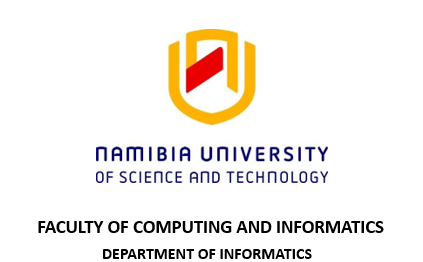
****

**PHONEBOOK APPLICATION PROJECT**

**06/10/2024**

**STUDENT NO: NAME:**

**223090204 NAKANDANGWA DIMSON**

**223009818 KAPEMBE PETRUS**

**224071882 HERMAN KONSTATINUS**

**224019589 LAZARUS THOMAS**

**222075287 MATHEUS NANGOLO**

**224067826 ANDREAS N RISTO**

**1.INTRODUCTION**

In today's fast-paced world, effective communication is essential for both personal and professional success. As our social networks grow, so does the need to manage contact information efficiently. The Phonebook Application is designed to address this need by providing a user-friendly platform for organizing and accessing contact details seamlessly. This application allows users to store vital information, such as names and phone numbers, and offers intuitive features for inserting, updating, deleting, and searching for contacts. The straightforward interface ensures that users of all ages can navigate the application with ease, making it a valuable tool for anyone looking to simplify their contact management process.

**2.FUNCTIONAL REQUIREMENTS**

1. Insert Contact

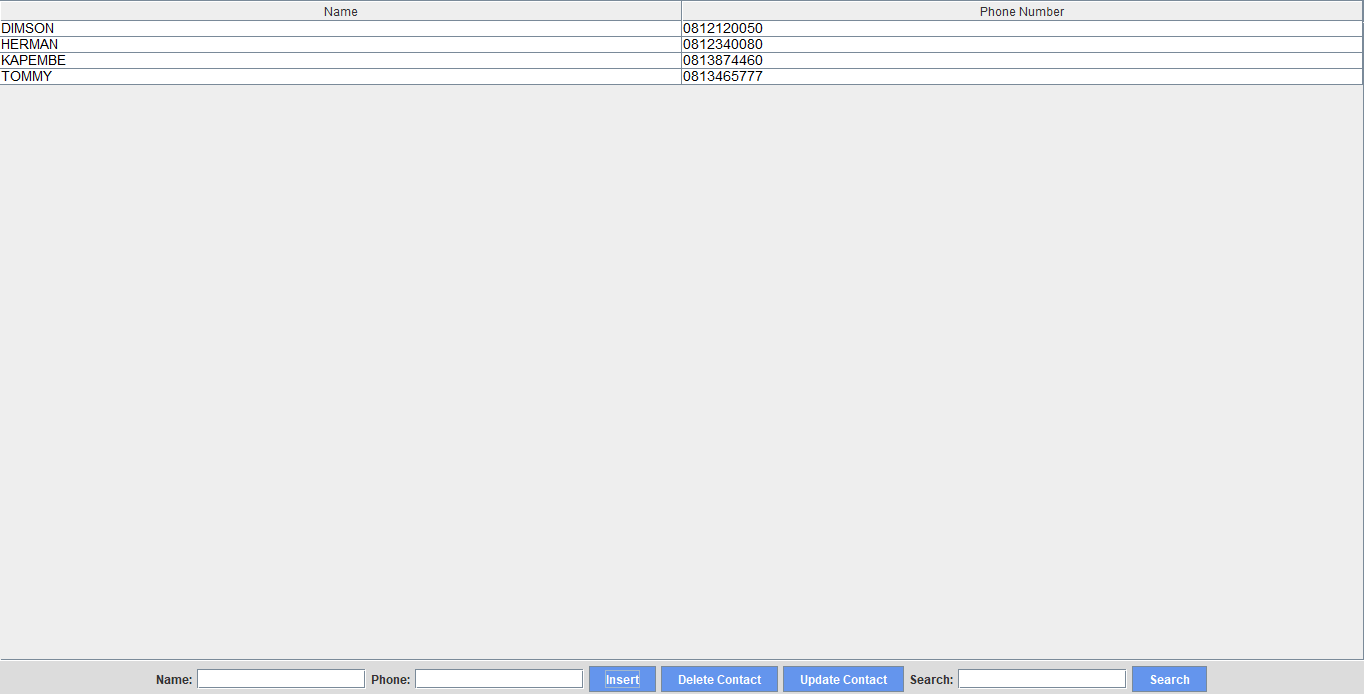
2. Update Contact

3. Delete Contact (with confirmation)

4. Search Contact

5. Sort Contacts

**4. User Interface Design**

****

**PROJECT (CODES)**

package com.mycompany.phonebook1;

