A black background with a black square

Description automatically generated with medium confidence

**DSA521S DATA STRUCTURES AND ALGORITHMS, SEMESTER 2, 2024**

**GROUP ASSIGNMENT PRESENTATION: SCORESHEET AND PROJECT DESCRIPTION TEMPLATE**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **SN** | **Name** | **Student Number** | **Specialisation** | **Role played in the project** | **Mark over 100% (for lecturers use only)** |
| 1 | OJ NAKAZIKO | 222040300 | Cyber Security | Pseudocode |  |
| 2 | UAARUKAPO KUHANGA | 224087975 | Informatics | Flowchart |  |
| 3 | INNOCENT SHEYANENA | 224086154 | Cyber Security | Pseudocode, modules and Functions |  |
| 4 | SHINGIRAI GUNDIDZA | 224018361 | Cyber Security | Source code, flowchart, Modules, Functions |  |
| 5 | REBEKA MATENDE | 223104167 | Computer Science | analysis of search algorithm |  |

**Name of Team Leader (TL):**

Kiami Quinga

**TITLE OF PROJECT**

**TABLE OF CONTENTS**

GROUP MEMBERS ----------------------------------------------------------------------------i

TITLE OF PROJECT ----------------------------------------------------------------------------ii

* DESCRIPTION OF PROJECT ------------------------------------------------------------1
* User Requirements: --------------------------------------------------------------------- 1

3.0 PROGRAMMING LANGUAGES/TOOLS USED---------------------------------------------1

4.0 THE JAVA PROGRAM SOURCE CODES------------------------------------------------4 – 11

5.0 Modules ----------------------------------------------------------------------------------------12

6.0Functions ---------------------------------------------------------------------------------------12

7.0 Pseudocode -----------------------------------------------------------------------------------12

8.0 Flowchart ---------------------------------------------------------------------------------------4

9.0 CONCLUSION-----------------------------------------------------------------------------------4

**DESCRIPTION OF PROJECT**

Our project is a **Phone Book** itallows to user enter and interact with the contact details, in the program. the user is able to manipulate the contact list with feature to Add, Delete, update, sort and display contact details by using a Graphical User Interface.

**User Requirements**

1.Allow the user to enter contact details the system.

2.It allows the user to manipulate the data by adding, editing, deleting and display.

3. the system can also store records of data.

4. it writes the code to a secured database using a LinkedList.

**PROGRAMMING LANGUAGES/TOOLS USED**

Write the name of the programming languages/tools you used here.

E.g.

Programming languages used: Java.

Platform Used: IntelliJ

**Modules:**

1. Insert Contact
2. Search Contact
3. Display All Contacts
4. Delete Contact
5. Update contact
6. Sort Contacts

**Functions:**

Insert Contact:   
Purpose: Adds a new contact to the phone directory.   
Description: This function adds a new contact item to the phonebook list.   
  
Search Contact:   
Purpose: Finds a contact using either their name or phone number.   
Description: Uses a linear search of the phonebook to discover a matching contact depending on the query.   
  
Display all contacts:   
The purpose is to output all contacts saved in the phonebook.   
This function checks if the phonebook is empty and, if not, iterates through the contacts and prints their information.

Delete Contact:   
Purpose: Removes a specific contact from the phonebook.   
Description: This function looks for a contact using a query and deletes it if found.   
  
Update Contact:   
The purpose is to update an existing contact's information.   
Description: The function finds a contact using a query and then updates its name and phone number with the new information.   
  
  
Sort Contacts:   
Purpose: Sort contacts alphabetically by name.   
This optional function employs a merge sort method to arrange contacts in alphabetical order, which improves search efficiency.

Analyse your search algorithm's efficiency: Searching an unsorted list takes O(n) time because each item must be checked individually. Sorting the list first provides for faster binary searches, but the sorting process requires O(n log n) time. Sorting only makes sense if you want to search the list several times, as it requires an initial time investment.

