



**NAMIBIA UNIVERSITY
OF SCIENCE AND TECHNOLOGY**

**Faculty of Computing & Informatics
Department of Software Engineering**

Phone Book Application

Name and initials	Last Name	Student Number
James R.J.C	Kamerika	221119434
Sharaun J.S.A	Katjizumo	221119396
Josia J.S	Taapopi	224082450
Nockia H	Kaapehi	224060295
Naledi D.V	Ndjahera	224063162
Gabbriela L	Kapitango	224086316

System code

```
import javax.swing.*;

import java.awt.*;

import java.io.*;

import java.util.ArrayList;


// Represents a single contact

class Contact implements Serializable {

    String name, phoneNumber, address;


    public Contact(String name, String phoneNumber, String address) {

        this.name = name;

        this.phoneNumber = phoneNumber;

        this.address = address;

    }


    @Override

    public String toString() {

        return "Name: " + name + ", Phone: " + phoneNumber + ", Address: " + address;

    }

}


// Manages the list of contacts

class PhoneBook implements Serializable {

    private ArrayList<Contact> contacts;

    private static final String FILE_NAME = "contacts.dat"; // File for storing contacts


    public PhoneBook() {

        contacts = new ArrayList<>();

    }

}
```

```
        loadContacts(); // Load contacts from file
    }

    // Add a new contact and save to file
    public void insertContact(String name, String phoneNumber, String address) {
        contacts.add(new Contact(name, phoneNumber, address));
        saveContacts(); // Save contacts after adding a new one
    }

    // Display all contacts
    public String displayContacts() {
        if (contacts.isEmpty()) return "Phonebook is empty.\n";
        StringBuilder contactsDisplay = new StringBuilder("Contacts:\n");
        for (Contact contact : contacts) {
            contactsDisplay.append(contact.toString()).append("\n");
        }
        return contactsDisplay.toString();
    }

    // Search for a contact by name
    public String searchContact(String name) {
        for (Contact contact : contacts) {
            if (contact.name.equalsIgnoreCase(name)) {
                return "Found: " + contact.toString() + "\n";
            }
        }
        return "Contact not found.\n";
    }

    // Delete a contact by name
    public String deleteContact(String name) {
```

```
for (Contact contact : contacts) {  
    if (contact.name.equalsIgnoreCase(name)) {  
        contacts.remove(contact);  
        saveContacts(); // Save contacts after deletion  
        return "Deleted: " + name + "\n";  
    }  
}  
return "Contact not found.\n";  
}
```

// Get a contact by name

```
public Contact getContactByName(String name) {  
    for (Contact contact : contacts) {  
        if (contact.name.equalsIgnoreCase(name)) {  
            return contact;  
        }  
    }  
    return null; // Return null if not found  
}
```

// Save contacts to a file

```
private void saveContacts() {  
    try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(FILE_NAME))) {  
        oos.writeObject(contacts);  
    } catch (IOException e) {  
        e.printStackTrace();  
    }  
}
```

// Load contacts from a file

```
@SuppressWarnings("unchecked")
```

```
private void loadContacts() {  
    File file = new File(FILE_NAME);  
    if (file.exists()) {  
        try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(file))) {  
            contacts = (ArrayList<Contact>) ois.readObject();  
        } catch (IOException | ClassNotFoundException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

// Main application frame

```
public class PhoneBookApp extends JFrame {  
    private PhoneBook phoneBook = new PhoneBook();  
    private JTextArea displayArea = new JTextArea();  
    private boolean isAdmin;  
    private Color currentBackgroundColor;  
    private Color currentTextColor;  
  
    public PhoneBookApp(boolean isAdmin) {  
        this.isAdmin = isAdmin;  
        setTitle("PB-App Dashboard");  
        setSize(700, 500);  
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
        setLayout(new BorderLayout());  
  
        // Initial theme  
        setTheme("darkgrey");  
  
        displayArea.setFont(new Font("SansSerif", Font.PLAIN, 14));
```

```
displayArea.setForeground(currentTextColor);  
displayArea.setBackground(currentBackgroundColor);
```

```
JLabel titleLabel = new JLabel("D-Phone Book", JLabel.CENTER);  
titleLabel.setFont(new Font("SansSerif", Font.BOLD, 24));  
titleLabel.setForeground(currentTextColor);  
add(titleLabel, BorderLayout.NORTH);
```

```
JPanel actionPanel = new JPanel();
```

```
 JButton themeButton = createStyledButton("Change Theme");  
themeButton.addActionListener(e -> changeTheme());
```

