**4-2 Milestone Three: Enhancement Two: Algorithms and Data Structure**

Natalia A. Santiago

Southern New Hampshire University

CS-499: Computer Science Capstone

Federico Bermudez

February 2, 2025

**Overview of the Artifact**

The artifact selected for this enhancement is the contact manager application, which was originally created three months ago as part of my CS-320: Current Emerging Trends in Computer Science coursework. Initially, the program allowed basic operations such as adding, updating, and deleting contacts, but it faced several limitations: the contact data was stored only in memory (resulting in data loss after the program closed), and there was no user interface (requiring direct code interaction).

For this enhancement, I focused on improving the program by addressing key algorithmic components and data structures that would make it more efficient and scalable. These improvements involved implementing data persistence using a JSON-based database, adding a simple command-line interface (CLI), and improving data handling by optimizing the program's underlying data structure.

**Justification for Inclusion in the ePortfolio**

This artifact was chosen for inclusion in my ePortfolio because it effectively demonstrates my growing expertise in algorithms and data structures, particularly in solving real-world problems with these concepts. The enhancements I made to the contact manager program allowed me to deepen my understanding of key data structures and how they interact with algorithms to improve performance, reliability, and usability.

The specific components that showcase my skills in algorithms and data structures include:

* Data Persistence: By introducing JSON-based storage, I effectively used a serialization algorithm to persist the contact data across sessions. This enhanced the program's usability by ensuring that data was no longer lost when the program was closed, demonstrating an understanding of efficient data handling and storage techniques.
* Data Structures: Initially, the program used basic in-memory data structures (such as lists or dictionaries). However, I restructured the contact storage system to more efficiently handle CRUD operations by using advanced data structures that support faster lookups and modifications. This reflects a deeper understanding of data structures like hash maps and trees, which are often employed in real-world applications to optimize access speed.
* Command-Line Interface (CLI): The CLI improvement not only enhanced usability but also introduced a structure for managing commands and inputs, demonstrating my understanding of how algorithms interact with user input and output. The decision to implement the CLI in a way that supports easy expansion in the future also highlighted my ability to design scalable systems.

**Reflection on Meeting Course Outcomes**

I met several course outcomes related to algorithms and data structures in this enhancement:

1. Design and Evaluate Computing Solutions: The switch to a JSON-based database for persistence required evaluating different data storage solutions and selecting the best one based on the application’s needs. I demonstrated my ability to make thoughtful design decisions, weighing the trade-offs between simplicity and efficiency.
2. Algorithmic Principles: The modifications to the underlying data structure for contact storage reflected my ability to use algorithmic principles to optimize performance. For example, by using more efficient data structures like hash maps for searching and storing contacts, I improved the speed and reliability of the application’s operations.
3. Data Structure Optimization: I improved the handling of user inputs and data storage by selecting the most appropriate data structures for the program’s needs, optimizing performance. This included implementing validation checks and error-handling algorithms to ensure the program behaves predictably and efficiently.

**Reflection on the Enhancement Process**

The process of enhancing this application was both a challenge and an opportunity to apply my knowledge of algorithms and data structures in a practical context. One of the initial challenges was choosing the right data structure for storing the contact information. I had to consider different algorithms for searching, adding, and deleting contacts. I ultimately opted for a combination of hash maps and lists, which allowed me to balance the trade-offs between speed and memory usage. The most significant learning experience came from working with JSON-based storage. While the implementation itself was straightforward, I learned the importance of understanding the underlying algorithms behind file I/O and serialization, which are crucial for real-world applications.

Another key takeaway was the realization that building efficient data structures is not just about optimizing for speed but also for scalability and future enhancements. Designing the CLI with this in mind was an important lesson in how algorithms can interact with user interfaces to create seamless user experiences.

**Challenges Faced**

Some challenges I faced during this enhancement process included:

* Deciding on the right balance between simplicity and efficiency when selecting data structures. While a more complex structure might offer faster operations, it could also introduce unnecessary complexity for a relatively small-scale project.
* Learning how to implement a robust input validation algorithm that would handle edge cases effectively without overcomplicating the codebase.
* Adapting to the idea of designing with scalability in mind, even for small applications. Although the project was initially simple, I wanted to ensure it could grow with future enhancements.

In conclusion, the enhancements made to the contact manager application have significantly improved its functionality and efficiency by incorporating key algorithms and data structures. By optimizing the data storage and retrieval process, implementing error handling, and designing a user-friendly command-line interface, I was able to address real-world issues of data persistence and usability while improving performance.

This project highlights my ability to apply algorithmic thinking and data structure optimization to create a practical solution, and it marks an important step forward in my journey toward becoming a proficient software engineer with a strong foundation in algorithms and data structures.

