



DSA521S GROUP PROJECT 2024

# CONTACT BUDDY PHONEBOOK APPLICATION PROJECT

**DESIGNED BY:**

Daniel R J Tjimuku: 224001884 (Group Leader)

Peter Kamati 224080806

Denzel Naftali: 224084143

Elton Lisho: 216070325

Nambinga Naikuti: 224016288

Shoopala Lison Shoole: 224073249

Lecturer: Mr. S. Tjiraso

NUST: Department of Computing And Informatics

# SECTION A

## Project Overview

### 1.1. Project Title

Simple Phonebook Application

### 1.2. Scope

The Simple Phonebook Application is a desktop-based system designed to manage contact information. It provides basic phonebook functionalities, including adding, searching, updating, deleting, and displaying contacts. The application uses basic data structures to store contact information and a simple graphical user interface (GUI) to interact with the user. This project is suitable for users looking for an easy-to-use, lightweight contact management system with standard operations.

## 2. User Requirements

### 2.1. Functional Requirements

- **Insert Contact:** The user should be able to add a contact with a name and phone number.
- **Search Contact:** The user should be able to search for a contact by name and view the corresponding phone number.
- **Delete Contact:** The user should be able to delete a contact by specifying the name.
- **Update Contact:** The user should be able to update the phone number of an existing contact by specifying the name.
- **Display All Contacts:** The user should be able to view all saved contacts in the system.

### 2.2. Non-Functional Requirements

- **Usability:** The system should provide an intuitive interface using a graphical user interface (GUI).
- **Performance:** The system should handle basic operations like adding, searching, updating, and deleting contacts efficiently with a small number of records (less than 1000 contacts).
- **Maintainability:** The system should be easily extendable if further functionalities are added, such as sorting contacts or saving the phonebook to a file.

## 3. System Functionalities

The system offers the following core functionalities:

- **Insert Contact:** Add a new contact to the phonebook by providing a name and a phone number.
- **Search Contact:** Search for an existing contact using the name.
- **Display All Contacts:** Display the full list of contacts stored in the phonebook.
- **Delete Contact:** Remove a contact from the phonebook by specifying the name.
- **Update Contact:** Update the phone number of an existing contact by searching for the contact and providing a new phone number.

## 4. Modules Overview

The Simple Phonebook Application is divided into three main modules:

- Module 1: Phonebook GUI
- Module 2: Phonebook Management
- Module 3: Contact Management

## 5. Detailed Description of Modules

### 5.1. Module 1: Phonebook GUI

#### **Purpose:**

The Phonebook GUI module provides the graphical interface for users to interact with the system. It displays input fields, buttons, and an area for displaying results.

#### **Components:**

- **Name Input Field:** Text field where the user enters the contact's name.
- **Phone Number Input Field:** Text field where the user enters the contact's phone number.
- **Search Field:** Text field where the user specifies the contact name to search or delete.
- **Display Area:** Area where the results of actions (e.g., search results, error messages) are shown.

#### **Buttons:**

- **Insert Button:** To trigger the insert operation.
- **Search Button:** To trigger the search operation.
- **Display All Button:** To display all contacts.
- **Delete Button:** To trigger the delete operation.
- **Update Button:** To trigger the update operation.

#### **Description of Functions:**

- **InsertButtonListener:** Handles the action when the user clicks the "Insert" button. It retrieves the name and phone number from the input fields and passes them to the Phonebook module.
- **SearchButtonListener:** Handles the action when the "Search" button is clicked. It takes the name from the search field and queries the Phonebook module to retrieve the contact details.

- **DisplayButtonListener:** Handles the action when the "Display All" button is clicked. It displays all contacts stored in the phonebook.
- **DeleteButtonListener:** Handles the action when the "Delete" button is clicked. It takes the name from the search field and calls the Phonebook module to remove the contact.
- **UpdateButtonListener:** Handles the action when the "Update" button is clicked. It takes the name and new phone number from the fields and updates the contact via the Phonebook module.

