**Project Name**: PhoneBookManageemnt

**Description:** This application allows users to manage contacts by adding, editing, deleting, searching, and displaying all contacts.

**Class Documentation  
Login**package PhoneBook;

import javax.swing.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

public class Login extends JFrame {

private JTextField userText;

private JPasswordField passwordText;

private JButton loginButton;

private JLabel userLabel, passwordLabel;

public Login() {

setTitle("Login Page");

setSize(350, 200);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setLocationRelativeTo(null);

setLayout(null);

// Username Label and TextField

userLabel = new JLabel("Username:");

userLabel.setBounds(50, 30, 80, 25);

add(userLabel);

userText = new JTextField();

userText.setBounds(140, 30, 150, 25);

add(userText);

// Password Label and PasswordField

passwordLabel = new JLabel("Password:");

passwordLabel.setBounds(50, 70, 80, 25);

add(passwordLabel);

passwordText = new JPasswordField();

passwordText.setBounds(140, 70, 150, 25);

add(passwordText);

// Login Button

loginButton = new JButton("Login");

loginButton.setBounds(140, 110, 150, 25);

add(loginButton);

// Action Listener for the login button

loginButton.addActionListener(new ActionListener() {

@Override

public void actionPerformed(ActionEvent e) {

String username = userText.getText();

String password = new String(passwordText.getPassword());

// Simple check (you can replace this with database authentication)

if (username.equals("admin") && password.equals("password")) {

JOptionPane.showMessageDialog(null, "Login Successful!");

// Open Home Page and close the Login form

ContactManager contactManager = new ContactManager(); // Initialize the ContactManager

new Home(contactManager).setVisible(true);

dispose(); // Close login window

} else {

JOptionPane.showMessageDialog(null, "Invalid credentials!");

}

}

});

setVisible(true);

}

public static void main(String[] args) {

new Login();

}

}  
 **Home**package PhoneBook;

import javax.swing.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

public class Home extends JFrame {

private ContactManager contactManager;

// Constructor

public Home(ContactManager contactManager) {

this.contactManager = contactManager;

// Set the title

setTitle("Phone Book");

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setSize(400, 400);

setLayout(null);

// Create buttons for the main options

JButton addButton = new JButton("Add");

JButton searchButton = new JButton("Search");

JButton displayButton = new JButton("Display All");

JButton editButton = new JButton("Edit");

JButton deleteButton = new JButton("Delete");

JButton exitButton = new JButton("Exit");

// Set bounds for each button

addButton.setBounds(50, 50, 100, 40);

searchButton.setBounds(50, 100, 100, 40);

displayButton.setBounds(50, 150, 100, 40);

editButton.setBounds(50, 200, 100, 40);

deleteButton.setBounds(50, 250, 100, 40);

exitButton.setBounds(50, 300, 100, 40);

// Add buttons to the frame

add(addButton);

add(searchButton);

add(displayButton);

add(editButton);

add(deleteButton);

add(exitButton);

// Add button actions

addButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

new AddContact(contactManager).setVisible(true);

dispose();

}

});

searchButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

new SearchContact(contactManager).setVisible(true);

dispose();

}

});

displayButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

new DisplayAllContacts(contactManager).setVisible(true);

dispose();

}

});

editButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

new EditContact(contactManager).setVisible(true);

dispose();

}

});

deleteButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

new DeleteContact(contactManager).setVisible(true);

dispose();

}

});

exitButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

System.exit(0); // Exit the application

}

});

// Set window visibility

setVisible(true);

}

// Main method to start the program with the login page

public static void main(String[] args) {

// Call the login page before showing the home page

new Login();

}

Home() {

throw new UnsupportedOperationException("Not supported yet."); // Generated from nbfs://nbhost/SystemFileSystem/Templates/Classes/Code/GeneratedMethodBody

}

}  
  
  
**1. Insert Contact  
Purpose:** This class serves as the main entry point or home screen of the application, where users can navigate to other features like adding, deleting, searching, or viewing contacts **key Functions:** Navigation buttons to direct users to different functionalities (e.g., add, delete, search)  
  
package PhoneBook;

import javax.swing.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

public class AddContact extends JFrame {

private JTextField nameField, phoneField, emailField;

private ContactManager contactManager;

public AddContact(ContactManager contactManager) {

this.contactManager = contactManager;

setTitle("Add Contact");

setLayout(null);

setSize(400, 400);

JLabel nameLabel = new JLabel("Name:");

nameField = new JTextField();

nameLabel.setBounds(50, 50, 100, 30);

nameField.setBounds(150, 50, 150, 30);

JLabel phoneLabel = new JLabel("Phone:");

phoneField = new JTextField();

phoneLabel.setBounds(50, 100, 100, 30);

phoneField.setBounds(150, 100, 150, 30);

JLabel emailLabel = new JLabel("Email:");

emailField = new JTextField();

emailLabel.setBounds(50, 150, 100, 30);

emailField.setBounds(150, 150, 150, 30);

JButton saveButton = new JButton("Save");

saveButton.setBounds(100, 250, 100, 30);

JButton backButton = new JButton("Back");

backButton.setBounds(220, 250, 100, 30);

add(nameLabel);

add(nameField);

add(phoneLabel);

add(phoneField);

add(emailLabel);

add(emailField);

add(saveButton);

add(backButton);

saveButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

String name = nameField.getText();

String phone = phoneField.getText();

String email = emailField.getText();

contactManager.addContact(new Contact(name, phone, email));

JOptionPane.showMessageDialog(null, "Contact added successfully!");

nameField.setText("");

phoneField.setText("");

emailField.setText("");

}

});

backButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

new Home(contactManager).setVisible(true);

dispose();

}

})

setVisible(true);

}

}

**2. Search Contact  
Purpose**: This class enables the user to search for a contact based on criteria (e.g., name, phone number).  
  