```
 JButton searchButton = createStyledButton("Search Contact");  
 JButton displayButton = createStyledButton("Display Contacts");  
 JButton backButton = createStyledButton("Back");
```

```
searchButton.addActionListener(e -> searchContact());  
displayButton.addActionListener(e -> displayArea.setText(phoneBook.displayContacts()));  
backButton.addActionListener(e -> goBack());
```

```
actionPanel.add(themeButton);  
actionPanel.add(searchButton);  
actionPanel.add(displayButton);  
actionPanel.add(backButton);
```

```
// If admin, add options for adding, deleting, and updating contacts
```

```
if (isAdmin) {
```

```
    JButton addButton = createStyledButton("Add Contact");  
    JButton deleteButton = createStyledButton("Delete Contact");  
    JButton updateButton = createStyledButton("Update Contact"); // New update button
```

```
addButton.addActionListener(e -> addContact());

deleteButton.addActionListener(e -> deleteContact());

updateButton.addActionListener(e -> updateContact()); // Action for the update button


actionPanel.add(addButton);

actionPanel.add(deleteButton);

actionPanel.add(updateButton); // Add the update button to the panel
}


add(actionPanel, BorderLayout.SOUTH);

add(new JScrollPane(displayArea), BorderLayout.CENTER);
}


// Set the theme
private void setTheme(String theme) {
    switch (theme.toLowerCase()) {
        case "darkgrey":
            currentBackgroundColor = Color.DARK_GRAY;
            currentTextColor = Color.WHITE;
            break;
        case "purple":
            currentBackgroundColor = new Color(128, 0, 128); // Purple
            currentTextColor = Color.WHITE;
            break;
        case "light":
            currentBackgroundColor = Color.LIGHT_GRAY;
            currentTextColor = Color.BLACK;
            break;
        case "pink":
            currentBackgroundColor = Color.PINK;
```

```
        currentTextColor = Color.BLACK;

        break;
    case "red":
        currentBackgroundColor = Color.RED;
        currentTextColor = Color.WHITE;
        break;
    default:
        currentBackgroundColor = Color.LIGHT_GRAY; // Default to light grey
        currentTextColor = Color.BLACK;
    }

    getContentPane().setBackground(currentBackgroundColor);
    displayArea.setBackground(currentBackgroundColor);
    displayArea.setForeground(currentTextColor);
}

// Change theme when button is clicked
private void changeTheme() {
    String[] themes = {"darkgrey", "purple", "light", "pink", "red"};
    String selectedTheme = (String) JOptionPane.showInputDialog(
        this,
        "Select a theme:",
        "Theme Selection",
        JOptionPane.QUESTION_MESSAGE,
        null,
        themes,
        themes[0]
    );

    if (selectedTheme != null) {
        setTheme(selectedTheme);
    }
}
```



```
    }  
}  
  
// Button styling  
private JButton createStyledButton(String text) {  
    JButton button = new JButton(text);  
    button.setFocusPainted(false);  
  
    button.addMouseListener(new java.awt.event.MouseAdapter() {  
        @Override  
        public void mouseEntered(java.awt.event.MouseEvent evt) {  
            button.setBackground(Color.LIGHT_GRAY);  
        }  
  
        @Override  
        public void mouseExited(java.awt.event.MouseEvent evt) {  
            button.setBackground(null);  
        }  
    });  
  
    return button;  
}  
  
// Navigate back to login  
private void goBack() {  
    dispose();  
    SwingUtilities.invokeLater(() -> new LoginScreen().setVisible(true));  
}  
  
// Add new contact  
private void addContact() {
```

```
String name = JOptionPane.showInputDialog(this, "Enter Name:");
String phone = JOptionPane.showInputDialog(this, "Enter Phone:");
String address = JOptionPane.showInputDialog(this, "Enter Address:");

if (isEmpty(name, phone, address)) return;

if (!name.matches("[a-zA-Z]+")) {
    JOptionPane.showMessageDialog(this, "Name must contain only letters.");
    return;
}

phoneBook.insertContact(name, phone, address);
displayArea.setText("Contact added: " + name);
}

// Update an existing contact
private void updateContact() {
    String name = JOptionPane.showInputDialog(this, "Enter Name of the Contact to Update:");
    if (isEmpty(name)) return;

    Contact existingContact = phoneBook.getContactByName(name);
    if (existingContact != null) {
        String newPhone = JOptionPane.showInputDialog(this, "Enter New Phone:",
            existingContact.phoneNumber);

        String newAddress = JOptionPane.showInputDialog(this, "Enter New Address:",
            existingContact.address);

        if (isEmpty(newPhone, newAddress)) return;