## 5.2. Module 2: Phonebook Management

### **Purpose:**

This module manages the list of contacts. It handles operations such as adding, searching, deleting, and updating contacts. It acts as the backend for the application, providing the core functionality required to manipulate the contact data.

### **Components:**

- **Contacts List:** An internal data structure (e.g., an array list) that stores all the contacts.

### **Description of Functions:**

- **insertContact(name, phoneNumber):** Adds a new contact to the list. If the contact already exists, the new contact replaces the old one.
- **searchContact(name):** Searches for a contact in the list by name. Returns the contact details if found, or null if not.
- **displayAllContacts():** Retrieves all contacts and returns them as a formatted string for display.
- **deleteContact(name):** Removes a contact from the list by matching the name. Returns true if the contact was found and deleted, or false if not.
- **updateContact(name, newPhoneNumber):** Updates the phone number of an existing contact. Returns true if the update was successful or false if the contact was not found.

## 5.3. Module 3: Contact Management

### **Purpose:**

This module handles individual contact objects, which store the name and phone number for each contact in the phonebook.

### **Components:**

- **Name:** The name of the contact.
- **Phone Number:** The phone number of the contact.

### **Description of Functions:**

- Constructor (name, phoneNumber): Initializes a new contact with a specified name and phone number.
- getName(): Returns the contact's name.
- getPhoneNumber(): Returns the contact's phone number.
- setPhoneNumber(newPhoneNumber): Updates the contact's phone number.
- toString(): Formats the contact's details as a string for display purposes.

## **6. Flow of Operation**

### **Insert Contact:**

User enters name and phone number.

The Phonebook module stores the new contact in the contacts list.

The GUI shows confirmation that the contact was added.

### **Search Contact:**

User enters a name in the search field.

The Phonebook module searches for the contact by name.

If found, the contact's details are displayed in the GUI.

### **Display All Contacts:**

User clicks the "Display All" button.

The Phonebook module retrieves all stored contacts.

The GUI displays the list of contacts.

### **Delete Contact:**

User enters a name in the search field.

The Phonebook module deletes the contact with the matching name.

The GUI shows confirmation if the contact was deleted or an error message if the contact was not found.

### **Update Contact:**

User enters a name and a new phone number.

The Phonebook module searches for the contact and updates the phone number if the contact is found.

The GUI shows confirmation if the contact was updated or an error message if the contact was not found.

## 7. PSEUDOCODE

### 1. START Phonebook Application

CREATE empty list of contacts

SETUP user interface with:

- Input fields for name and phone number
- Buttons for: Insert, Search, Display, Delete, and Update
- Area to display results or messages

WAIT for user action (button clicks)

### 2. WHEN user clicks "Insert" button:

GET name and phone number from input fields

IF name and phone number are NOT empty:

CREATE a new contact with the provided name and phone number

ADD the new contact to the list of contacts

DISPLAY message: "Contact inserted: [name] - [phone number]"

ELSE:

DISPLAY error message: "Both fields must be filled"

CLEAR input field

### 3. WHEN user clicks "Search" button:

GET name from search field

IF name is NOT empty:

SEARCH the contact list for a contact with the provided name

IF contact is found:

DISPLAY the contact's name and phone number

ELSE:

DISPLAY message: "Contact not found"

ELSE:

DISPLAY error message: "Please enter a name to search"

CLEAR search field

#### 4. WHEN user clicks "Display All" button:

IF contact list is NOT empty:

DISPLAY all contacts in the list, showing their names and phone numbers

ELSE:

DISPLAY message: "No contacts found"

#### 5. WHEN user clicks "Delete" button:

GET name from search field

IF name is NOT empty:

SEARCH the contact list for a contact with the provided name

IF contact is found:

REMOVE the contact from the list

DISPLAY message: "Contact deleted: [name]"

ELSE:

DISPLAY message: "Contact not found"

ELSE:

DISPLAY error message: "Please enter a name to delete"