**Java Code:**

import java.util.LinkedList;  
import java.util.Scanner;  
  
class Contact {  
 private String firstName;  
 private String lastName;  
 private String email;  
 private String phoneNumber;  
  
 public Contact(String firstName, String lastName, String email, String phoneNumber) {  
 this.firstName = firstName;  
 this.lastName = lastName;  
 this.email = email;  
 this.phoneNumber = phoneNumber;  
 }  
  
 @Override  
 public String toString() {  
 return "Name: " + firstName + " " + lastName + ", Email: " + email + ", Phone Number: " + phoneNumber;  
 }  
  
 public String getFirstName() {  
 return firstName;  
 }  
  
 public void setFirstName(String firstName) {  
 this.firstName = firstName;  
 }  
  
 public String getLastName() {  
 return lastName;  
 }  
  
 public void setLastName(String lastName) {  
 this.lastName = lastName;  
 }  
  
 public String getEmail() {  
 return email;  
 }  
  
 public void setEmail(String email) {  
 this.email = email;  
 }  
  
 public String getPhoneNumber() {  
 return phoneNumber;  
 }  
  
 public void setPhoneNumber(String phoneNumber) {  
 this.phoneNumber = phoneNumber;  
 }  
}  
  
public class ContactManager {  
 private LinkedList<Contact> contacts = new LinkedList<>();  
 private Scanner scanner = new Scanner(System.*in*);  
  
 public void addContact() {  
 System.*out*.print("Enter first name: ");  
 String firstName = scanner.nextLine();  
 System.*out*.print("Enter last name: ");  
 String lastName = scanner.nextLine();  
 System.*out*.print("Enter email: ");  
 String email = scanner.nextLine();  
 System.*out*.print("Enter phone number: ");  
 String phoneNumber = scanner.nextLine();  
  
 Contact contact = new Contact(firstName, lastName, email, phoneNumber);  
 contacts.add(contact);  
 System.*out*.println("Contact added successfully.");  
 }  
  
 public void searchContact() {  
 System.*out*.print("Enter search keyword: ");  
 String keyword = scanner.nextLine().toLowerCase();  
 boolean found = false;  
 for (Contact contact : contacts) {  
 if (contact.getFirstName().toLowerCase().contains(keyword) ||  
 contact.getLastName().toLowerCase().contains(keyword) ||  
 contact.getEmail().toLowerCase().contains(keyword) ||  
 contact.getPhoneNumber().contains(keyword)) {  
 System.*out*.println("Contact found: " + contact);  
 found = true;  
 }  
 }  
 if (!found) {  
 System.*out*.println("Contact not found.");  
 }  
 }  
  
 public void displayAllContacts() {  
 for (Contact contact : contacts) {  
 System.*out*.println(contact);  
 }  
 }  
  
 public void deleteContact() {  
 System.*out*.print("Enter the first name of the contact to delete: ");  
 String firstName = scanner.nextLine();  
 contacts.removeIf(contact -> contact.getFirstName().equalsIgnoreCase(firstName));  
 System.*out*.println("Contact deleted successfully.");  
 }  
  
 public void sortContacts() {  
 contacts.sort((c1, c2) -> c1.getFirstName().compareToIgnoreCase(c2.getFirstName()));  
 System.*out*.println("Contacts sorted by first name.");  
 }  
  
 public void updateContact() {  
 System.*out*.print("Enter the first name of the contact to update: ");  
 String firstName = scanner.nextLine();  
 for (Contact contact : contacts) {  
 if (contact.getFirstName().equalsIgnoreCase(firstName)) {  
 System.*out*.print("Enter new first name: ");  
 String newFirstName = scanner.nextLine();  
 System.*out*.print("Enter new last name: ");  
 String newLastName = scanner.nextLine();  
 System.*out*.print("Enter new email: ");  
 String newEmail = scanner.nextLine();  
 System.*out*.print("Enter new phone number: ");  
 String newPhoneNumber = scanner.nextLine();  
 contact.setFirstName(newFirstName);  
 contact.setLastName(newLastName);  
 contact.setEmail(newEmail);  
 contact.setPhoneNumber(newPhoneNumber);  
 System.*out*.println("Contact updated successfully.");  
 return;  
 }  
 }  
 System.*out*.println("Contact not found.");  
 }  
  
