistent with the most recent database state, as the array is regularly updated after any changes.

3. Search Contact

The Search Contact feature enhances data accessibility by enabling users to find specific records based on different criteria. The available search options include searching by Name, Phone Number, Blood Group, Place, or the First Letter of the Name. Once the user selects the search type and provides the corresponding keyword, the system scans the Java array and uses string comparison to locate matching entries. Results are displayed instantly in the terminal, making it easy to locate contact details without browsing the full list.

4. Update Contact

This module allows users to modify the information of an existing contact. The system first prompts the user to enter the ID of the contact they wish to update. It then displays the contact's current information for confirmation. Users are asked to enter new values for any fields they want to change. These updated values are saved both in the in-memory array and in the MySQL database using an SQL UPDATE query. Once the changes are applied, the application reloads the full contact list from the database to ensure both memory and storage are in sync.

5. Delete Contact

The Delete Contact module is responsible for removing a contact from the system. Users are prompted to enter the ID of the contact to be deleted. A confirmation step ensures accidental deletions are avoided. If confirmed, the system executes an SQL DELETE query through JDBC to remove the contact from the database. The in-memory array is then refreshed by reloading the contact data from the updated database, ensuring accuracy and integrity of displayed records.

6. Emergency Numbers (within Main Menu)

Although not implemented as a standalone module, the Emergency Numbers feature is integrated directly into the main menu of the application. When selected, it displays a list of important, hardcoded emergency contact numbers like Ambulance (108), Police (100), and Fire (101). These numbers are static and non-editable, offering users quick access to critical services during emergencies. Their inclusion in the main menu ensures accessibility without complicating the modular structure of the application.

Folder Structure

Folder Structure

- cli_contact_manager/
  - ├── app/
  - │ └── Main.java
  - ├── db/
  - │ └── ContactDatabase.java
  - ├── models/
  - │ └── Contact.java
  - ├── service/
  - │ └── ContactService.java
  - └── module-info.java

app/

The app module contains Main.java, which serves as the starting point of the application. It displays the menu and handles user interactions via the console.

db/

The db module contains ContactDatabase.java, which manages the database connection. It provides a centralized method to connect with MySQL using JDBC.

models/

The models module defines the Contact class, representing a contact's structure. It holds attributes like name, phone, email, and provides getter/setter methods.