**Key Functions**:

* Input search criteria
* Display search results.

package PhoneBook;

import javax.swing.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

public class SearchContact extends JFrame {

private JTextField nameField;

private JTextArea resultArea;

private ContactManager contactManager;

public SearchContact(ContactManager contactManager) {

this.contactManager = contactManager;

setTitle("Search Contact");

setLayout(null);

setSize(400, 300);

JLabel nameLabel = new JLabel("Enter Name to Search:");

nameLabel.setBounds(50, 30, 150, 30);

nameField = new JTextField();

nameField.setBounds(200, 30, 150, 30);

JButton searchButton = new JButton("Search");

searchButton.setBounds(150, 70, 100, 30);

JButton backButton = new JButton("Back");

backButton.setBounds(150, 110, 100, 30);

resultArea = new JTextArea();

resultArea.setBounds(50, 150, 300, 100);

resultArea.setEditable(false);

add(nameLabel);

add(nameField);

add(searchButton);

add(backButton);

add(resultArea);

searchButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

String name = nameField.getText();

Contact contact = contactManager.searchContact(name);

if (contact != null) {

resultArea.setText(contact.toString());

} else {

resultArea.setText("Contact not found.");

}

}

**3. Display all contacts   
Purpose**: This class displays a list of all contacts stored in the phonebook

**Key Functions**:

* Retrieve and display all contact details from the phonebook.

package PhoneBook;

import javax.swing.\*;

import java.awt.\*;

import java.util.LinkedList;

public class DisplayAllContacts extends JFrame {

private ContactManager contactManager;

private JTextArea textArea;

public DisplayAllContacts(ContactManager contactManager) {

this.contactManager = contactManager;

setTitle("All Contacts");

setSize(400, 300);

setLayout(new BorderLayout());

textArea = new JTextArea();

textArea.setEditable(false);

displayContacts();

JScrollPane scrollPane = new JScrollPane(textArea);

add(scrollPane, BorderLayout.CENTER);

JButton backButton = new JButton("Back");

backButton.addActionListener(e -> {

this.setVisible(false);

new Home(contactManager).setVisible(true); // Go back to Home

});

add(backButton, BorderLayout.SOUTH);

setVisible(true);

}

private void displayContacts() {

LinkedList<Contact> contacts = contactManager.getAllContacts(); // Assuming this method exists

if (contacts.isEmpty()) {

textArea.setText("No contacts found.");

} else {

StringBuilder sb = new StringBuilder();

for (Contact contact : contacts) {

sb.append("Name: ").append(contact.getName()).append("\n");

sb.append("Phone: ").append(contact.getPhone()).append("\n");

sb.append("Email: ").append(contact.getEmail()).append("\n\n");

}

textArea.setText(sb.toString());

}

}

}

**4. Delete Contact**   
**Purpose**: This class provides the functionality to delete a contact from the phonebook  
**Key Functions**:

* Search for a contact by name or number.
* Delete the contact from the list.

package PhoneBook;

import javax.swing.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

public class DeleteContact extends JFrame {

private JTextField nameField;

private ContactManager contactManager;

public DeleteContact(ContactManager contactManager) {

this.contactManager = contactManager;

setTitle("Delete Contact");

setLayout(null);

setSize(400, 200);

JLabel nameLabel = new JLabel("Enter Name to Delete:");

nameLabel.setBounds(50, 30, 150, 30);

nameField = new JTextField();

nameField.setBounds(200, 30, 150, 30);

JButton deleteButton = new JButton("Delete");

deleteButton.setBounds(150, 70, 100, 30);

JButton backButton = new JButton("Back");

backButton.setBounds(150, 110, 100, 30);

add(nameLabel);

add(nameField);

add(deleteButton);

add(backButton);

deleteButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

String name = nameField.getText();

boolean deleted = contactManager.deleteContact(name);

if (deleted) {

JOptionPane.showMessageDialog(null, "Contact deleted successfully.");

} else {

JOptionPane.showMessageDialog(null, "Contact not found.");

}

nameField.setText(""); // Clear the input field

}

});

backButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

new Home(contactManager).setVisible(true);

dispose();

}

});

setVisible(true);

}

}

**5. Update Contact**   
**Purpose**: This class allows the user to modify details of an existing contact.

**Key Functions**:

* Search for a contact.
* Update the contact's details.

package PhoneBook;

import javax.swing.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

public class EditContact extends JFrame {

private JTextField nameField, phoneField, emailField;

private ContactManager contactManager;

public EditContact(ContactManager contactManager) {

this.contactManager = contactManager;

setTitle("Edit Contact");

setLayout(null);

setSize(400, 300);

JLabel nameLabel = new JLabel("Enter Name:");

nameLabel.setBounds(50, 30, 150, 30);

nameField = new JTextField();

nameField.setBounds(200, 30, 150, 30);

JLabel phoneLabel = new JLabel("New Phone:");

phoneLabel.setBounds(50, 70, 150, 30);

phoneField = new JTextField();

phoneField.setBounds(200, 70, 150, 30);

JLabel emailLabel = new JLabel("New Email:");

emailLabel.setBounds(50, 110, 150, 30);

emailField = new JTextField();

emailField.setBounds(200, 110, 150, 30);

JButton editButton = new JButton("Edit");

editButton.setBounds(150, 150, 100, 30);

JButton backButton = new JButton("Back");

backButton.setBounds(150, 190, 100, 30);

add(nameLabel);

add(nameField);

add(phoneLabel);

add(phoneField);

add(emailLabel);

add(emailField);

add(editButton);

add(backButton);

editButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

String name = nameField.getText();

String newPhone = phoneField.getText();

String newEmail = emailField.getText();

boolean updated = contactManager.editContact(name, newPhone, newEmail);

if (updated) {

JOptionPane.showMessageDialog(null, "Contact updated successfully.");

} else {

JOptionPane.showMessageDialog(null, "Contact not found.");

}

clearFields(); // Clear input fields after editing

}

});

backButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

new Home(contactManager).setVisible(true);

dispose();

}

})

setVisible(true);

}

private void clearFields() {

nameField.setText("");

phoneField.setText("");

emailField.setText("");

}

}

**6. ContactManager  
Purpose**: This class manages the collection of contacts. It serves as a backend for storing and retrieving contacts.