        // Remove old contact and add updated contact
        phoneBook.deleteContact(name);
        phoneBook.insertContact(existingContact.name, newPhone, newAddress);
    }
}
```

```
        displayArea.setText("Contact updated: " + existingContact.name);
    } else {
        displayArea.setText("Contact not found.");
    }
}

// Search for a contact
private void searchContact() {
    String name = JOptionPane.showInputDialog(this, "Enter Name to Search:");

    if (isEmpty(name)) return;

    if (!name.matches("[a-zA-Z]+")) {
        JOptionPane.showMessageDialog(this, "Name must contain only letters.");
        return;
    }

    displayArea.setText(phoneBook.searchContact(name));
}

// Delete a contact
private void deleteContact() {
    String name = JOptionPane.showInputDialog(this, "Enter Name to Delete:");

    if (isEmpty(name)) return;

    if (!name.matches("[a-zA-Z]+")) {
        JOptionPane.showMessageDialog(this, "Name must contain only letters.");
        return;
    }
}
```

```
        displayArea.setText(phoneBook.deleteContact(name));
    }
}
```

```
// Check for empty fields
```

```
private boolean isEmpty(String... fields) {
    for (String field : fields) {
        if (field == null || field.trim().isEmpty()) {
            JOptionPane.showMessageDialog(this, "Please fill in the empty fields.");
            return true;
        }
    }
    return false;
}
```

```
public static void main(String[] args) {
    SwingUtilities.invokeLater(() -> new LoginScreen().setVisible(true));
}
}
```

```
// Login screen for user authentication
```

```
class LoginScreen extends JFrame {
    public LoginScreen() {
        setTitle("Login");
        setSize(350, 200);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new GridLayout(2, 2));
        getContentPane().setBackground(Color.decode("#ADD8E6")); // Baby blue background

        JButton adminLoginButton = new JButton("Admin Login");
        JButton userLoginButton = new JButton("User Login");
    }
}
```

```

// Admin login action

adminLoginButton.addActionListener(e -> {

    String password = JOptionPane.showInputDialog(this, "Enter Admin Password:");

    if (password != null && !password.trim().isEmpty()) {

        // Check if password is set (you can store this in a better way, but for now, we use a static
        example)

        if (password.equals("admin123")) { // Use a predefined admin password

            new PhoneBookApp(true).setVisible(true);

            dispose();

        } else {

            JOptionPane.showMessageDialog(this, "Invalid password!");

        }

    }

});

// User login action

userLoginButton.addActionListener(e -> new PhoneBookApp(false).setVisible(true));

add(adminLoginButton);

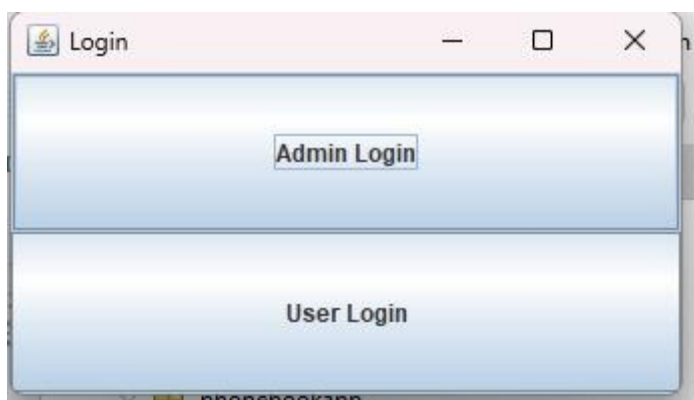
add(userLoginButton);