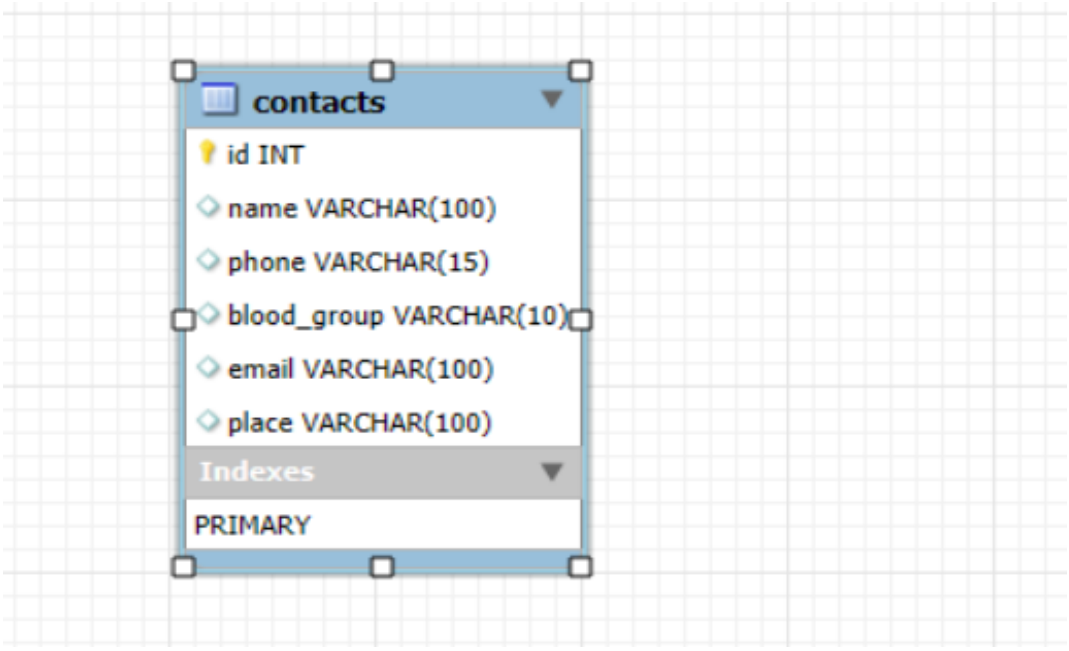
service/

The service module includes ContactService.java, where all core logic resides. It handles add, search, update, delete, and fetch operations using arrays and databases.

module-info.java

This file defines the project as a module in Java. It declares dependencies and organizes code structure in a modular format (optional in small projects).

ER Diagram



Entity Definition: The diagram defines a single entity called contacts, which represents each individual saved in the contact manager system.

Attributes and Data Types: The contacts table includes six attributes — id (Primary Key, INT), name, phone, blood_group, email, and place (all VARCHAR types with appropriate length for each field).

Primary Key Constraint: The id field uniquely identifies each contact entry, ensuring data integrity and enabling efficient retrieval, update, and deletion operations in the application.

Implementation Details

The Contact Manager project uses a fixed-size array of 100 elements to temporarily store contact data in memory. All contact details are also stored permanently in a MySQL table named contacts, which holds fields like name, phone, email, blood group, and place. The project performs core database operations such as INSERT, SELECT, UPDATE, and DELETE using JDBC (Java Database Connectivity), which connects the Java application to the MySQL database and enables data manipulation.

Pseudocode:

Simple Step-by-Step Logic of How the Contact Manager Works :

Start
  Load contacts from DB into array
  Show menu
  Perform action (add, search, etc.)
  Update array and DB
End

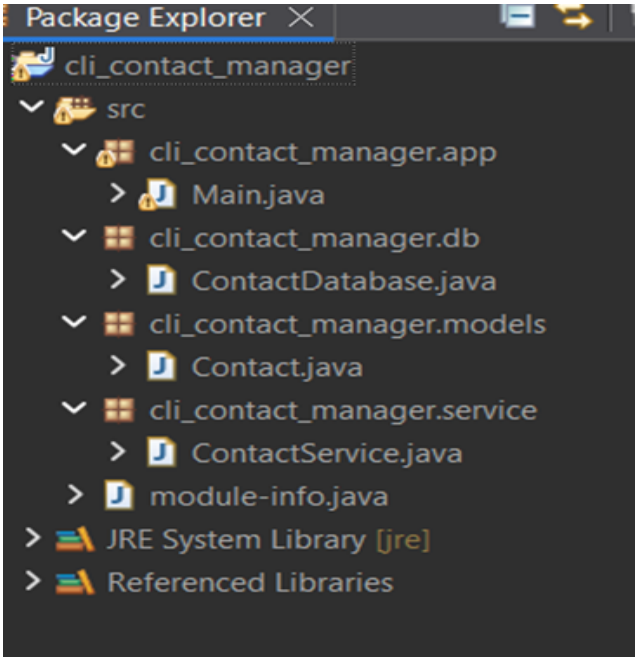
Learnings

- Java OOPs concepts

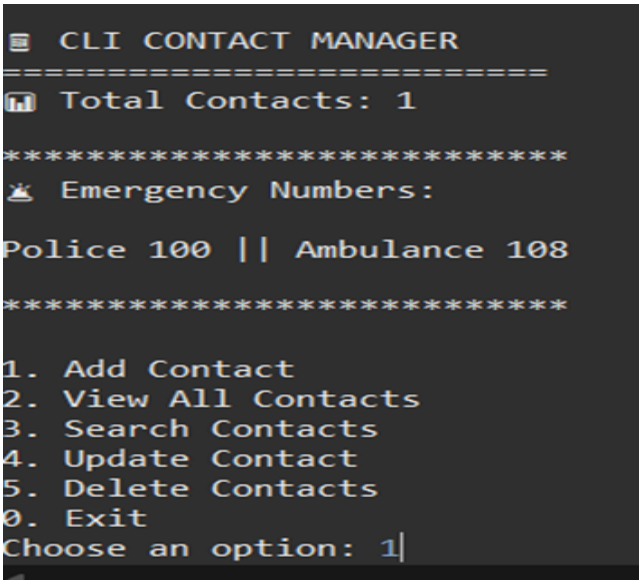
- JDBC and database management
- Search/filtering logic using arrays
- Debugging SQL queries