**Key Functions**:

* Store contacts in an array or a list.
* Methods for adding, editing, deleting, and searching contacts.  
    
  **Methods**:
* addContact(Contact contact): Adds a new contact.
* getAllContacts(): Retrieves all contacts.

findContact(String name): Finds a contact by name  
  
package PhoneBook;

import java.util.LinkedList;

public class ContactManager {

private LinkedList<Contact> contacts;

public ContactManager() {

contacts = new LinkedList<>();

}

public void addContact(Contact contact) {

contacts.add(contact);

}

public Contact searchContact(String name) {

for (Contact contact : contacts) {

if (contact.getName().equalsIgnoreCase(name)) {

return contact;

}

}

return null;

}

public boolean editContact(String name, String newPhone, String newEmail) {

Contact contact = searchContact(name);

if (contact != null) {

contact.setPhone(newPhone);

contact.setEmail(newEmail);

return true; // Edit successful

}

return false; // Contact not found

}

public boolean deleteContact(String name) {

Contact contact = searchContact(name);

if (contact != null) {

contacts.remove(contact);

return true; // Deletion successful

}

return false;

}

public LinkedList<Contact> getAllContacts() {

return contacts;

}

}

**7. Home  
  
Purpose**: This class serves as the main entry point or home screen of the application, where users can navigate to other features like adding, deleting, searching, or viewing contacts.

**Key Functions**:

* Navigation buttons to direct users to different functionalities (e.g., add, delete, search).

package PhoneBook;

import javax.swing.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

public class Home extends JFrame {

private ContactManager contactManager;

public Home(ContactManager contactManager) {

this.contactManager = contactManager;

setTitle("Phone Book");

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setSize(400, 400);

setLayout(null);

// Create buttons for the main options

JButton addButton = new JButton("Add");

JButton searchButton = new JButton("Search");

JButton displayButton = new JButton("Display All");

JButton editButton = new JButton("Edit");

JButton deleteButton = new JButton("Delete");

JButton exitButton = new JButton("Exit");

addButton.setBounds(50, 50, 100, 40);

searchButton.setBounds(50, 100, 100, 40);

displayButton.setBounds(50, 150, 100, 40);

editButton.setBounds(50, 200, 100, 40);

deleteButton.setBounds(50, 250, 100, 40);

exitButton.setBounds(50, 300, 100, 40);

add(addButton);

add(searchButton);

add(displayButton);

add(editButton); add(deleteButton);

add(exitButton);

addButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

new AddContact(contactManager).setVisible(true);

dispose();

}

});

searchButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

new SearchContact(contactManager).setVisible(true);

dispose();

}

});

displayButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

new DisplayAllContacts(contactManager).setVisible(true);

dispose();

}

});

editButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

new EditContact(contactManager).setVisible(true); // Open EditContact screen

dispose();

}

});

deleteButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

new DeleteContact(contactManager).setVisible(true);

dispose();

}

});

exitButton.addActionListener(new ActionListener() {

public void actionPerformed(ActionEvent e) {

System.exit(0);

}

});

setVisible(true);

}

public static void main(String[] args) {

ContactManager contactManager = new ContactManager();

new Home(contactManager).setVisible(true);

}

#### } **8. Contact**

**Purpose**: This is a model class that defines the structure of a contact. It holds data like name, phone number, etc.

**Attributes**:

* + String name
  + String phoneNumber

package PhoneBook;

public class Contact {

private String name;

private String phone;

private String email;

public Contact(String name, String phone, String email) {

this.name = name;

this.phone = phone;

this.email = email;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getPhone() {

return phone;

}

public void setPhone(String phone) {

this.phone = phone;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

@Override

public String toString() {

return "Name: " + name + "\nPhone: " + phone + "\nEmail: " + email;

}

}

**Pseudocodes for Functionalities**

**1.Log In**

1.Start

2.Create window titled "Login page"

3.Set window size to 350x200

4.Set layout to null

5.Create label for "Username"

6.Create text field for username

7.Create label for "Password"

8.Create password field for password

9.Create "Login" button

10.When "login" button clicked:

a.Get username and password from fields

b.If username is "Admin" AND password is "password":

i.Show "Login Successful"

ii.Create contactManager

iii.Open Home screen

iv.Close login window

c.Else :

i.Show "Invalid credentials!"

11.Show window

12.End

**2.Home**

1.Start

2.Create window titled "Phone Book"

3.Set window size to 400x400

4.Set layout to null

5.Create buttons for:

a. Add

b. Search

c. Display All

d. Edit

e. Delete

f. Exit

6.Set position for each button

7.Add buttons to window

8.When "Add" button is clicked:

a. Open Add Contact screen

b. Close Home window

9.When "Search" button is clicked:

a. Open Search Contact screen

b. Close Home window

10.When "Display All" button clicked:

a. Open Display All Contacts screen

b. Close Home window

11.When "Edit" button is clicked:

a. Open Edit Contact screen

b. Close Home window

12.When "Delete" button clicked:

a. Open Delete Contact screen

b. Close Home window

13.When "Exit" button clicked:

a. Exit application

14.Show window

15.End

**3.Add Contact**

1.Start

2.Initialize contactManager

3.Create window titled "Add Contact"

4.Create fields for Name , Phone , Email

5.Create "Save" button

6.Create "Back" button

7.When "Save" button clicked:

8.Get Name , Phone , Email from fields

9.Add contact to contactManager

10.Show "Contact added successfully" message

11.Clear input fields

12.When "Back" button clicked:

13.Open Home screen

14.Close addContact window

15 Show addContact window

16.End

**4.Search Contact**

1.Start

2.Initialize contactManager

3.Create window titled "Search Contact"

4.Create field for name input

5.Create "Search" button

6.Create "Back" button

7.Create area for displaying results

8.When "Search" button clicked :

9.Get name from input field

10.Search contact in contactManager

11.If contact found:

12.Displaycontact details

13.Else

14.Display "Contact not found."