}

}

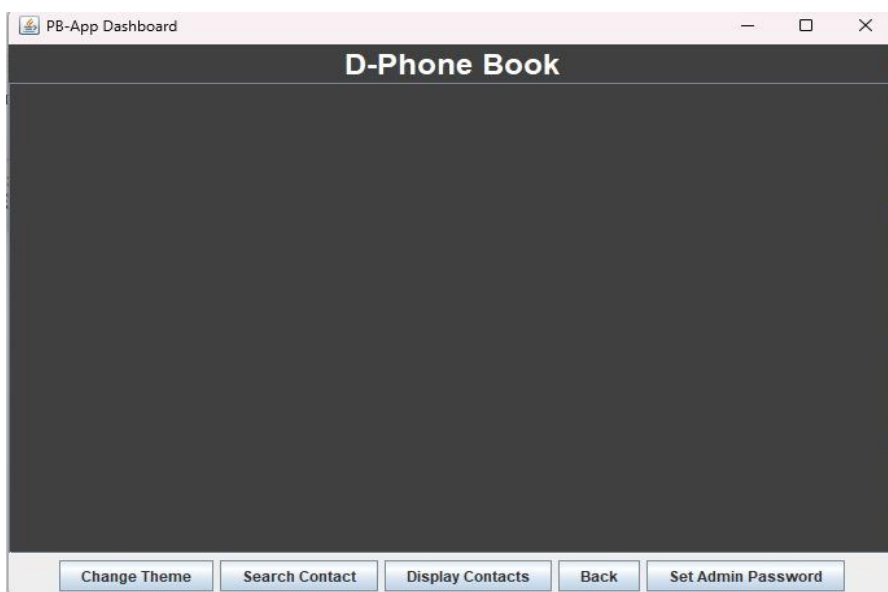
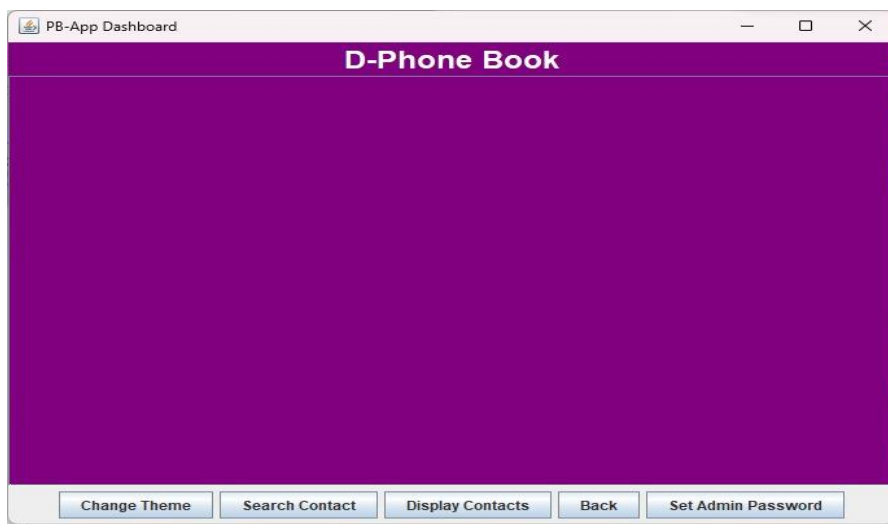
```

USER INTERFACE (VIEW)



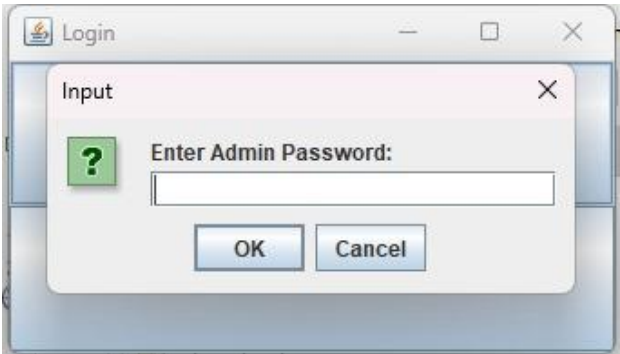
Our application allows both users and admins to log In

THEME INTERFACE

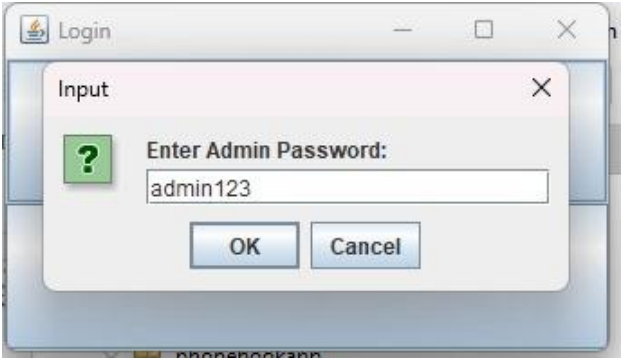


We added features that suit our users preservation.

Admins GUI

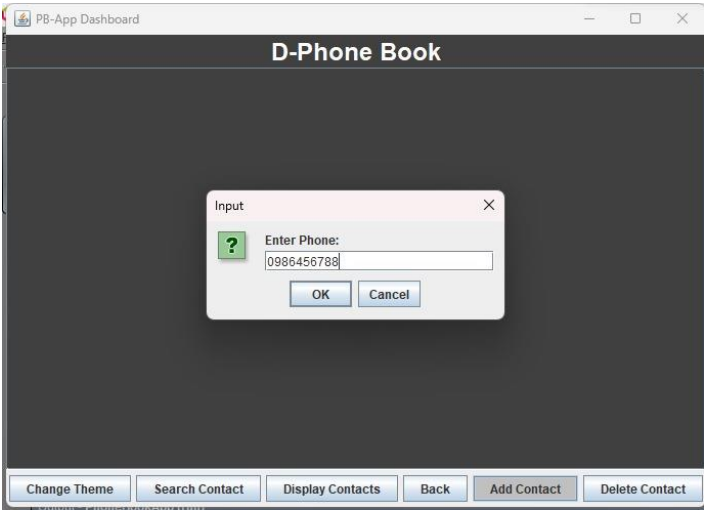