Sample Screenshots / Outputs

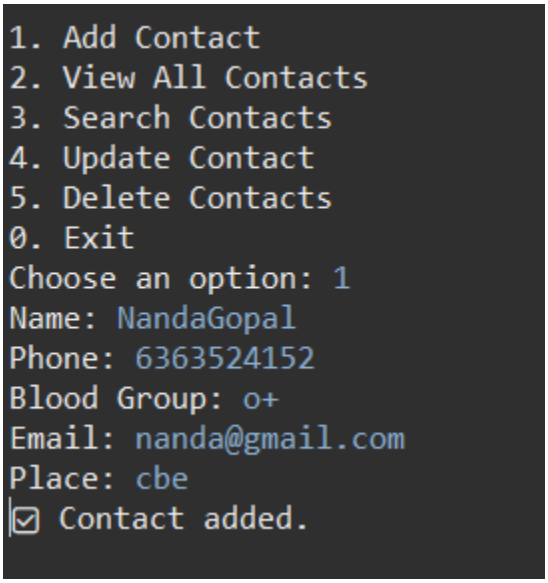
1.Folder Structure



2.Main Menu View



3. Add contact



4. View contact

```
1. Add Contact
2. View All Contacts
3. Search Contacts
4. Update Contact
5. Delete Contacts
0. Exit
Choose an option: 2

🖨 Contact List:

----------------------------
ID          : 1
Name        : Gowdham G
Phone       : 9629000315
Blood Group : o+
Email       : gowtham21@gmail.com
Place       : cbe

----------------------------
ID          : 2
Name        : Gowthamen J
Phone       : 6363524152
Blood Group : ab
Email       : gowthaman@gmail.com
Place       : kovilpalayam

----------------------------
ID          : 3
Name        : NandaGopal
Phone       : 6363524152
Blood Group : o+
Email       : nanda@gmail.com
Place       : cbe
```

5.Search Contact

```
1. Add Contact
2. View All Contacts
3. Search Contacts
4. Update Contact
5. Delete Contacts
0. Exit
Choose an option: 3

=== Search Menu ===
1. By Name
2. By Phone
3. By Blood Group
4. By Place
5. By First Letter of Name
Enter option: 5
Enter first letter: N

----------------------------
ID          : 3
Name        : NandaGopal
Phone       : 6363524152
Blood Group : o+
Email       : nanda@gmail.com
Place       : cbe
```