15.Endif

16.When "Back" button clicked:

17.Open Home screen

18.Close searchContact window

19.Show searchContact window

20.End

**5.Display**

1.Start

2.Initialize contactManager

3.Create window titled "All contacts"

4.Set window size to 400x300

5.Create text area for contacts

6.Call displayContacts()

7.Create scroll pane for text area

8.Add scroll pane to window

9.Create "Back" button

10.When "Back" button clicked:

a.Go back to home screen

11.Show window

12.End

**6.Edit Contact**

1.Start

2.Initialize contactManager

3.Create window titled "Edit Contact"

4.Set window size to 400x300

5.Create fields for name , new phone , and new email

6.Create "Edit" button

7.Create "Back" button

8.When "Edit" button clicked:

a. Get name , new phone and new email from fields

b. Call contactManager to edit contact

c. If contact updated :

I. Show "Contact updated successfully"

d. Else:

i: Show:"Cotac not found"

e. Clear input fields

9.When "Back" button clicked:

a.Go back to Home screen

10.Show window

11.End

**7.Delete Contact**

1.Start

2.Create window titled "Delete Contact"

3.Set window size to 400x200

4.Set layout to null

5.Create label "Enter Name to delete"

6.Create input field for name

7.Create buttons:

a.Delete

b.Back

8.Set positions for label , input field and buttons

9.Add components to the window

10.When "Delete" button clicked:

a. Get name from input field

b. Attempt to delete contact from contact manager

c. If deleted , show "Contact deleted successfully."

d. If not found , show "Contact not found."

e. Clear input field

11.When "Back" button clicked:

