**Data Structures and Algorithms Project**

**on**

**PHONEBOOK**

By:

DIVYA(23CSU103)

DIVYANSHI(23CSU105)

Submitted To:

Dr. Anuradha Dhull

A logo of a university

Description automatically generated

**Department of Computer Science and Engineering**

**The NorthCap University**

HUDA, Sec-23A, Gurugram, Haryana – 122017

Session 2023-24

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Table of Contents** |  |  |
|  |  |  |  |
| **S.No** |  | **Page** |  |
|  |  | **No.** |  |
|  |  |  |  |
| **1.** | **Project Title** |  |  |
|  |  |  |
|  |  |  |  |
| **2.** | **Description of Project:** |  |  |
|  | **Problem Statement** |  |
|  |  |  |
|  |  |  |  |
| **3.** | **Design** |  |  |
|  | **3.1Diagram/Program Steps** |  |  |
|  |  |  |  |
| **4.** | **Output (Screenshots)** |  |  |
|  |  |  |  |
| **5.** | **Conclusion and Future Scope** |  |  |
|  |  |  |  |

**Description**

Design a PHONEBOOK Consisting of All Features Supported

**Definition of Done**

1. The phonebook application can add, edit, and delete contacts.
2. Users can search contacts by name, phone number, or other stored information.
3. Multiple contact details (like name, phone number, email, address, etc.) can be stored for each contact.
4. Contact list can be displayed in alphabetical order or customized order.
5. Users are notified of invalid inputs or errors, with suggestions for correction (e.g., invalid phone number or duplicate contact).

**Algorithm**

1. **Initialize**:

* Create a Contact class to store the following:
  + name, phoneNumber, email, and address.
* Create a PhoneBook class with a List<Contact> to store all contacts.

1. **Menu-Driven System**:

* Display the following options in a loop until the user exits:
  1. Add Contact
  2. Edit Contact
  3. Delete Contact
  4. Search Contacts
  5. Display Contacts
  6. Exit

1. **Add Contact**:

* Input the contact details: name, phoneNumber, email, address.
* Validate:
  + Check if a contact with the same name already exists in the list.
  + Ensure the phone number is a valid 10-digit number.
* If valid, add the contact to the list.

1. **Edit Contact**:

* Input the name of the contact to edit.
* Search the list for a contact with the specified name.
* If found:
  + Update the provided fields (phoneNumber, email, address).
  + Validate the updated details before saving.
* If not found, display an error message.

1. **Delete Contact**:

* Input the name of the contact to delete.
* Search the list for a contact with the specified name.
* If found, remove the contact from the list.
* If not found, display an error message.

1. **Search Contacts**:

* Input a search query.
* Iterate through the list and check if the query matches any field (name, phoneNumber, email, or address) of each contact.
* Display matching contacts or indicate if no matches are found.

1. **Display Contacts**:

* Input the preferred display order (alphabetical or default order).
* If alphabetical:
  + Sort the list by the name field using a comparator.
* Display all contacts with their details.

1. **Validation**:

* Validate phone numbers using regex (\d{10}).
* Ensure names are unique within the list.