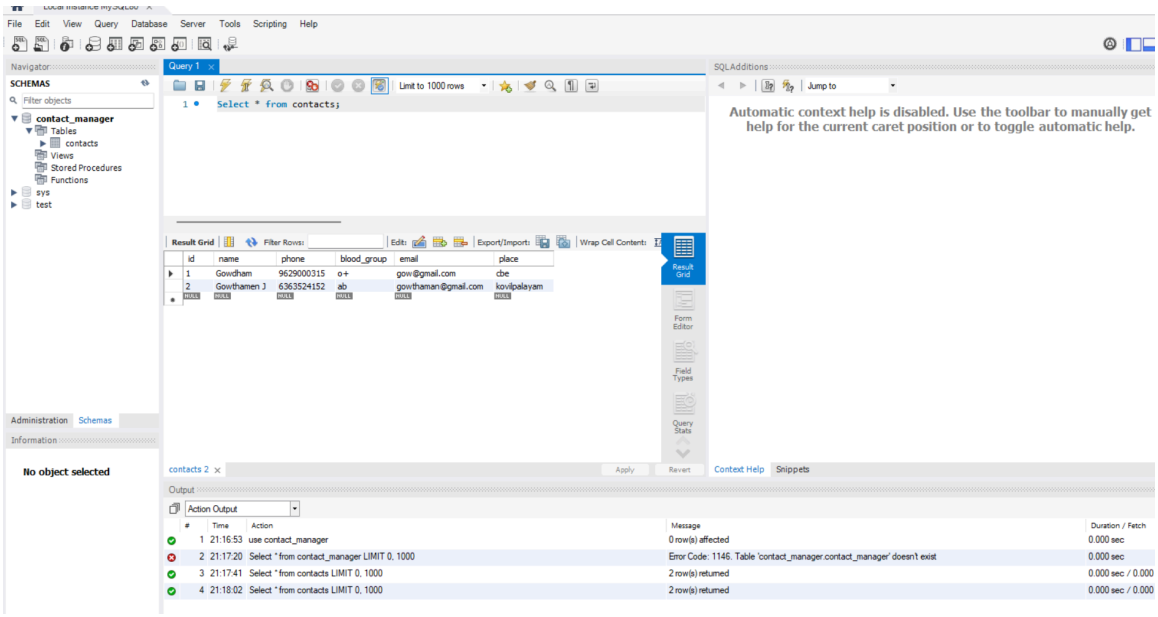
6.Updating Contact

```
1. Add Contact
2. View All Contacts
3. Search Contacts
4. Update Contact
5. Delete Contacts
0. Exit
Choose an option: 4
Enter ID to update: 1
Editing:

----------------------------
ID           : 1
Name         : Gowdham G
Phone        : 9629000315
Blood Group  : o+
Email        : gowtham21@gmail.
Place        : cbe
New Name: Gowdham
New Phone: 9629000315
New Blood Group: o+
New Email: gow@gmail.com
New Place: cbe
☑ Contact updated.
```

7.Update Contact

```
1. Add Contact
2. View All Contacts
3. Search Contacts
4. Update Contact
5. Delete Contacts
0. Exit
Choose an option: 5
Enter Contact ID to delete: 3
Are you sure you want to delete this contact?

---------------------------
ID          : 3
Name        : NandaGopal
Phone       : 6363524152
Blood Group : o+
Email       : nanda@gmail.com
Place       : cbe
Type 'yes' to confirm: yes
☑ Contact deleted successfully.
```

8.Database



Testing & Validation

- All modules tested manually
- Contacts fetched from DB match array data
- Console messages confirm status of each action
- Search functions tested with multiple fields (name, place, etc.)
- Update and delete tested using unique contact IDs to verify correct data changes

Challenges Faced

- ID mismatch after deletion due to auto-increment
- SQL connection exceptions
- Maintaining sync between array and database
- Handling empty or invalid user inputs during operations

- Managing fixed array size (limited to 100 contacts), which required checks before adding new entries

Appendix

1. SQL Schema:

```
CREATE DATABASE contact_manager;

USE contact_manager;

CREATE TABLE contacts (

   id INT AUTO_INCREMENT PRIMARY KEY,

   name VARCHAR(100),

   phone VARCHAR(15),

   blood_group VARCHAR(10),email VARCHAR(100),

   place VARCHAR(100));
```

Additional Notes:

- Maximum of 100 contacts can be stored in the array.
- The array is refreshed every time a DB operation occurs.
- Project tested in Eclipse IDE using Java 17 and MySQL 8.0.
- No external frameworks/libraries were used.

References

- https://www.w3schools.com/sql/
- https://docs.oracle.com/javase/
- https://dev.mysql.com/doc/

Conclusion

The CLI Contact Manager project successfully meets its objective of managing contacts using Java, arrays, and MySQL. It allows users to add, view, search, update, and delete contacts via a user-friendly console interface. The system ensures that data remains synchronized between the database and in-memory storage, demonstrating the integration of core Java concepts with JDBC.

For reference and future improvements, the complete source code is available on GitHub:

GitHub Repository: https://github.com/gowdham2106/cli_contact_manager