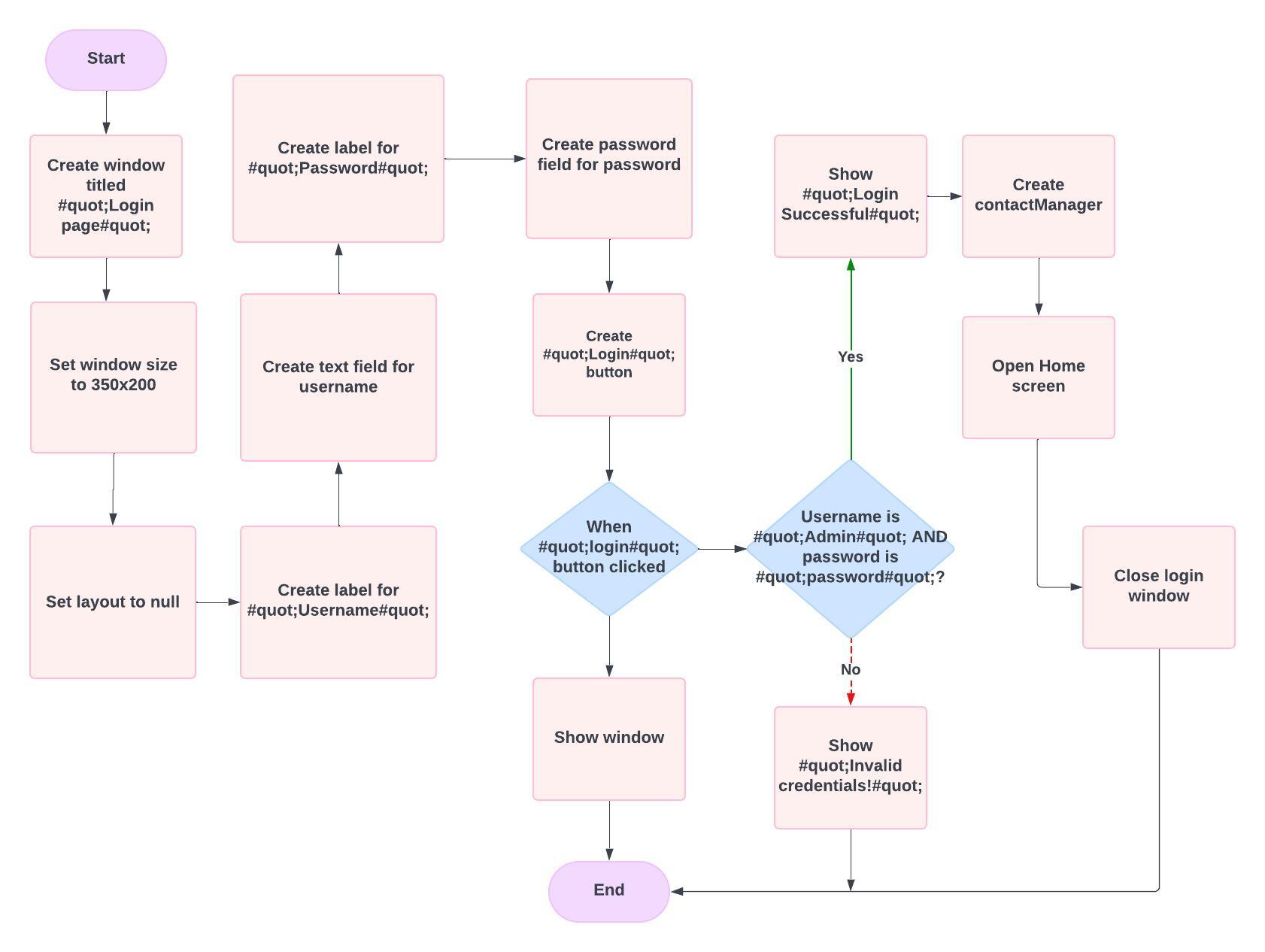
a.Open Home window

b.Close Delete Contact window

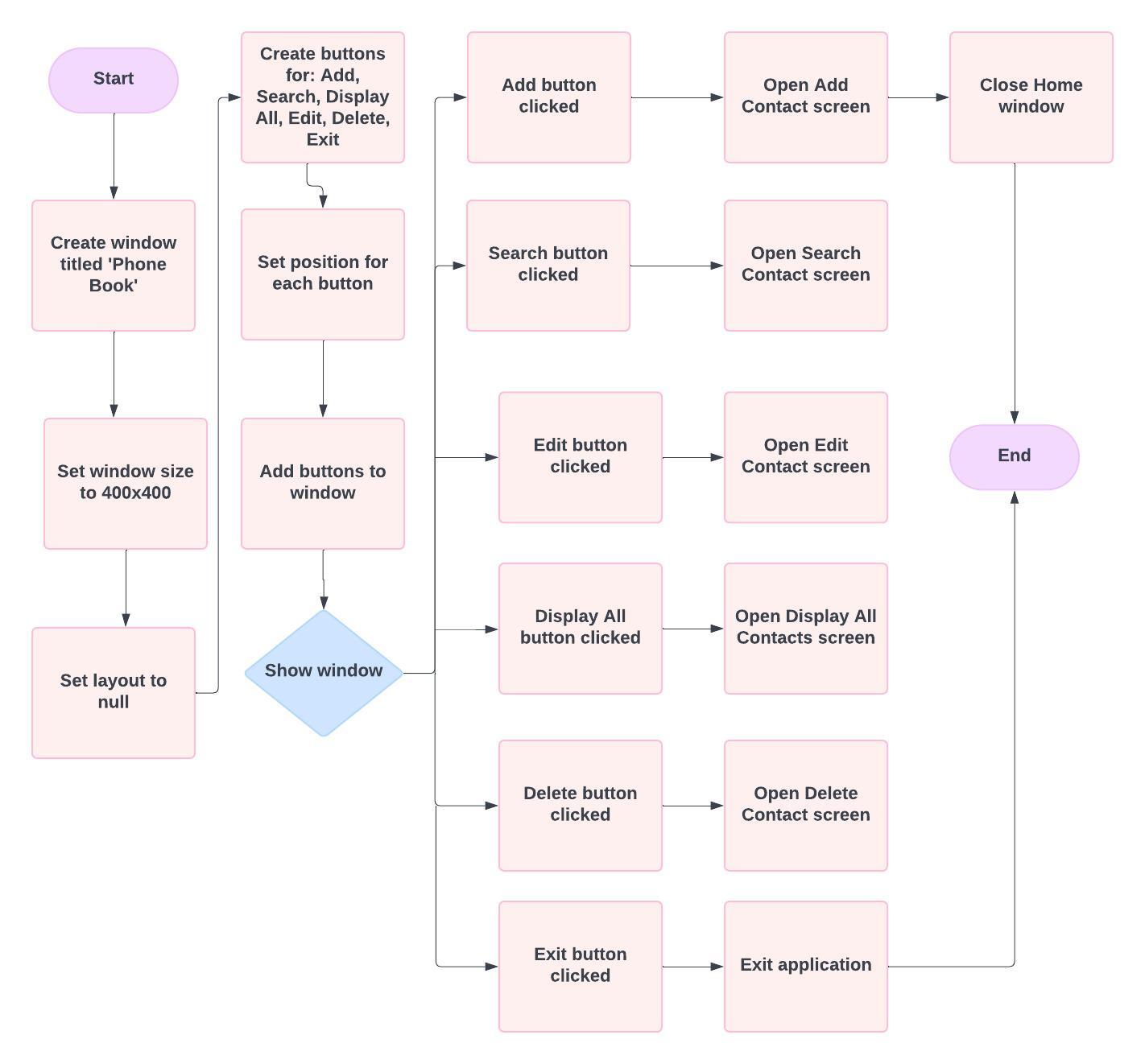
12.Show the window

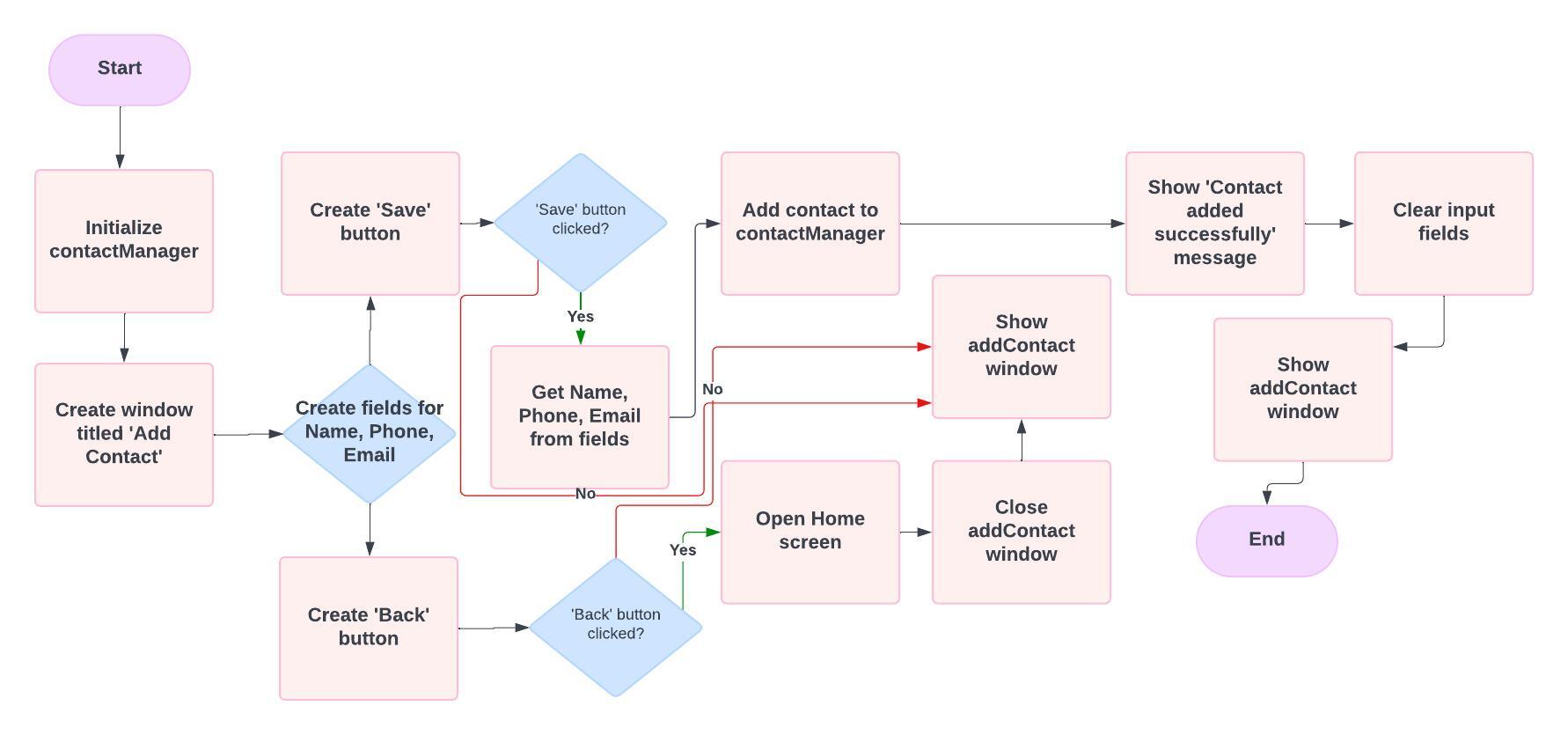
13.End

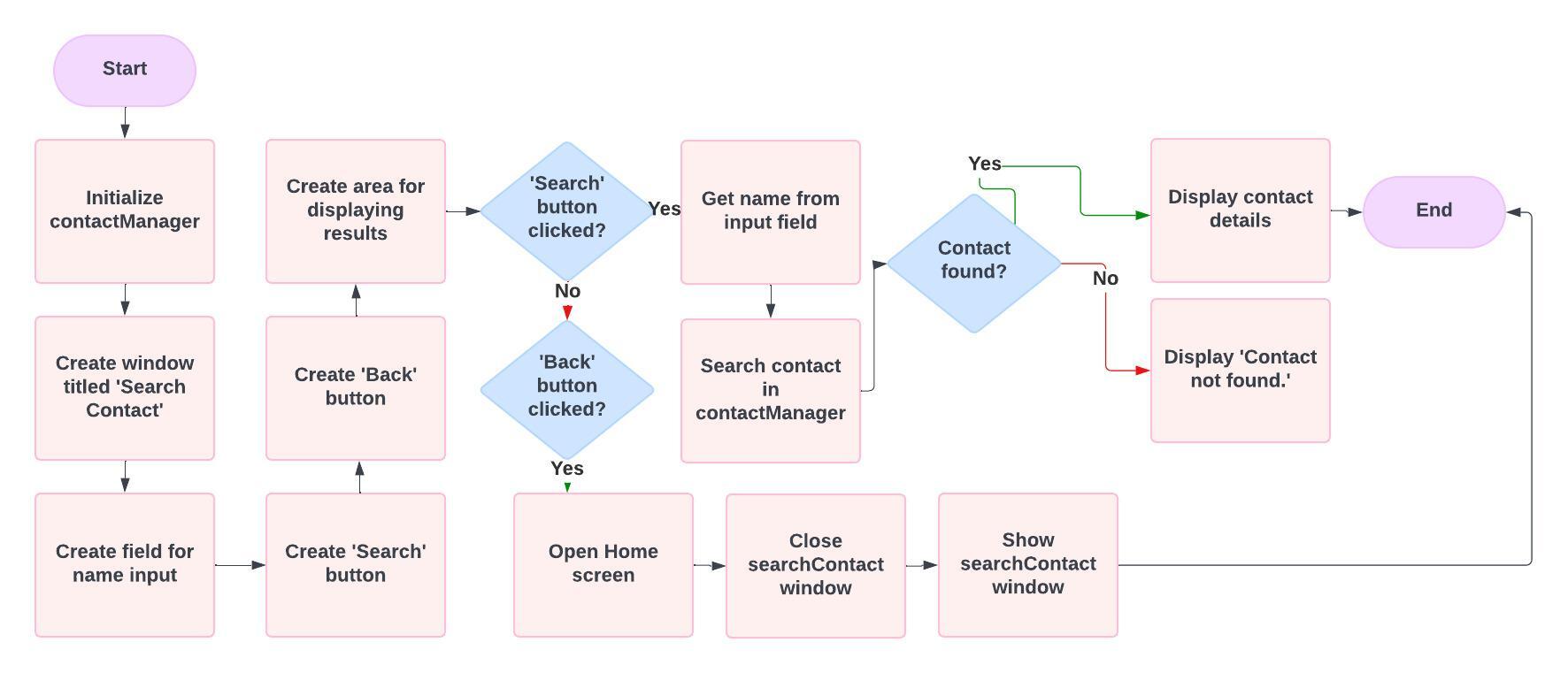
**Flowcharts**

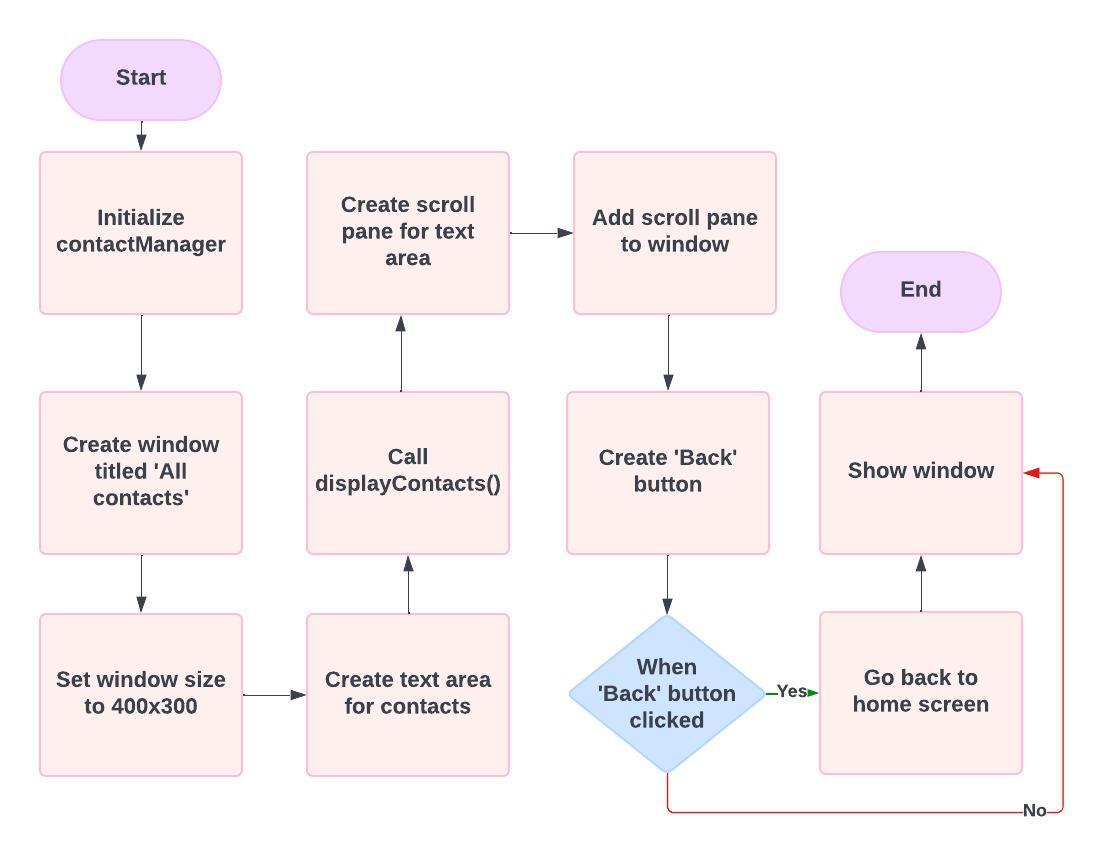
***Log In***

***Home Flowchart***

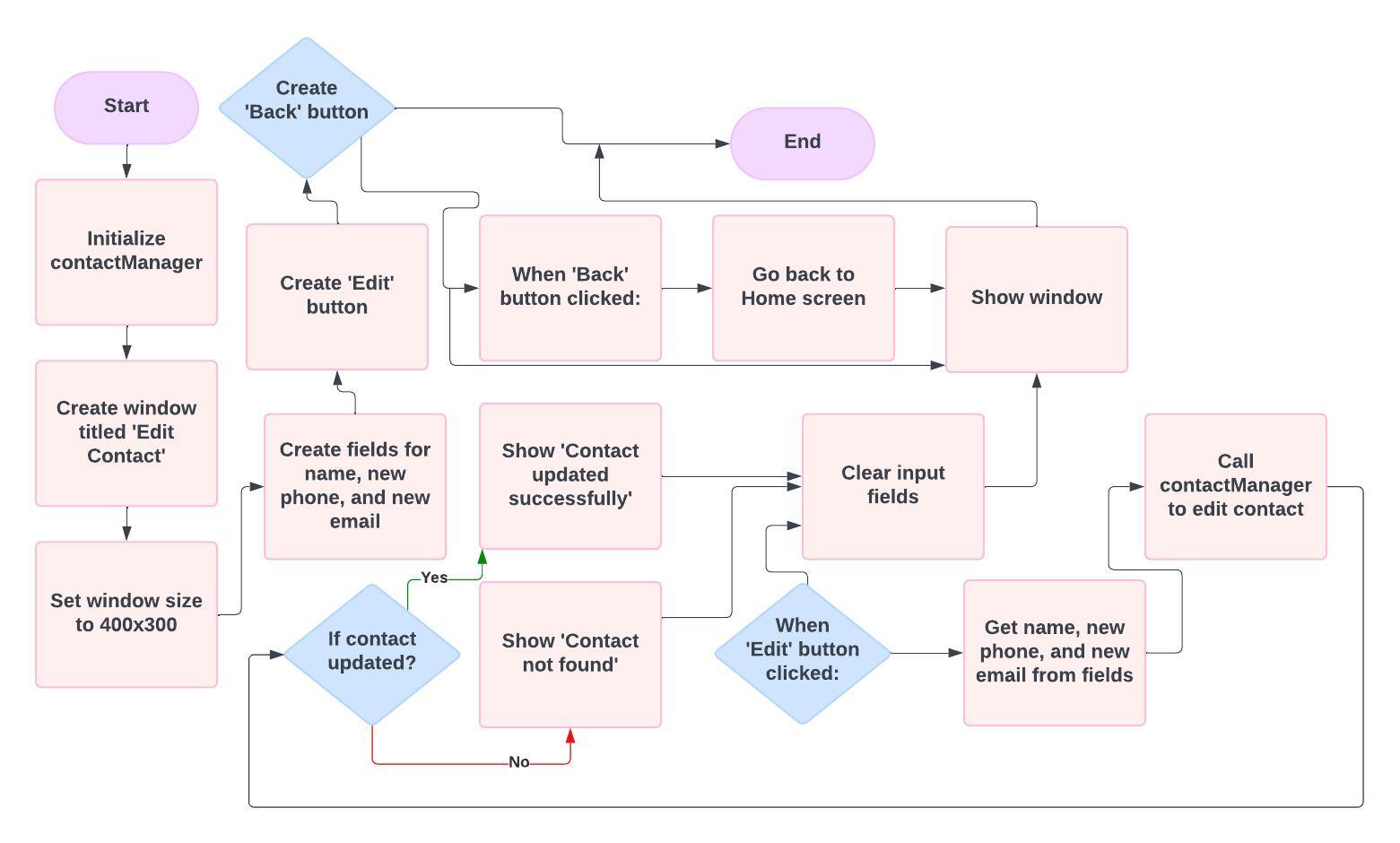


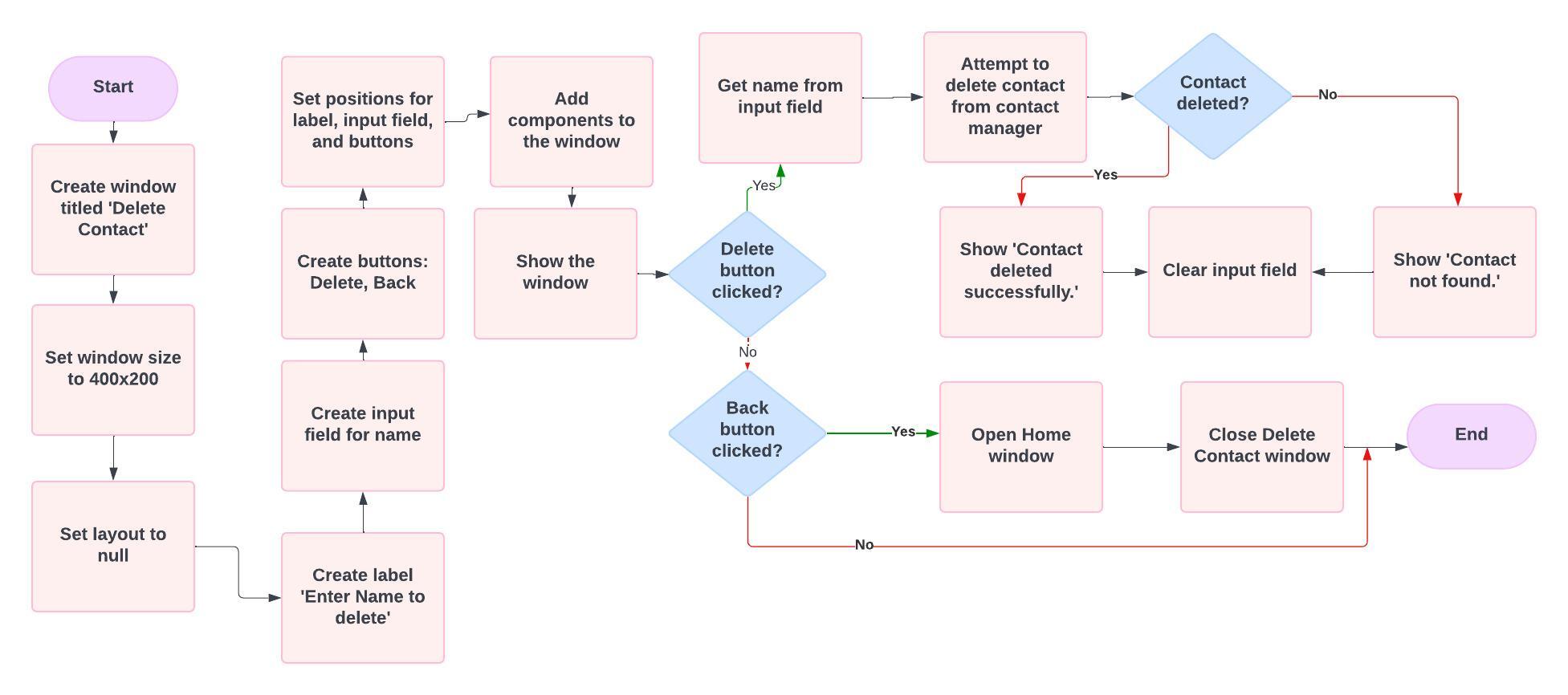
***Add Contact Flowchart***

***Search Contact Flowchart***

***Display Flowchart***

***Edit Contact Flowchart***



***Delete Contact Flowchart***

**PhoneBook Management System**

## **Project Overview**

The **PhoneBook Management System** we created, is a Java-based application which is created to manage