 public static void main(String[] args) {  
 ContactManager contactManager = new ContactManager();  
 Scanner scanner = new Scanner(System.*in*);  
  
 while (true) {  
 System.*out*.println("\nContact Manager Menu:");  
 System.*out*.println("1. Add Contact");  
 System.*out*.println("2. Search Contact");  
 System.*out*.println("3. Display All Contacts");  
 System.*out*.println("4. Delete Contact");  
 System.*out*.println("5. Update Contact");  
 System.*out*.println("6. Sort Contacts");  
 System.*out*.println("7. Exit");  
 System.*out*.print("Enter your choice: ");  
  
 int choice = scanner.nextInt();  
 scanner.nextLine(); // Consume the newline character  
  
 switch (choice) {  
 case 1:  
 contactManager.addContact();  
 break;  
 case 2:  
 contactManager.searchContact();  
 break;  
 case 3:  
 contactManager.displayAllContacts();  
 break;  
 case 4:  
 contactManager.deleteContact();  
 break;  
 case 5:  
 contactManager.updateContact();  
 break;  
 case 6:  
 contactManager.sortContacts();  
 break;  
 case 7:  
 System.*out*.println("Exiting program...");  
 System.*exit*(0);  
 default:  
 System.*out*.println("Invalid choice. Please try again.");  
 }  
 }  
 }  
}

**Pseudocode**

**START**

**FUNCTION displayMenu()**

**// This function displays the main menu options to the user.**

PRINT "1. Insert Contact**" // Option to add a new contact**

PRINT "2. Update Contact" **// Option to update an existing contact**

PRINT "3. Delete Contact" **// Option to delete a contact**

PRINT "4. Sort Contacts**" // Option to sort the contacts alphabetically**

PRINT "5. Exit**" // Option to exit the application**

**END FUNCTION**

**FUNCTION insertContact()**

**// This function allows the user to add a new contact.**

PRINT "Enter Name:" **// Prompt the user for the contact's name**

READ name **// Read the name input from the user**

PRINT "Enter Phone Number:" **// Prompt for the contact's phone number**

READ phoneNumber **// Read the phone number input from the user**

ADD {name, phoneNumber} TO contacts **// Store the new contact in the contacts collection**

PRINT "Contact added successfully." **// Confirm the addition of the contact**

END FUNCTION

**FUNCTION updateContact()**

**// This function allows the user to update an existing contact's phone number.**

PRINT "Enter the name of the contact to update:" **// Prompt for the name of the contact to update**

READ name **// Read the name input from the user**

IF name EXISTS IN contacts THEN **// Check if the contact exists**

PRINT "Enter new Phone Number:" **// Prompt for the new phone number**

READ newPhoneNumber **// Read the new phone number input**

UPDATE phoneNumber OF contact WHERE name = name TO newPhoneNumber **// Update the contact's phone number**

PRINT "Contact updated successfully." **// Confirm the update**

ELSE

PRINT "Contact not found." **// Inform the user if the contact does not exist**

END IF

END FUNCTION

**FUNCTION deleteContact()**

**// This function allows the user to delete a contact.**

PRINT "Enter the name of the contact to delete:"  **// Prompt for the name of the contact to delete**

READ name **// Read the name input from the user**

IF name EXISTS IN contacts THEN **// Check if the contact exists**

REMOVE contact WHERE name = name FROM contacts **// Remove the contact from the contacts collection**

PRINT "Contact deleted successfully." **// Confirm the deletion**

ELSE

PRINT "Contact not found." **// Inform the user if the contact does not exist**

END IF

END FUNCTION

**FUNCTION sortContacts()**

**// This function sorts the contacts alphabetically by name.**

SORT contacts BY name **// Sort the contacts collection by name**

PRINT "Contacts sorted." **// Confirm that sorting has occurred**

FOR EACH contact IN contacts DO **// Iterate through each contact in the sorted list**

