Client and Case Management System for St. Mary’s Legal Services

Introduction

St. Mary's Legal Services is a firm that focuses and specialises in intellectual property claims and operates from different locations which requires a centralised system to effectively help store and manage all the case files and client information. The Client and Case Management System is a Java-based programme that addresses this need by offering powerful features for storing and retrieving all the case files, preserving complete sensitive client information, and securely processing legal documents without any breach in the legislation laws . This system intends to streamline legal professionals' workflows by enabling simple navigation and engagement with the system, consequently increasing productivity and efficiency across all branches of the law business.

Project Structure

The project is modularly structured, with related functionalities encapsulated into classes. The main components of the system are:

- Client: This represents a client with attributes like ID, name, address, email, and phone number.
- Case: This represents a case associated with a client, containing case ID, title, description, client information, and related documents.
- Document: This represents a document associated with a case, containing document ID, name, and content.
- DatabaseManager: This manages the in-memory database operations for clients and cases.
- ConsoleApp: This provides the console-based user interface for interacting with the system.
- App: This is main entry point of the application.

Detailed Class Descriptions

1. Client Class

The Client class is where the details of a client are stored . It contains a lot of information such as the client ID, name, address, email, and phone number. The class includes getters and setters for each attribute to ensure data encapsulation and manipulation via defined methods.

The following is the Java code for this:

```
public static class Client { private int clientId; private String name; private String address;
private String email; private String phoneNumber; // Constructor publicClient(int clientId,
String name, String address, String email, String phoneNumber) {this.clientId = clientId;
this.name = name; this.address = address; this.email = email;this.phoneNumber =
phoneNumber; } // Getters and Setters public int getClientId() {return clientId; } public void
setClientId(int clientId) { this.clientId = clientId; }public String getName() { return name; }
public void setName(String name) { this.name = name; } public String getAddress() { return
address; } public void setAddress(String address) { this.address = address; } public String
getEmail() { return email; } publicvoid setEmail(String email) { this.email = email; } public
String getPhoneNumber() {return phoneNumber; } public void setPhoneNumber(String
phoneNumber) { this.phoneNumber = phoneNumber; } @Override public String toString() {
return "Client [clientId=" + clientId + ", name=" + name + ", address=" + address + ", email="
+ email + ", phoneNumber=" + phoneNumber + "]"; } }
```

The Client class is the system's core component, representing individual clients and providing all relevant personal information. This approach ensures that each client is uniquely identifiable and managed successfully.

2. Case Class

The Case class represents a case associated with a client. It contains case ID, title, description, client, and a list of documents. The class also has methods for adding documents and retrieving information about the case.

The following is the Java code for this:

```
public static class Case { private int caseId; private String title; private String description;
private Client client; private java.util.List<Document> documents; // Constructor public
Case(int caseId, String title, String description, Client client) {this.caseId = caseId; this.title =
title; this.description = description; this.client = client; this.documents = new
java.util.ArrayList<>(); } // Getters and Setters public intgetCaseId() { return caseId; } public
void setCaseId(int caseId) { this.caseId = caseId; } public String getTitle() { return title; }
public void setTitle(String title) {this.title = title; } public String getDescription() { return
description; } public voidsetDescription(String description) { this.description = description; }
public Client getClient() { return client; } public void setClient(Client client) { this.client =
client; } public java.util.List<Document> getDocuments() { return documents; } publicvoid
addDocument(Document document) { this.documents.add(document); } @Override
publicString toString() { return "Case [caseId=" + caseId + ", title=" + title + ", description=" +
description + ", client=" + client + ", documents=" + documents + "]"; } }
```

The Case class is very critical for relating legal issues to individual customers. Each case can be efficiently recorded, changed, and referenced, and managed guaranteeing the continuity of the client-case relationship.

3. Document Class

The Document class contains the details of a document associated with a case. It contains properties like document ID, name, and content. This class additionally includes getters and setters for each attribute.