CLEAR search field

**6. WHEN user clicks "Update" button:**

GET name from search field

GET new phone number from update field

IF name and new phone number are NOT empty:

SEARCH the contact list for a contact with the provided name

IF contact is found:

UPDATE the contact's phone number to the new one

DISPLAY message: "Contact updated: [name] - [new phone number]"

ELSE:

DISPLAY message: "Contact not found"

ELSE:

DISPLAY error message: "Please fill in both fields"

CLEAR input fields

**7. CONTINUE to wait for user action (insert, search, display, delete, update)**

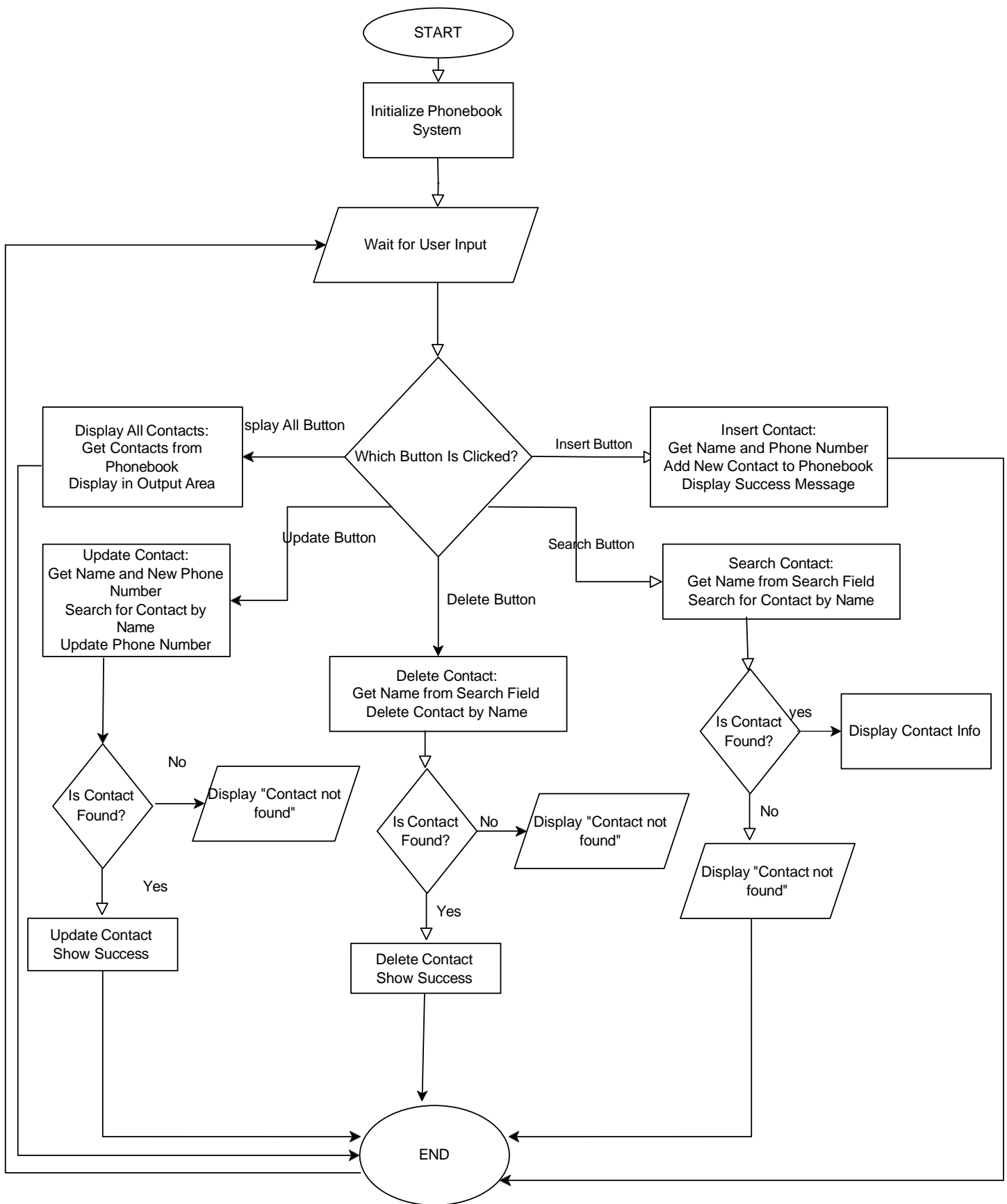
IF user closes the application:

EXIT the program

END



## 8. FLOWCHART



## SECTION B

### PROGRAM IMPLEMENTATION USING JAVA

This is a simple implementation of a GUI-based Phonebook application using Swing in Java. The system will use basic Swing components such as JFrame, JPanel, JTextField, JButton, and JTextArea to create a simple phonebook interface.

### SOURCE CODE

#### 1. PHONEBOOK GUI CLASS

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;

public class PhonebookGUI extends JFrame {
    private Phonebook phonebook;
    private JTextField nameField, phoneField, searchField, updateField;
    private JTextArea displayArea;

    public PhonebookGUI() {
        phonebook = new Phonebook();

        // Setting up the main frame
        setTitle("Phonebook Application");
        setSize(500, 500);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new BorderLayout());

        // Create components for the GUI
        nameField = new JTextField(15);
        phoneField = new JTextField(15);
        searchField = new JTextField(15);
        updateField = new JTextField(15);
        displayArea = new JTextArea(15, 40);
        displayArea.setEditable(false);

        // Adding buttons
        JButton insertButton = new JButton("Insert Contact");
        JButton searchButton = new JButton("Search Contact");
        JButton displayButton = new JButton("Display All Contacts");
        JButton deleteButton = new JButton("Delete Contact");
        JButton updateButton = new JButton("Update Contact");

        // Adding listeners to buttons
        insertButton.addActionListener(new InsertButtonListener());
        searchButton.addActionListener(new SearchButtonListener());
        displayButton.addActionListener(new DisplayButtonListener());
        deleteButton.addActionListener(new DeleteButtonListener());
    }
}
```

```

updateButton.addActionListener(new UpdateButtonListener());

// Creating a panel for adding contacts
JPanel inputPanel = new JPanel();
inputPanel.setLayout(new GridLayout(4, 2));
inputPanel.add(new JLabel("Name:"));
inputPanel.add(nameField);
inputPanel.add(new JLabel("Phone Number:"));
inputPanel.add(phoneField);
inputPanel.add(insertButton);
inputPanel.add(displayButton);

// Creating a panel for searching, deleting, and updating contacts
JPanel actionPanel = new JPanel();
actionPanel.setLayout(new GridLayout(3, 2));
actionPanel.add(new JLabel("Search/Delete Contact by Name:"));
actionPanel.add(searchField);
actionPanel.add(searchButton);
actionPanel.add(deleteButton);
actionPanel.add(new JLabel("Update Contact with New Phone Number:"));
actionPanel.add(updateField);
actionPanel.add(updateButton);

// Scroll pane for displaying contacts
JScrollPane scrollPane = new JScrollPane(displayArea);

// Adding panels and components to the frame
add(inputPanel, BorderLayout.NORTH);
add(actionPanel, BorderLayout.CENTER);
add(scrollPane, BorderLayout.SOUTH);
}

// Action listeners for buttons
private class InsertButtonListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        String name = nameField.getText();
        String phoneNumber = phoneField.getText();

        if (!name.isEmpty() && !phoneNumber.isEmpty()) {
            phonebook.insertContact(name, phoneNumber);
            displayArea.setText("Contact inserted: " + name + " - " + phoneNumber);
            nameField.setText("");
            phoneField.setText("");
        } else {
            displayArea.setText("Please fill in both fields.");
        }
    }
}

private class SearchButtonListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        String name = searchField.getText();
        Contact contact = phonebook.searchContact(name);
        if (contact != null) {
            displayArea.setText("Contact found: " + contact);
        }
    }
}

```