A screenshot of a 'Login' window. It contains an 'Input' dialog box with a green question mark icon, the text 'Enter Admin Password:', an empty text field, and 'OK' and 'Cancel' buttons.



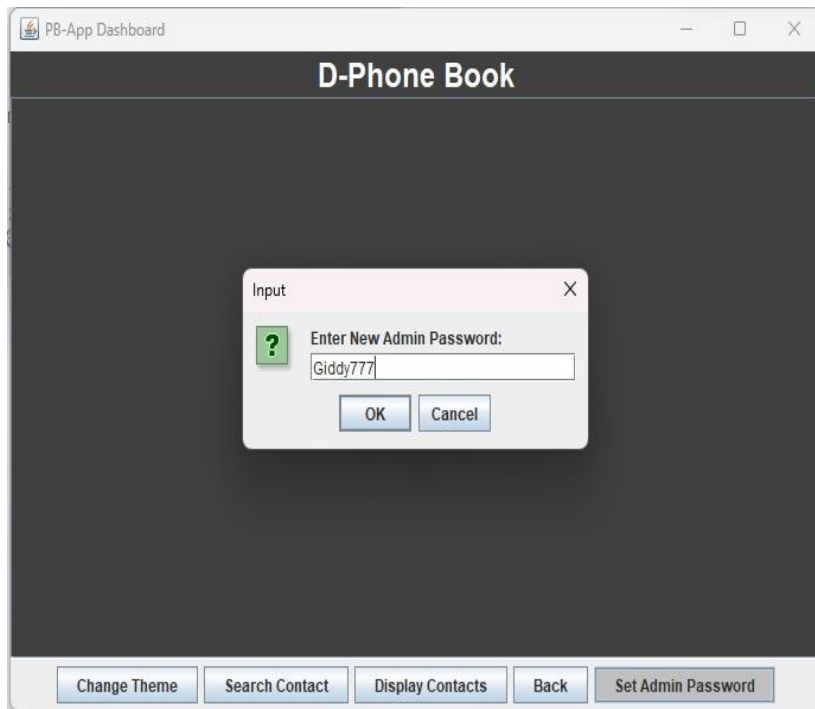
A screenshot of the same 'Login' window, but the 'Input' dialog box now shows the password 'admin123' entered in the text field.