Java code i've used for this :

```
public static class Document { private int documentId; private String name; privateString
content; // Constructor public Document(int documentId, String name, String content) {
this.documentId = documentId; this.name = name; this.content = content; } // Getters and
Setters public int getDocumentId() { return documentId; } public voidsetDocumentId(int
documentId) { this.documentId = documentId; } public String getName() {return name; }
public void setName(String name) { this.name = name; } public String getContent() { return
content; } public void setContent(String content) { this.content = content; } @Override public
String toString() { return "Document [documentId=" + documentId + ", name=" + name + ",
content=" + content + "]"; } }
```

Documents are very important in any legal dispute because they contain important information and evidence. The Document class enables full administration and retrieval of these papers, making them easily available when needed.

4. DatabaseManager Class

The DatabaseManager class simulates database operations with in-memory lists. It supports CRUD operations for clients and cases. This class is responsible for data integrity and handles the insertion, retrieval, update, and deletion of clients and cases.

Java :

```
public static class DatabaseManager { private java.util.List<Client> clients =
newjava.util.ArrayList<>(); private java.util.List<Case> cases = new java.util.ArrayList<>(); //
CRUD operations for Client public void addClient(Client client) { clients.add(client); } public
Client getClient(int clientId) { for (Client client : clients) { if (client.getClientId() == clientId) {
return client; } } return null; }public void updateClient(Client client) { for (int i = 0; i <
clients.size(); i++) { if(clients.get(i).getClientId() == client.getClientId()) { clients.set(i, client);
return; } } } public void deleteClient(int clientId) { clients.removeIf(client -> client.getClientId()
== clientId); } // CRUD operations for Case public void addCase(Case c) { cases.add(c); }
public Case getCase(int caseId) { for (Case c : cases) { if(c.getCaseId() == caseId) { return c;
} } return null; } public void updateCase(Case c) {for (int i = 0; i < cases.size(); i++) { if
(cases.get(i).getCaseId() == c.getCaseId()) { cases.set(i, c); return; } } } public void
deleteCase(int caseId) { cases.removeIf(c -> c.getCaseId() == caseId); } public
java.util.List<Client> getClients() { return clients; } public java.util.List<Case> getCases() {
return cases; } }
```

The DatabaseManager class is the application's backbone, responsible for all data operations. By managing in-memory lists, it simulates a database environment, allowing for quick CRUD operations for clients and cases without the requirement for a persistent storage solution.

5. ConsoleApp Class

The ConsoleApp class creates a console based user interface for interacting with the system. It accepts user input and calls the necessary methods in the DatabaseManager class to conduct CRUD. The main function of this class has a loop that displays a menu to the user and processes their selections.