a list of contacts using an ArrayList. It provides basic functionalities such as adding, editing, searching, deleting, and displaying contact information. The system operates without a database, instead using an ArrayList to store contacts dynamically during runtime.

## Features

* **Add Contact**: Allows users to create and store new contact information in an ArrayList.
* **Edit Contact**: Enables users to modify existing contact information stored in the list.
* **Delete Contact**: Removes contact details from the ArrayList.
* **Search Contact**: Provides functionality to search for a contact by name or other criteria.
* **Display All Contacts**: Displays a list of all stored contacts within the ArrayList.
* **Login Functionality**: Simple user authentication before accessing the phonebook management system.

## Installation and Setup

1. **Clone the repository**: https://github.com/WHYNOTJOSH/PhonebookProject-NUST.git
2. **Run the application**:
   * Open the project in NetBeans or your preferred IDE.
   * Build and run the project.

## File Structure

Here’s an overview of the key files in the project:

* **AddContact.java**: Manages the functionality for adding new contacts to the ArrayList.
* **ConnectionClass.java**: [Remove or update this if not used, or explain any role it may have].
* **Contact.java**: Represents the contact entity with fields like name, phone number, and email.
* **ContactManager.java**: Contains the logic to manage contacts stored in the ArrayList.
* **DeleteContact.java**: Handles the removal of contacts from the ArrayList.
* **DisplayAllContacts.java**: Displays all the contacts stored in the phonebook.
* **EditContact.java**: Handles the functionality for updating contact details.
* **Home.java**: The main page or dashboard for the application after user login.
* **Login.java**: Manages user authentication to access the system.
* **SearchContact.java**: Provides functionality to search for contacts within the ArrayList.

## Technologies We Used

* **Java**: Core programming language used for developing the system.
* **ArrayList**: Java's ArrayList is used to dynamically store and manage the list of contacts during runtime.

# 