import javax.swing.\*;

import javax.swing.table.DefaultTableModel;

import java.awt.\*;

public class Phonebook1 extends JFrame {

private JTable table;

private DefaultTableModel model;

private JTextField nameField;

private JTextField phoneField;

private JTextField searchField;

public Phonebook1() {

setTitle("Phonebook");

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setSize(600, 400);

setLocationRelativeTo(null);

// Set a background color for the main frame

getContentPane().setBackground(new Color(220, 220, 220)); // Light grey background

// Create table model and table

model = new DefaultTableModel(new String[]{"Name", "Phone Number"}, 0);

table = new JTable(model);

table.setBackground(Color.WHITE); // Table background color

table.setForeground(Color.BLACK); // Table text color

table.setFont(new Font("Arial", Font.PLAIN, 14)); // Table font

// Enable sorting by column

table.setAutoCreateRowSorter(true); // Enable sorting by column

JScrollPane scrollPane = new JScrollPane(table);

// Create input fields

nameField = new JTextField(15);

phoneField = new JTextField(15);

searchField = new JTextField(15); // Search field

// Create buttons

JButton insertButton = new JButton("Insert");

JButton deleteButton = new JButton("Delete Contact");

JButton updateButton = new JButton("Update Contact");

JButton searchButton = new JButton("Search");

// Set button colors

Color buttonColor = new Color(100, 149, 237); // Cornflower blue

Color textColor = Color.WHITE; // White text

insertButton.setBackground(buttonColor);

insertButton.setForeground(textColor);

deleteButton.setBackground(buttonColor);

deleteButton.setForeground(textColor);

updateButton.setBackground(buttonColor);

updateButton.setForeground(textColor);

searchButton.setBackground(buttonColor);

searchButton.setForeground(textColor);

// Add action listeners

insertButton.addActionListener(e -> insertContact());

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

updateButton.addActionListener(e -> updateContact());

searchButton.addActionListener(e -> searchContact());

// Create a panel for input fields and buttons

JPanel panel = new JPanel();

panel.setBackground(new Color(220, 220, 220)); // Same background color as the frame

panel.add(new JLabel("Name:"));

panel.add(nameField);

panel.add(new JLabel("Phone:"));

panel.add(phoneField);

panel.add(insertButton);

panel.add(deleteButton);

panel.add(updateButton);

panel.add(new JLabel("Search:"));

panel.add(searchField);

panel.add(searchButton);

// Add components to frame

add(scrollPane, BorderLayout.CENTER);

add(panel, BorderLayout.SOUTH);

}

private void insertContact() {

String name = nameField.getText();

String phone = phoneField.getText();

if (!name.isEmpty() && !phone.isEmpty()) {

if (!isPhoneUnique(phone)) {

JOptionPane.showMessageDialog(this, "Phone number already exists.");

} else {

model.addRow(new Object[]{name, phone});

nameField.setText("");

phoneField.setText("");

}

} else {

JOptionPane.showMessageDialog(this, "Please enter both name and phone number.");

}

}

private void deleteContact() {

int selectedRow = table.getSelectedRow();

if (selectedRow != -1) {

int confirm = JOptionPane.showConfirmDialog(

this,

"Are you sure you want to delete this contact?",

"Confirm Delete",

JOptionPane.YES\_NO\_OPTION,

JOptionPane.QUESTION\_MESSAGE

);

if (confirm == JOptionPane.YES\_OPTION) {

model.removeRow(selectedRow);

}

} else {

JOptionPane.showMessageDialog(this, "Please select a contact to delete.");

}

}

private void updateContact() {

int selectedRow = table.getSelectedRow();

if (selectedRow != -1) {

String name = nameField.getText();