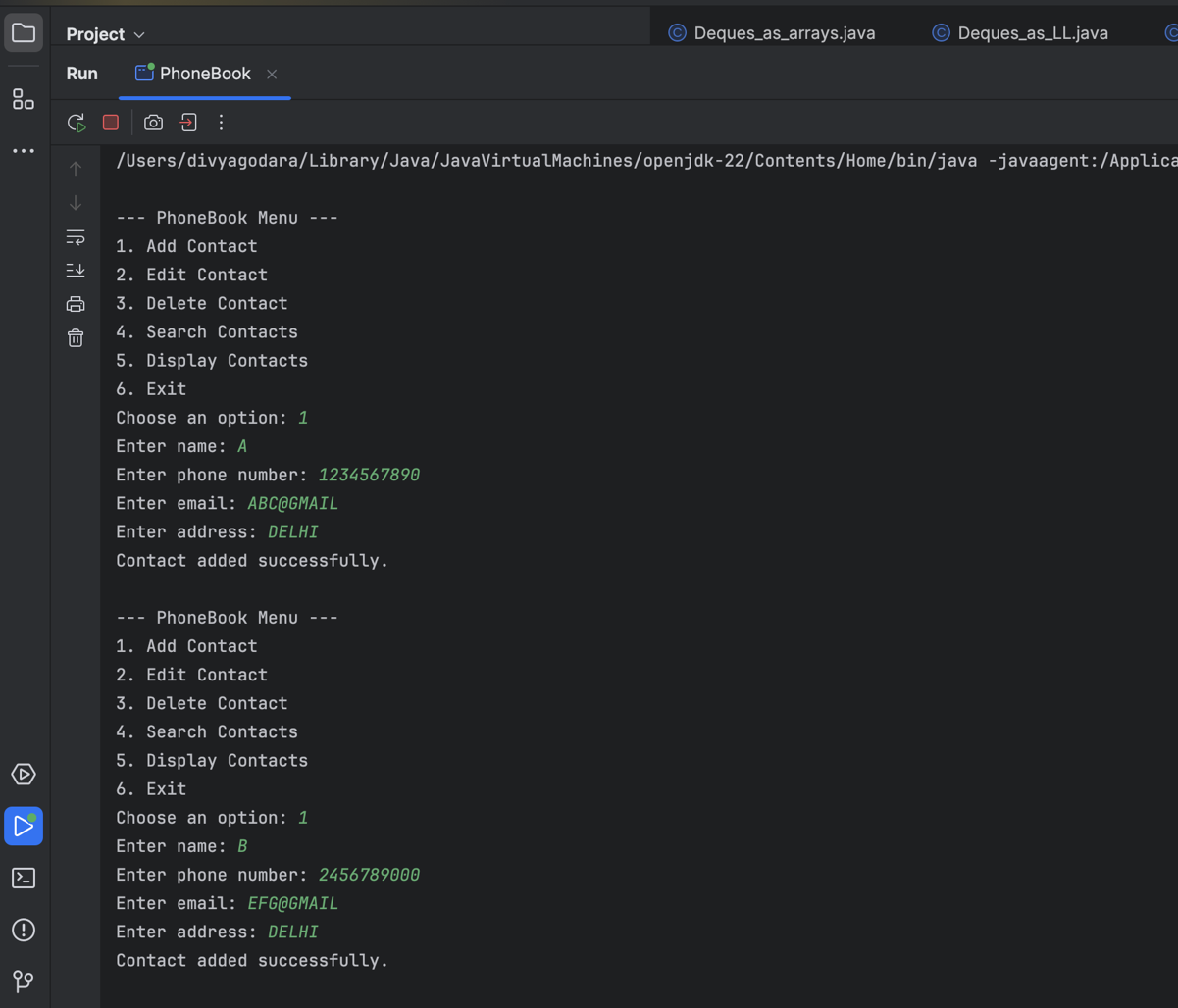
1. **Exit**:

* Terminate the program when the user selects the exit option.

**Code**

import java.util.\*;  
  
public class PhoneBook {  
  
 // Contact class to store contact details  
 static class Contact {  
 String name;  
 String phoneNumber;  
 String email;  
 String address;  
  
 Contact(String name, String phoneNumber, String email, String address) {  
 this.name = name;  
 this.phoneNumber = phoneNumber;  
 this.email = email;  
 this.address = address;  
 }  
  
 @Override  
 public String toString() {  
 return "Name: " + name + "\nPhone Number: " + phoneNumber + "\nEmail: " + email + "\nAddress: " + address;  
 }  
 }  
  
 private final Map<String, Contact> contacts = new HashMap<>();  
  
 // Method to add a contact  
 public void addContact(String name, String phoneNumber, String email, String address) {  
 if (contacts.containsKey(name)) {  
 System.*out*.println("Error: Duplicate contact. A contact with this name already exists.");  
 } else if (!isValidPhoneNumber(phoneNumber)) {  
 System.*out*.println("Error: Invalid phone number. Please enter a valid phone number.");  
 } else {  
 contacts.put(name, new Contact(name, phoneNumber, email, address));  
 System.*out*.println("Contact added successfully.");  
 }  
 }  
  
 // Method to edit a contact  
 public void editContact(String name, String phoneNumber, String email, String address) {  
 Contact contact = contacts.get(name);  
 if (contact == null) {  
 System.*out*.println("Error: Contact not found.");  
 } else {  
 if (phoneNumber != null && !isValidPhoneNumber(phoneNumber)) {  
 System.*out*.println("Error: Invalid phone number. Please enter a valid phone number.");  
 return;  
 }  
  
 contact.phoneNumber = phoneNumber != null ? phoneNumber : contact.phoneNumber;  
 contact.email = email != null ? email : contact.email;  
 contact.address = address != null ? address : contact.address;  
 System.*out*.println("Contact updated successfully.");  
 }  
 }  
  
 // Method to delete a contact  
 public void deleteContact(String name) {  
 if (contacts.remove(name) != null) {  
 System.*out*.println("Contact deleted successfully.");  
 } else {  
 System.*out*.println("Error: Contact not found.");  
 }  
 }  
  
 // Method to search contacts  
 public void searchContacts(String query) {  
 boolean found = false;  
 for (Contact contact : contacts.values()) {  
 if (contact.name.contains(query) || contact.phoneNumber.contains(query) ||  
 (contact.email != null && contact.email.contains(query)) ||  
 (contact.address != null && contact.address.contains(query))) {  
 System.*out*.println(contact);  
 System.*out*.println("------------------------");  
 found = true;  
 }  
 }  
 if (!found) {  
 System.*out*.println("No contacts found for the given query.");  
 }  
 }  
  
 // Method to display all contacts  
 public void displayContacts(boolean alphabeticalOrder) {  
 List<Contact> contactList = new ArrayList<>(contacts.values());  
 if (alphabeticalOrder) {  
 contactList.sort(Comparator.*comparing*(contact -> contact.name));  
 }  
  
 for (Contact contact : contactList) {  
 System.*out*.println(contact);  
 System.*out*.println("------------------------");  
 }  
 }  
  