Java code :

```
public static class ConsoleApp { private static DatabaseManager database =
newDatabaseManager(); private static Scanner scanner = new Scanner(System.in); public
static void main(String[] args) { while (true) { System.out.println("1. Add Client");
System.out.println("2. View Client"); System.out.println("3. Update Client");
System.out.println("4. Delete Client"); System.out.println("5. Add Case");
System.out.println("6. View Case"); System.out.println("7. Update Case");
System.out.println("8. Delete Case"); System.out.println("9. Exit"); int choice
=scanner.nextInt(); scanner.nextLine();  // Consume newline switch (choice) { case 1:
addClient(); break; case 2: viewClient(); break; case 3: updateClient(); break; case 4:
deleteClient(); break; case 5: addCase(); break; case 6: viewCase(); break; case 7:
updateCase(); break; case 8: deleteCase(); break; case 9: System.exit(0); break; default:
System.out.println("Invalid choice."); } } } private static void addClient() {
System.out.print("Enter client ID: "); int id = scanner.nextInt(); scanner.nextLine();  //
Consume newline System.out.print("Enter client name: "); String name =scanner.nextLine();
System.out.print("Enter client address: "); String address =scanner.nextLine();
System.out.print("Enter client email: "); String email =scanner.nextLine();
System.out.print("Enter client phone number: "); String phoneNumber =scanner.nextLine();
```

```
Client client = new Client(id, name, address, email, phoneNumber);
database.addClient(client); System.out.println("Client added successfully."); } privatestatic
void viewClient() { System.out.print("Enter client ID: "); int id =scanner.nextInt();
scanner.nextLine();  // Consume newline Client client =database.getClient(id); if (client !=
null) { System.out.println(client); } else { System.out.println("Client not found."); } } private
static void updateClient() { System.out.print("Enter client ID to update: "); int id =
scanner.nextInt(); scanner.nextLine();  // Consume newline Client client =
database.getClient(id); if (client != null) { System.out.print("Enter new name: "); String name
= scanner.nextLine(); System.out.print("Enter new address: "); String address =
scanner.nextLine(); System.out.print("Enter new email: "); String email = scanner.nextLine();
System.out.print("Enter new phone number: "); String phoneNumber = scanner.nextLine();
client.setName(name); client.setAddress(address); client.setEmail(email);
client.setPhoneNumber(phoneNumber); database.updateClient(client);
System.out.println("Client updated successfully."); } else { System.out.println("Client not
found."); } } private static void deleteClient() { System.out.print("Enter client ID to delete: ");
int id = scanner.nextInt(); scanner.nextLine();  // Consume newlinedatabase.deleteClient(id);
System.out.println("Client deleted successfully."); } privatestatic void addCase() {
System.out.print("Enter case ID: "); int id = scanner.nextInt(); scanner.nextLine();  //
Consume newline System.out.print("Enter case title: "); Stringtitle = scanner.nextLine();
System.out.print("Enter case description: "); Stringdescription = scanner.nextLine();
System.out.print("Enter client ID for this case: ");int clientId = scanner.nextInt();
scanner.nextLine();  // Consume newline Client client =database.getClient(clientId); if (client
!= null) { Case c = new Case(id, title, description, client); database.addCase(c);
System.out.println("Case added successfully."); } else { System.out.println("Client not
found."); } } private staticvoid viewCase() { System.out.print("Enter case ID: "); int id =
scanner.nextInt(); scanner.nextLine();  // Consume newline Case c = database.getCase(id); if
(c != null) { System.out.println(c); } else { System.out.println("Case not found."); } } private
staticvoid updateCase() { System.out.print("Enter case ID to update: "); int id
=scanner.nextInt(); scanner.nextLine();  // Consume newline Case c =
database.getCase(id);if (c != null) { System.out.print("Enter new title: "); String title
=scanner.nextLine(); System.out.print("Enter new description: "); String description
=scanner.nextLine(); c.setTitle(title); c.setDescription(description); database.updateCase(c);
System.out.println("Case updated successfully."); } else { System.out.println("Case not
found."); } } private static void deleteCase() { System.out.print("Enter case ID to delete: "); int
id = scanner.nextInt(); scanner.nextLine();  // Consume newline database.deleteCase(id);
System.out.println("Case deleted successfully."); } }
```

The ConsoleApp class provides a very simple and easy to use interface for users to interact with the system, including options for viewing,adding, updating, and deleting clients and cases. This class is very important for assuring the application's usability.

Implementation and Functionality

The Client and Case Management System is built using Java, utilising object-oriented principles to ensure modularity and scalability. The following features are available:

- Add Client: This prompts the user to enter client details and adds the client to the database.
- View Client: This allows the user to retrieve and display details of a specific client using the client ID.
- Update Client: This enables the user to update the details of an existing client.
- Delete Client: This removes a client from the database using the client ID.
- Add Case: This prompts the user to enter case details and associates the case with a client.
- View Case: This allows the user to retrieve and display details of a specific case using the case ID.
- Update Case: This enables the user to update the details of an existing case.
- Delete Case: This removes a case from the database using the case ID.

Conclusion

This St. Mary's Legal Services' Client and Case Management System offers a very trustworthy and reliable solution for managing all the customers and cases in a very neat , efficient , systematic and effective manner. The solution gives and provides high data integrity, modularity, and scalability by putting together object-oriented design concepts with Java's rich capabilities. The console-based interface also provides a very easy to use platform for all users to perform critical CRUD operations on clients and cases, making it an important tool for this firm.

This paper describes the system's design, implementation, and functionality, demonstrating how it fits the demands of clients and case managers. Future additions could include incorporating a persistent storage solution, adding user authentication, and designing a graphical user interface to significantly improve usability.