LOGGING
M,4TJI34OITJO4T

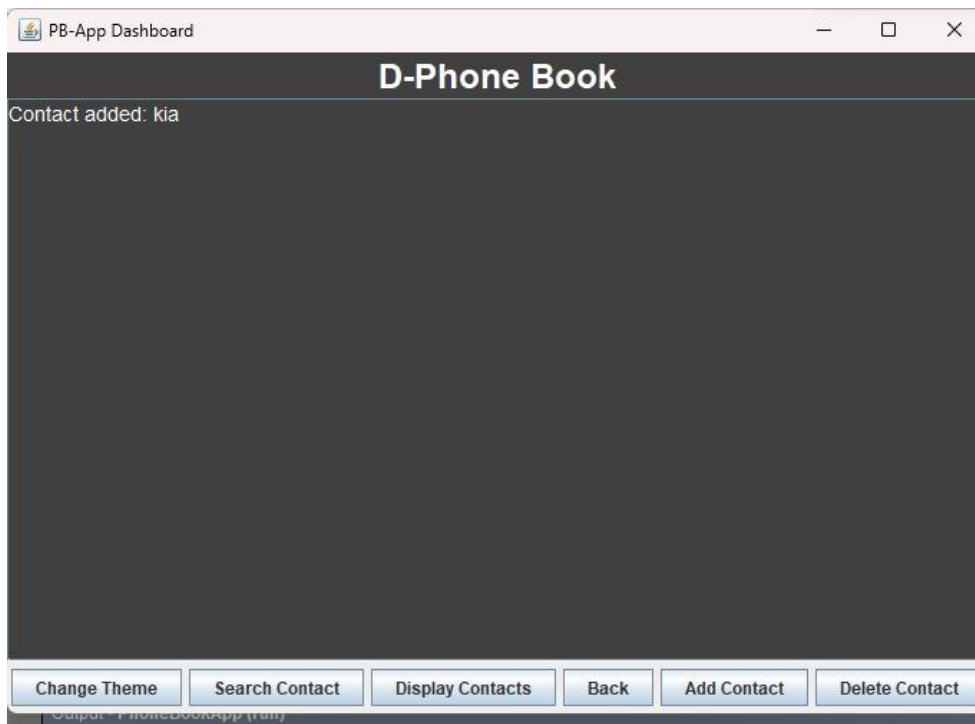


A screenshot of the 'PB-App Dashboard' window titled 'D-Phone Book'. It features a dark background with a central 'Input' dialog box asking to 'Enter Phone:' with the number '09886456788' entered. At the bottom, there is a row of buttons: 'Change Theme', 'Search Contact', 'Display Contacts', 'Back', 'Add Contact', and 'Delete Contact'.