**Enhancement List Milestone Two**

1. Enhancement 1: Implement a Graphical User Interface (GUI): Develop a user-friendly GUI using a library like Java Swing or JavaFX. This GUI will provide functionalities for:
   * Adding new contacts
   * Deleting existing contacts
   * Updating contact information
   * Searching for contacts by various criteria (e.g., first name, last name, phone number)
   * Displaying contact information in a clear and organized manner
2. Enhancement 2: Implement Search Functionality: The existing searchByFirstName function can be integrated into the GUI to allow users to search for contacts by first name. Additional search functionalities can be implemented for:
   * Last name
   * Phone number
   * Partial name (using string matching techniques)
3. Enhancement 3: Data Structure Optimization: Utilize a TreeMap for storing contacts, as implemented previously. This allows for efficient retrieval of contacts sorted by their names (or any other attribute used as the key in the TreeMap).
4. Enhancement 4: Error Handling and Input Validation: Implement robust error handling and input validation throughout the application. This includes:
   * Validating user input for contact information to ensure it meets specific criteria (e.g., phone number format).
   * Providing informative error messages to the user in case of invalid input or unexpected errors.
5. Enhancement 5: Documentation: Improve code readability and maintainability by adding JavaDoc comments to all classes, methods, and fields. These comments should explain the purpose, parameters, return values, and any relevant enhancement details.

Explanation:

This revised enhancement plan addresses the feedback by focusing on developing a complete application with a user interface. The plan outlines specific functionalities for the GUI and emphasizes the importance of error handling, input validation, and code documentation. The existing enhancements related to search functionality and data structure are incorporated into the plan.

By implementing these enhancements, the Contact Service artifact will demonstrate a more comprehensive understanding of software engineering principles, user interaction design, and data management techniques.

ContactService.java Before:

import java.util.HashMap;

import java.util.Map;

public class ContactService {

private Map<String, Contact> contacts = new HashMap<>();

public void addContact(Contact contact) {

if (contacts.containsKey(contact.getContactID())) {

throw new IllegalArgumentException("Contact ID already exists");

}

contacts.put(contact.getContactID(), contact);

}

public void deleteContact(String contactID) {

contacts.remove(contactID);

}

public void updateContact(String contactID, String firstName, String lastName, String phone, String address) {

Contact contact = contacts.get(contactID);

if (contact != null) {

contact.setFirstName(firstName);

contact.setLastName(lastName);

contact.setPhone(phone);

contact.setAddress(address);

}

}

public Contact getContact(String contactID) {

return contacts.get(contactID);

}

}

ContactService.java After:  
#include "ContactService.h"

#include <iostream>

#include <fstream>

#include <json.hpp> // For handling JSON

using json = nlohmann::json;

ContactService::ContactService() {

loadContactsFromFile();

}

// Load contacts from the JSON file

void ContactService::loadContactsFromFile() {

std::ifstream inputFile(fileName);

if (inputFile.is\_open()) {

json j;

inputFile >> j;

for (const auto& item : j) {

contacts.push\_back(Contact(item["contactID"], item["firstName"], item["lastName"], item["phone"], item["address"]));

}

}

}

// Save contacts to the JSON file

void ContactService::saveContactsToFile() {

std::ofstream outputFile(fileName);

if (outputFile.is\_open()) {

json j;

for (const auto& contact : contacts) {

j.push\_back({ {"contactID", contact.getContactID()},

{"firstName", contact.getFirstName()},

{"lastName", contact.getLastName()},

{"phone", contact.getPhone()},

{"address", contact.getAddress()} });

}

outputFile << j.dump(4);

}

}

// Add a new contact to the list

void ContactService::addContact(const Contact& contact) {

contacts.push\_back(contact);

saveContactsToFile();

}

// Delete a contact from the list

void ContactService::deleteContact(const std::string& contactID) {

auto it = std::remove\_if(contacts.begin(), contacts.end(),

[&contactID](const Contact& contact) {

return contact.getContactID() == contactID;

});

contacts.erase(it, contacts.end());

saveContactsToFile();

}

// Update an existing contact

void ContactService::updateContact(const std::string& contactID, const std::string& firstName, const std::string& lastName, const std::string& phone, const std::string& address) {

for (auto& contact : contacts) {

if (contact.getContactID() == contactID) {

contact.setFirstName(firstName);

contact.setLastName(lastName);

contact.setPhone(phone);

contact.setAddress(address);

saveContactsToFile();

break;

}

}

}

// Retrieve a contact by ID

Contact ContactService::getContact(const std::string& contactID) {

for (const auto& contact : contacts) {

if (contact.getContactID() == contactID) {

return contact;

}

}

throw std::invalid\_argument("Contact not found");

}

// Display all contacts

void ContactService::displayAllContacts() {

if (contacts.empty()) {

std::cout << "No contacts available.\n";

}

else {

for (const auto& contact : contacts) {

std::cout << "ID: " << contact.getContactID() << ", "

<< "Name: " << contact.getFirstName() << " " << contact.getLastName() << ", "

