ADDRESS BOOK ORGANIZER

"Address Book Organizer" is a Java program designed to help users manage and organize their contact information efficiently. It provides a user-friendly interface for adding, displaying, and storing contacts, including names, phone numbers, and email addresses.

Key Features:

1. Contact Management: Allows users to add, edit, and remove contact entries with detailed information, including names, phone numbers, and email addresses.

2. User-Friendly Interface: Provides an intuitive menu-driven interface for easy interaction with the program.

3. Data Persistence: Stores contact information in memory, enabling users to maintain their address book even after closing the program.

4. Display Contacts: Offers the ability to view a list of all stored contacts, including their names, phone numbers, and email addresses.

5. Data Validation: Ensures data integrity by validating user input and handling common input errors gracefully.

Now, let's break down the code in detail:

1. Contact Class (`class Contact`):
- This class defines the structure of a contact, including name, phone number, and email.
- It includes a constructor, getters, and a `toString()` method to represent contact details as a string.

2. AddressBook Class (`class AddressBook`):
- This class manages a list of contacts using an `ArrayList`.
- It provides methods for adding contacts (`addContact`) and retrieving all contacts (`getContacts`).

3. Menu Class (`class Menu`):
- This class handles the program's user interface, displaying a menu and processing user choices.
- It takes an `AddressBook` instance as a parameter to interact with the contact data.
- The `displayMenu` method shows the available options, and the `processChoice` method handles user input to execute the selected action.

4. Main Class (`public class AddressBookOrganizer`):
- The main class initializes a `Scanner` for user input and creates instances of `AddressBook` and `Menu`.
- It runs an infinite loop to repeatedly display the menu, get user input, and perform the chosen action.
- It catches and handles `InputMismatchException` to manage invalid inputs gracefully.

Code :-

```
import java.util.InputMismatchException;
import java.util.Scanner;
import java.util.ArrayList;
import java.util.List;

class Contact {
   private String name;
   private String phoneNumber;
   private String email;

   public Contact(String name, String phoneNumber, String email)
      { this.name = name;
      this.phoneNumber = phoneNumber;
      this.email = email;

   }  public String getName() {
      return name;
   }
   public String getPhoneNumber() {
      return phoneNumber;
```

```
  }
   public String getEmail() {
      return email;
   }
   public String toString() {
      return "Name: " + name + ", Phone Number: " + phoneNumber + ", Email: " +
email;
   }
}
class AddressBook {
   private List<Contact> contacts = new ArrayList<>();

   public void addContact(Contact contact) {
      contacts.add(contact);
   }
   public List<Contact> getContacts() {
      return contacts;
   }
}
class Menu {
   private AddressBook addressBook;

   public Menu(AddressBook addressBook) {
      this.addressBook = addressBook;
   }
   public void displayMenu() {
      System.out.println("Address Book Organizer Menu:");
      System.out.println("1. Add Contact");
      System.out.println("2. Display Contacts");
      System.out.println("3. Exit");
      System.out.print("Enter your choice: ");

   } public void processChoice(int choice) {
      Scanner scanner = new Scanner(System.in);
      switch (choice) {
         case 1:
            System.out.print("Enter Name: ");
            String name = scanner.next();
            System.out.print("Enter Phone Number: ");
```

```
            String phoneNumber = scanner.next();
            System.out.print("Enter Email: ");
            String email = scanner.next();
            addressBook.addContact(new Contact(name, phoneNumber,
            email)); System.out.println("Contact added successfully!");
            break;
        case 2:
            System.out.println("Contacts in Address Book:");
            for (Contact contact : addressBook.getContacts()) {
                System.out.println(contact);
            }
            break;
        case 3:
            System.out.println("Exiting the Address Book Organizer.");
            System.exit(0);
        default:
            System.out.println("Invalid choice. Please enter 1, 2, or
            3."); break;
      }
   }
} public class AddressBookOrganizer {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      AddressBook addressBook = new AddressBook();
      Menu menu = new Menu(addressBook);

      while (true) {
         menu.displayMenu();
         try {
            int choice = scanner.nextInt();
            menu.processChoice(choice);
         } catch (InputMismatchException e) {
            System.out.println("Invalid input. Please enter a number.");
            scanner.nextLine(); // Clear the input buffer
         }
      }
   }
}
```

Output :-

```
C:\Users\Nitro\Desktop\ >java AddressBookOrganizer
Address Book Organizer Menu:
1. Add Contact
2. Display Contacts
3. Exit
Enter your choice:
```

```
C:\Users\Nitro\Desktop\ >java AddressBookOrganizer
Address Book Organizer Menu:
1. Add Contact
2. Display Contacts
3. Exit
Enter your choice: 1
Enter Name: Bhavarth
Enter Phone Number: 6654218795
Enter Email: bhavarth16@gmail.com
Contact added successfully!
```

```
Address Book Organizer Menu:
1. Add Contact
2. Display Contacts
3. Exit
Enter your choice: 2
Contacts in Address Book:
Name: Bhavarth, Phone Number: 6654218795, Email: bhavarth16@gmail.com
```

Strengths:

- Simplifies contact management, making it easy to organize and retrieve contact information. - Provides a straightforward and user-friendly interface for users of all experience levels. - The code is modular, making it easy to maintain and extend with additional features.

Weaknesses:

- Lack of advanced features: The program is relatively basic and does not include advanced functionalities such as search or sorting capabilities.
- Limited data storage: The contact information is stored in memory during program execution and will be lost when the program is closed.

Opportunities:

- Integration with external databases: The program could be extended to connect with external databases or cloud storage for more extensive and persistent data management. - Additional features: Further enhancements, such as search functionality or export/import options, could be added to improve the program's usability.

Threats:

- Competition: There are many existing contact management solutions and applications, both desktop and web-based, which could pose competition.
- Security concerns: Handling contact information requires careful consideration of data security and privacy.

Conclusion:

The "Address Book Organizer" is a simple yet functional Java program that serves as a basic tool for managing contact information. While it lacks advanced features found in commercial contact management software, it provides a solid foundation for learning Java programming, including user input handling, data management, and menu-driven interfaces.