Adding a user using
phone number



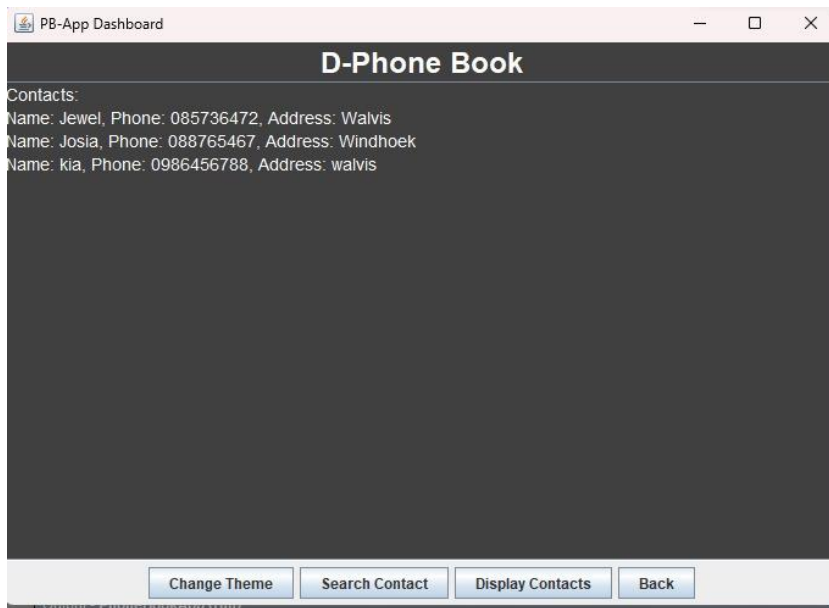
Resetting
password as an
admin



Add
contact by
name

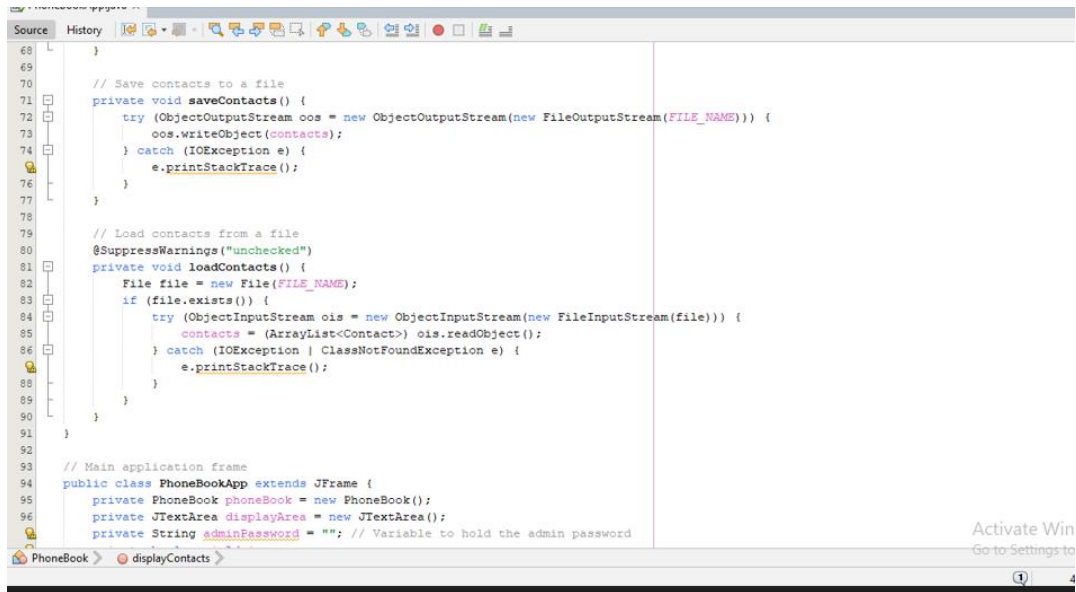


Searching
by name



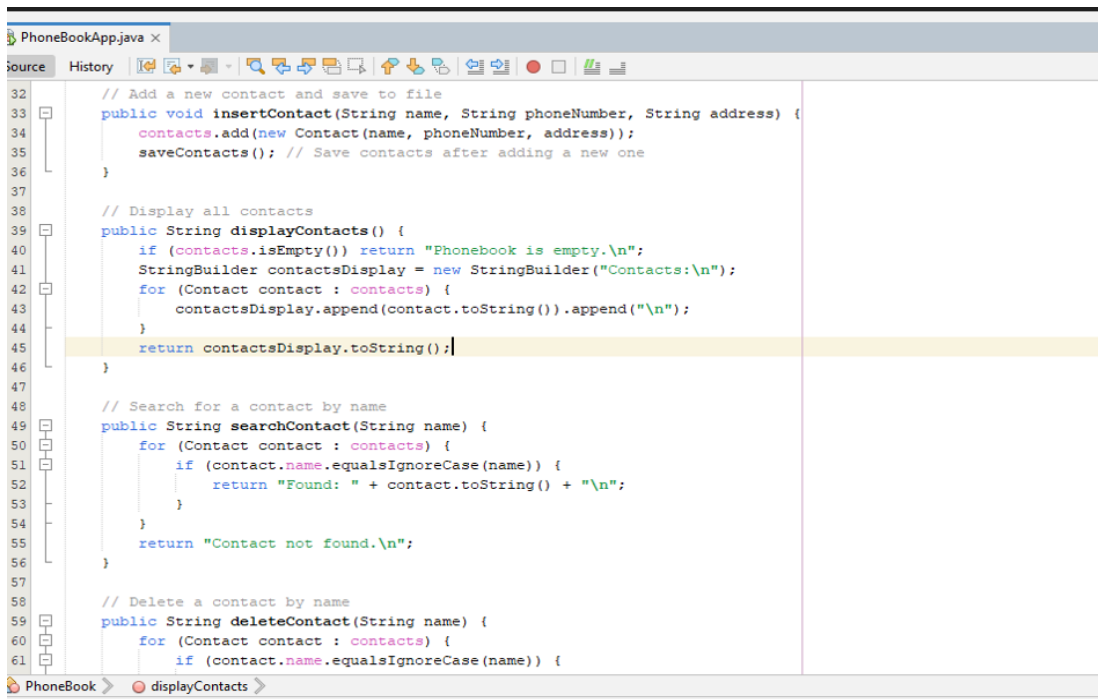
Display all Users
this is for users

Our code in Netbeans

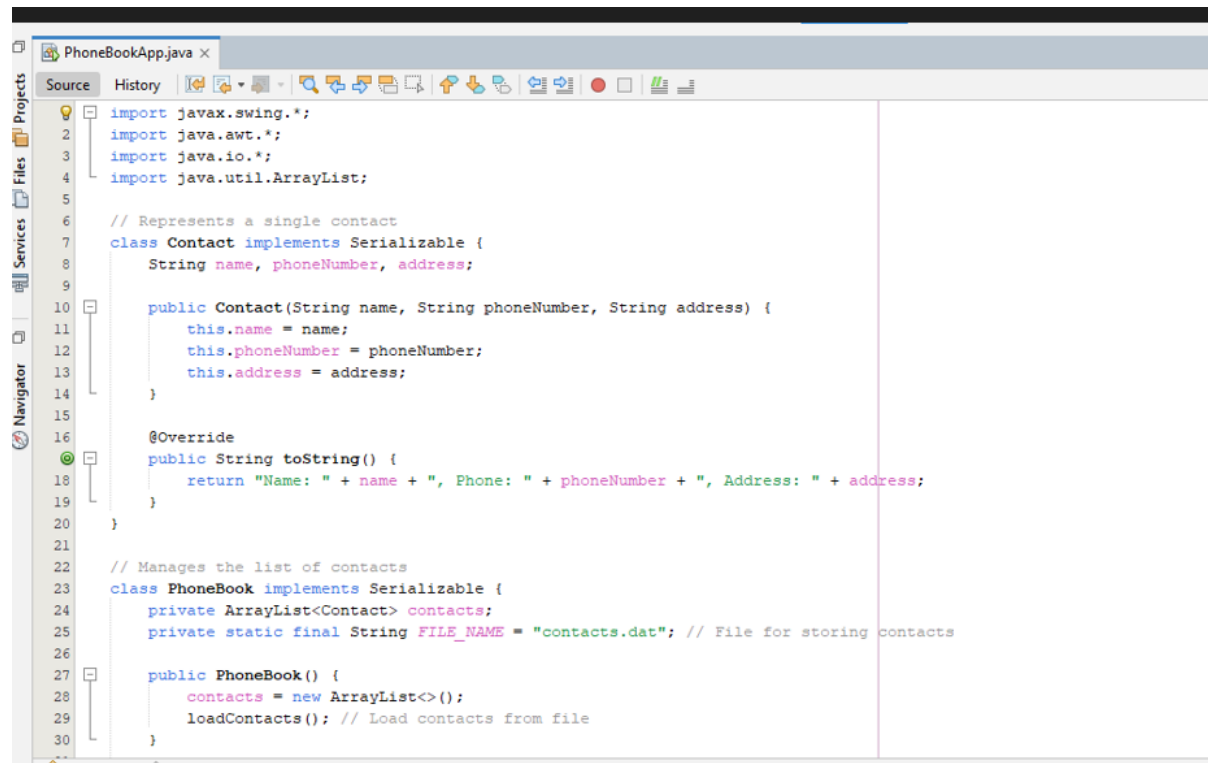