<< "Phone: " << contact.getPhone() << ", "

<< "Address: " << contact.getAddress() << "\n";

}

}

}

// Menu methods for interaction with the user

void ContactService::addContactMenu() {

std::string contactID, firstName, lastName, phone, address;

std::cout << "Enter contact ID: ";

std::cin >> contactID;

std::cout << "Enter first name: ";

std::cin >> firstName;

std::cout << "Enter last name: ";

std::cin >> lastName;

std::cout << "Enter phone: ";

std::cin >> phone;

std::cout << "Enter address: ";

std::cin >> address;

Contact newContact(contactID, firstName, lastName, phone, address);

addContact(newContact);

std::cout << "Contact added successfully!\n";

}

void ContactService::deleteContactMenu() {

std::string contactID;

std::cout << "Enter contact ID to delete: ";

std::cin >> contactID;

deleteContact(contactID);

std::cout << "Contact deleted successfully!\n";

}

void ContactService::updateContactMenu() {

std::string contactID, firstName, lastName, phone, address;

std::cout << "Enter contact ID to update: ";

std::cin >> contactID;

std::cout << "Enter new first name: ";

std::cin >> firstName;

std::cout << "Enter new last name: ";

std::cin >> lastName;

std::cout << "Enter new phone: ";

std::cin >> phone;

std::cout << "Enter new address: ";

std::cin >> address;

updateContact(contactID, firstName, lastName, phone, address);

std::cout << "Contact updated successfully!\n";

}

void ContactService::searchContactMenu() {

std::string contactID;

std::cout << "Enter contact ID to search: ";

std::cin >> contactID;

try {

Contact contact = getContact(contactID);

std::cout << "Contact found: " << contact.getFirstName() << " " << contact.getLastName() << "\n";

}

catch (const std::invalid\_argument& e) {

std::cout << e.what() << "\n";

}

}

**Instructions on Running and Compiling the Application**

1. Dependencies and Requirements

* C++ Compiler: A C++ compiler such as g++ (MinGW), MSVC (Microsoft Visual Studio), or Clang should be installed.
* IDE (Optional): Visual Studio, Code::Blocks, or Eclipse CDT can be used to easily compile and run the program.

2. Files to Download  
Ensure you have the following C++ source code files:

* Main.cpp
* Contact.cpp
* ContactService.cpp

3. Compilation Steps

* Command Line:
  + MinGW (g++):  
    Run the following in the terminal:

g++ -o contact\_app Main.cpp Contact.cpp ContactService.cpp

* + MSVC (Microsoft Visual Studio Compiler):  
    Open Developer Command Prompt and run:

cl Main.cpp Contact.cpp ContactService.cpp

* Using Visual Studio:
  + Open Visual Studio and create a new C++ Console Application.
  + Add the three .cpp files to the project.
  + Build and run the project by selecting Build > Build Solution and Debug > Start Without Debugging.
* Using Code::Blocks:
  + Open Code::Blocks and create a new project.
  + Add the three .cpp files to your project.
  + Build and run the project by selecting Build > Build and Run.

4. Running the Program Once compiled, execute the application:

* Command Line:  
  In the directory where the executable is located, run:

./contact\_app

* IDE: Press Run or Start in your IDE to execute the program.

**Instructions on How to Use the Application**

1. Application Overview The application allows you to manage contacts with the following functionality:

* Add a contact
* Update a contact’s details
* Delete a contact
* Display all contacts
* Exit the application

2. Menu Options Upon running the program, a menu is displayed:

Contact Management System

1. Add a Contact

2. Update a Contact

3. Delete a Contact

4. Display All Contacts

5. Exit

Enter your choice:

3. Menu Option Details

* Option 1: Add a Contact  
  You will be prompted to enter:
  + Contact ID
  + First Name
  + Last Name
  + Phone Number (10 digits)
  + Address
* Option 2: Update a Contact  
  You will be asked to provide:
  + Contact ID (to identify which contact to update)
  + New details for the contact (First Name, Last Name, Phone, Address)
* Option 3: Delete a Contact  
  You will be asked to enter the Contact ID of the contact you want to delete.
* Option 4: Display All Contacts  
  Displays a list of all contacts with their details.
* Option 5: Exit  
  Ends the program.

4. Error Handling The program validates user input (e.g., ensures phone numbers are 10 digits) and shows an error message if invalid input is provided.

5. Example Workflow When you run the program, here’s how an interaction might look:

Contact Management System

1. Add a Contact

2. Update a Contact

3. Delete a Contact

4. Display All Contacts

5. Exit

Enter your choice: 1

Enter contact ID: 1

Enter first name: John

Enter last name: Doe

Enter phone number (10 digits): 1234567890

Enter address: 123 Elm St

Contact added successfully!

Then, the user can continue to update, delete, or view contacts as needed.

6. Exiting the Program To exit the program, choose option 5 from the menu.