 // Utility method to validate phone numbers  
 private boolean isValidPhoneNumber(String phoneNumber) {  
 return phoneNumber.matches("\\d{10}"); // Ensures phone number is exactly 10 digits  
 }  
  
 public static void main(String[] args) {  
 PhoneBook phoneBook = new PhoneBook();  
 Scanner scanner = new Scanner(System.*in*);  
  
 while (true) {  
 System.*out*.println("\n--- PhoneBook Menu ---");  
 System.*out*.println("1. Add Contact");  
 System.*out*.println("2. Edit Contact");  
 System.*out*.println("3. Delete Contact");  
 System.*out*.println("4. Search Contacts");  
 System.*out*.println("5. Display Contacts");  
 System.*out*.println("6. Exit");  
 System.*out*.print("Choose an option: ");  
  
 int choice = scanner.nextInt();  
 scanner.nextLine(); // Consume the newline character  
  
 switch (choice) {  
 case 1:  
 System.*out*.print("Enter name: ");  
 String name = scanner.nextLine();  
 System.*out*.print("Enter phone number: ");  
 String phoneNumber = scanner.nextLine();  
 System.*out*.print("Enter email: ");  
 String email = scanner.nextLine();  
 System.*out*.print("Enter address: ");  
 String address = scanner.nextLine();  
 phoneBook.addContact(name, phoneNumber, email, address);  
 break;  
 case 2:  
 System.*out*.print("Enter name of the contact to edit: ");  
 String editName = scanner.nextLine();  
 System.*out*.print("Enter new phone number (or press Enter to skip): ");  
 String newPhoneNumber = scanner.nextLine();  
 newPhoneNumber = newPhoneNumber.isEmpty() ? null : newPhoneNumber;  
 System.*out*.print("Enter new email (or press Enter to skip): ");  
 String newEmail = scanner.nextLine();  
 newEmail = newEmail.isEmpty() ? null : newEmail;  
 System.*out*.print("Enter new address (or press Enter to skip): ");  
 String newAddress = scanner.nextLine();  
 newAddress = newAddress.isEmpty() ? null : newAddress;  
 phoneBook.editContact(editName, newPhoneNumber, newEmail, newAddress);  
 break;  
 case 3:  
 System.*out*.print("Enter name of the contact to delete: ");  
 String deleteName = scanner.nextLine();  
 phoneBook.deleteContact(deleteName);  
 break;  
 case 4:  
 System.*out*.print("Enter search query: ");  
 String query = scanner.nextLine();  
 phoneBook.searchContacts(query);  
 break;  
 case 5:  
 System.*out*.print("Display contacts in alphabetical order? (yes/no): ");  
 boolean alphabeticalOrder = scanner.nextLine().equalsIgnoreCase("yes");  
 phoneBook.displayContacts(alphabeticalOrder);  
 break;  
 case 6:  
 System.*out*.println("Exiting PhoneBook. Goodbye!");  
 scanner.close();  
 return;  
 default:  
 System.*out*.println("Invalid choice. Please try again.");  
 }  
 }  
 }  
}

**OUTPUT:**

****

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**FUTURE SCOPE**

1. **Graphical User Interface (GUI):**

* Develop a GUI using JavaFX or Swing for enhanced user experience.
* Allow drag-and-drop functionality for contact organization.

1. **Cloud Integration:**

* Sync contacts with cloud storage (e.g., Google Drive, AWS).
* Enable cross-platform accessibility through cloud-based services.

1. **Advanced Search Features:**

* Add filters (e.g., search by specific fields like "only by email" or "only by phone number").
* Implement fuzzy search for more flexible query matching.

1. **Export/Import Functionality:**

* Allow exporting contact lists to CSV, Excel, or VCF formats.
* Support importing contacts from external files or accounts (e.g., Gmail).

1. **Enhanced Security:**

* Encrypt sensitive contact information.
* Add user authentication to restrict access.

1. **Mobile Application Development:**

* Develop Android/iOS versions of the application.
* Integrate features like calling, messaging, or emailing directly from the app.

1. **Integration with Social Platforms:**

* Fetch and update contact information from social media profiles.
* Notify users about updates to linked contacts.

1. **Performance Optimization:**

* Improve search performance for large contact databases.
* Optimize memory usage for efficient storage and retrieval.

1. **AI-Based Features:**

* Use machine learning to suggest duplicate contacts or auto-complete missing details.
* Add voice commands for hands-free operations.

1. **Backup and Restore:**

* Provide automatic and manual backup options.
* Allow restoring deleted or modified contacts from a history log.

**BIBLIOGRAPHY**

[**https://www.oracle.com/in/java/**](https://www.oracle.com/in/java/)

[**https://www.geeksforgeeks.org/java/**](https://www.geeksforgeeks.org/java/)

[**https://www.javatpoint.com/java-tutorial**](https://www.javatpoint.com/java-tutorial)