```

        } else {
            displayArea.setText("Contact not found.");
        }
    }
}

private class DisplayButtonListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        displayArea.setText("All Contacts:\n" + phonebook.displayAllContacts());
    }
}

private class DeleteButtonListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        String name = searchField.getText();
        if (phonebook.deleteContact(name)) {
            displayArea.setText("Contact deleted: " + name);
        } else {
            displayArea.setText("Contact not found.");
        }
    }
}

private class UpdateButtonListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        String name = searchField.getText();
        String newPhoneNumber = updateField.getText();

        if (phonebook.updateContact(name, newPhoneNumber)) {
            displayArea.setText("Contact updated: " + name + " - " + newPhoneNumber);
        } else {
            displayArea.setText("Contact not found.");
        }
    }
}

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

```

## 2. PHONEBOOK CLASS

```

import java.util.ArrayList;
import java.util.List;

public class Phonebook {
    private ArrayList<Contact> contacts;

    public Phonebook() {
        contacts = new ArrayList<>();
    }
}

```

```

}

// 1. Insert Contact
public void insertContact(String name, String phoneNumber) {
    Contact newContact = new Contact(name, phoneNumber);
    contacts.add(newContact);
}

// 2. Search Contact
public Contact searchContact(String name) {
    for (Contact contact : contacts) {
        if (contact.getName().equalsIgnoreCase(name)) {
            return contact;
        }
    }
    return null;
}

// 3. Display All Contacts
public String displayAllContacts() {
    if (contacts.isEmpty()) {
        return "No contacts found.";
    } else {
        StringBuilder allContacts = new StringBuilder();
        for (Contact contact : contacts) {
            allContacts.append(contact).append("\n");
        }
        return allContacts.toString();
    }
}

// 4. Delete Contact
public boolean deleteContact(String name) {
    Contact contact = searchContact(name);
    if (contact != null) {
        contacts.remove(contact);
        return true;
    }
    return false;
}

// 5. Update Contact
public boolean updateContact(String name, String newPhoneNumber) {
    Contact contact = searchContact(name);
    if (contact != null) {
        contact.setPhoneNumber(newPhoneNumber);
        return true;
    }
    return false;
}
}

```

### 3. CONTACT CLASS

```
public class Contact {
    private String name;
    private String phoneNumber;