```
68     }
69
70     // Save contacts to a file
71     private void saveContacts() {
72         try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(FILE_NAME))) {
73             oos.writeObject(contacts);
74         } catch (IOException e) {
75             e.printStackTrace();
76         }
77     }
78
79     // Load contacts from a file
80     @SuppressWarnings("unchecked")
81     private void loadContacts() {
82         File file = new File(FILE_NAME);
83         if (file.exists()) {
84             try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(file))) {
85                 contacts = (ArrayList<Contact>) ois.readObject();
86             } catch (IOException | ClassNotFoundException e) {
87                 e.printStackTrace();
88             }
89         }
90     }
91
92     // Main application frame
93     public class PhoneBookApp extends JFrame {
94         private PhoneBook phoneBook = new PhoneBook();
95         private JTextArea displayArea = new JTextArea();
96         private String adminPassword = ""; // Variable to hold the admin password
97     }
```

Activate Win
Go to Settings to



```
32     // Add a new contact and save to file
33     public void insertContact(String name, String phoneNumber, String address) {
34         contacts.add(new Contact(name, phoneNumber, address));
35         saveContacts(); // Save contacts after adding a new one
36     }
37
38     // Display all contacts
39     public String displayContacts() {
40         if (contacts.isEmpty()) return "Phonebook is empty.\n";
41         StringBuilder contactsDisplay = new StringBuilder("Contacts:\n");
42         for (Contact contact : contacts) {
43             contactsDisplay.append(contact.toString()).append("\n");
44         }
45         return contactsDisplay.toString();
46     }
47
48     // Search for a contact by name
49     public String searchContact(String name) {
50         for (Contact contact : contacts) {
51             if (contact.name.equalsIgnoreCase(name)) {
52                 return "Found: " + contact.toString() + "\n";
53             }
54         }
55         return "Contact not found.\n";
56     }
57
58     // Delete a contact by name
59     public String deleteContact(String name) {
60         for (Contact contact : contacts) {
61             if (contact.name.equalsIgnoreCase(name)) {
```



The screenshot shows an IDE window titled "PhoneBookApp.java". The code defines two classes: `Contact` and `PhoneBook`. The `Contact` class implements `Serializable` and has attributes `name`, `phoneNumber`, and `address`. It includes a constructor and a `toString` method. The `PhoneBook` class also implements `Serializable` and manages a list of `Contact` objects, with a static final `FILE_NAME` set to "contacts.dat". It includes a constructor that initializes the list and loads contacts from the file.

```
1 import javax.swing.*;
2 import java.awt.*;
3 import java.io.*;
4 import java.util.ArrayList;
5
6 // Represents a single contact
7 class Contact implements Serializable {
8     String name, phoneNumber, address;
9
10    public Contact(String name, String phoneNumber, String address) {
11        this.name = name;
12        this.phoneNumber = phoneNumber;
13        this.address = address;
14    }
15
16    @Override
17    public String toString() {
18        return "Name: " + name + ", Phone: " + phoneNumber + ", Address: " + address;
19    }
20 }
21
22 // Manages the list of contacts
23 class PhoneBook implements Serializable {
24     private ArrayList<Contact> contacts;
25     private static final String FILE_NAME = "contacts.dat"; // File for storing contacts
26
27     public PhoneBook() {
28         contacts = new ArrayList<>();
29         loadContacts(); // Load contacts from file
30     }
31 }
```