PRINT contact.name, contact.phoneNumber **// Print each contact's name and phone number**

END FOR

END FUNCTION

**FUNCTION main()**

**// This is the main function that controls the flow of the application.**

WHILE TRUE DO **// Start an infinite loop to keep the application running**

displayMenu() **// Display the menu options to the user**

PRINT "Choose an option:" **// Prompt the user to choose an option**

READ option **// Read the user's choice**

SWITCH option DO **// Switch statement to handle different options**

CASE 1:

insertContact() **// Call function to insert a new contact**

BREAK

CASE 2:

updateContact() **// Call function to update a contact** BREAK

CASE 3:

deleteContact() **// Call function to delete a contact**

CASE 4:

sortContacts() **// Call function to sort contacts**

BREAK

CASE 5:

PRINT "Exiting the application." **// Inform the user that the application will close**

EXIT  **// Exit the infinite loop**

DEFAULT:

PRINT "Invalid option. Please try again." **// Inform the user of invalid input**

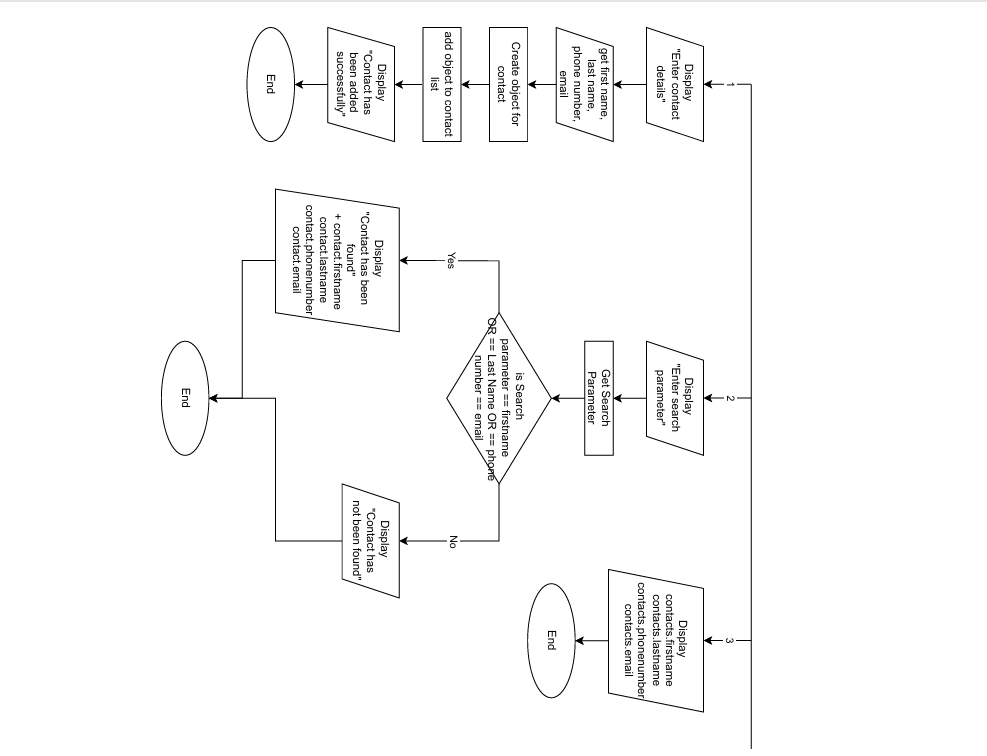
END SWITCH

END WHILE

END FUNCTION

CALL main() **// Start the application by calling the main function**

**FLOWCHART**



A diagram of a flowchart

Description automatically generated

A diagram of a flowchart

Description automatically generated

**CONCLUSION**

In conclusion, our project was an eye-opening experience that demonstrated the value of collaboration, planning, innovation, and practical application. We learned how to bridge the gap between theory and real-world implementation, making our Mark System more dependable, dynamic, and, ultimately, profitable. It was a difficult road that took perseverance and quality assurance efforts, but we eventually succeeded in translating our theoretical knowledge into a practical, usable solution.