String phone = phoneField.getText();

if (!name.isEmpty() && !phone.isEmpty()) {

if (!isPhoneUnique(phone) && !phone.equals(model.getValueAt(selectedRow, 1))) {

JOptionPane.showMessageDialog(this, "Phone number already exists.");

} else {

model.setValueAt(name, selectedRow, 0);

model.setValueAt(phone, selectedRow, 1);

nameField.setText("");

phoneField.setText("");

}

} else {

JOptionPane.showMessageDialog(this, "Please enter both name and phone number.");

}

} else {

JOptionPane.showMessageDialog(this, "Please select a contact to update.");

}

}

private void searchContact() {

String searchName = searchField.getText();

if (!searchName.isEmpty()) {

boolean found = false;

for (int i = 0; i < model.getRowCount(); i++) {

String name = (String) model.getValueAt(i, 0);

if (name.equalsIgnoreCase(searchName)) {

table.setRowSelectionInterval(i, i);

found = true;

break;

}

}

if (!found) {

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

}

} else {

JOptionPane.showMessageDialog(this, "Please enter a name to search.");

}

}

private boolean isPhoneUnique(String phone) {

for (int i = 0; i < model.getRowCount(); i++) {

String existingPhone = (String) model.getValueAt(i, 1);

if (existingPhone.equals(phone)) {

return false;

}

}

return true;

}

public static void main(String[] args) {

SwingUtilities.invokeLater(() -> {

Phonebook1 phonebook1 = new Phonebook1();

phonebook1.setVisible(true);

});

}

}

**5. Algorithms Representation**

**1. Main Module**

**Pseudocode**

FUNCTION main()

CREATE an instance of Phonebook1

SET Phonebook1 visibility to true

END FUNCTION

**2. Phonebook1 Class**

**a. Constructor**

**Pseudocode:**

FUNCTION Phonebook1()

SET title to "Phonebook"

SET default close operation to EXIT\_ON\_CLOSE

SET size to 600x400

CENTER the window on the screen

SET background color of the frame to light grey

INITIALIZE table model with columns "Name" and "Phone Number"

CREATE a table with the model

ENABLE sorting on the table

CREATE input fields for name, phone number, and search

CREATE buttons for Insert, Delete, Update, and Search

SET button colors

ADD action listeners for buttons

- Add: Call insertContact()

- Delete: Call deleteContact()

- Edit: Call updateContact()

- Search: Call searchContact()

CREATE a panel for input fields and buttons

ADD components to the frame

END FUNCTION

**b. Insert Contact Function**

**Pseudocode:**

FUNCTION insertContact()

GET name from nameField

GET phone from phoneField

IF name is not empty AND phone is not empty THEN

ADD a new row to model with name and phone

CLEAR nameField and phoneField

ELSE

SHOW message "Please enter both name and phone number."

END IF

END FUNCTION

**c. Delete Contact Function**

**Pseudocode:**

FUNCTION deleteContact()

GET selected row index from table

IF selected row index is not -1 THEN

SHOW confirmation dialog "Are you sure you want to delete this contact?"

IF user confirms THEN

REMOVE the selected row from model

END IF

ELSE

SHOW message "Please select a contact to delete."

END IF

END FUNCTION

**d. Update Contact Function**

**Pseudocode:**

FUNCTION updateContact()

GET selected row index from table

IF selected row index is not -1 THEN

GET name from nameField

GET phone from phoneField

IF name is not empty AND phone is not empty THEN

SET model value at selected row, column 0 to name

SET model value at selected row, column 1 to phone

CLEAR nameField and phoneField

ELSE

SHOW message "Please enter both name and phone number."

END IF

ELSE

SHOW message "Please select a contact to update."

END IF

END FUNCTION

**e. Search Contact Function**

**Pseudocode:**

FUNCTION searchContact()

GET search name from searchField

IF search name is not empty THEN

SET found to false

FOR each row in model DO

GET name from model at current row

IF name matches search name (case-insensitive) THEN

SELECT the current row in table

SET found to true

BREAK the loop

END IF

END FOR

IF found is false THEN

SHOW message "Contact not found."