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

    public String getName() {
        return name;
    }

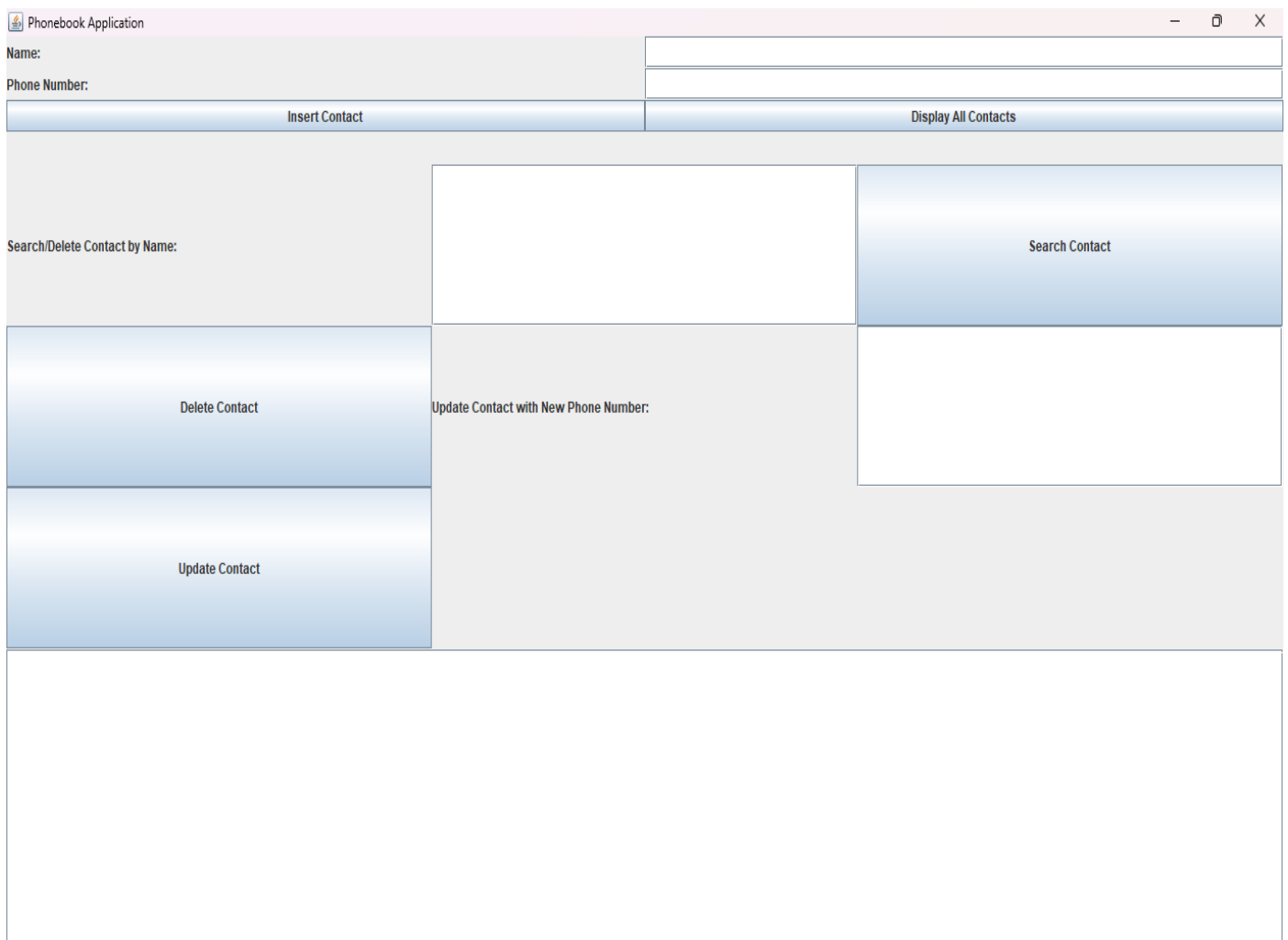
    public String getPhoneNumber() {
        return phoneNumber;
    }

    public void setPhoneNumber(String phoneNumber) {
        this.phoneNumber = phoneNumber;
    }

    @Override
    public String toString() {
        return "Name: " + name + ", Phone: " + phoneNumber;
    }
}
```

## 2. SCREENSHOTS OF GRAPHICAL USER INTERFACE

### 1. Home Screen



The screenshot displays the 'Phonebook Application' window. At the top, there are input fields for 'Name:' and 'Phone Number:'. Below these are two buttons: 'Insert Contact' and 'Display All Contacts'. A section labeled 'Search/Delete Contact by Name:' contains a text input field and a 'Search Contact' button. Further down, there are buttons for 'Delete Contact' and 'Update Contact', alongside a label 'Update Contact with New Phone Number:' and another text input field. The bottom half of the window is a large, empty rectangular area.

Phonebook Application		— □ ×
Name:	<input type="text"/>	
Phone Number:	<input type="text"/>	
Insert Contact		Display All Contacts
Search/Delete Contact by Name:	<input type="text"/>	Search Contact
Delete Contact	Update Contact with New Phone Number:	<input type="text"/>
Update Contact		
<div></div>		

## 2. Contacts Displayed

Phonebook Application

Name:

Phone Number:

Insert Contact

Display All Contacts

Search/Delete Contact by Name:

Search Contact

Delete Contact

Update Contact with New Phone Number:

Update Contact

All Contacts:

Name: John, Phone: 081456789  
Name: Garry, Phone: 081326547  
Name: Mike, Phone: 081457896  
Name: Jessie, Phone: 081698732  
Name: Helena, Phone: 081752368