END IF

ELSE

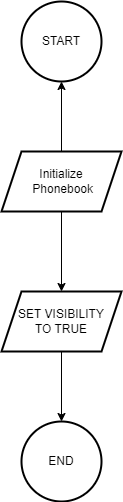
SHOW message "Please enter a name to search."

END IF

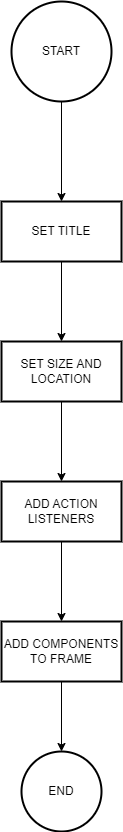
END FUNCTION

**Flowchart Representation**

**Main Module Flowchart:**



**Phonebook1 Constructor Flowchart:**

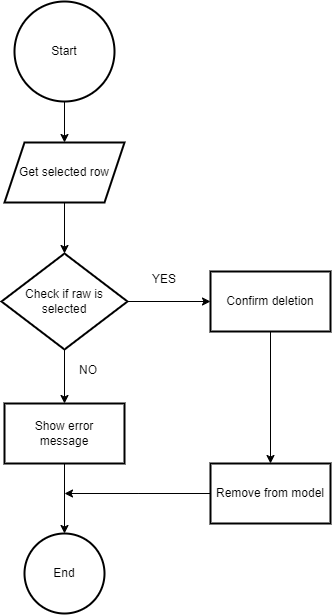
****

**Insert Contact Flowchart:**

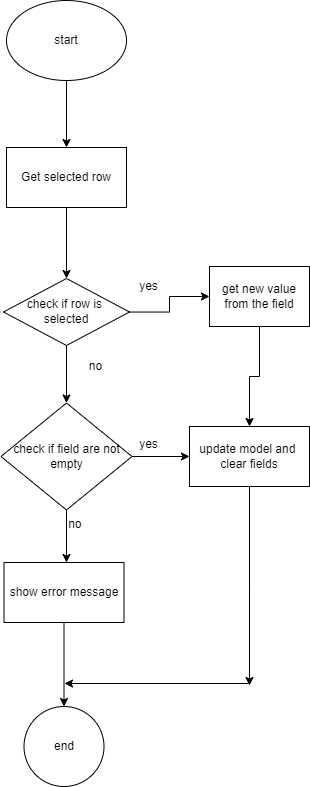
A diagram of a flowchart

Description automatically generated

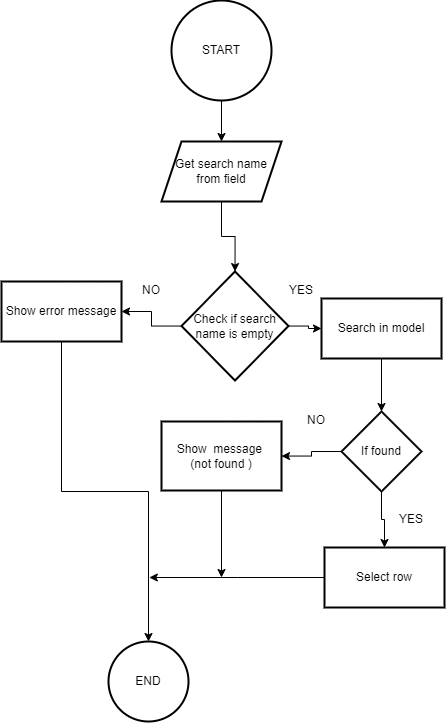
**Delete Contact Flowchart:**



**Update Contact Flowchart:**



**Search Contact Flowchart:**



**Testing and Validation**

The Phonebook application was tested for key functionalities, including inserting, deleting, updating, and searching contacts. The add function ensures that both name and phone number fields are filled before adding a contact, with validation prompts for incomplete inputs. The delete function requires user confirmation before removing a contact, and updating allows updates to selected entries, ensuring both fields are properly filled. The search function accurately highlights matching contacts, and if no match is found, an appropriate message is displayed. Overall, the tests confirmed that the application performs reliably in managing contacts.

**Conclusion**

In conclusion, the Phonebook application successfully meets its primary goal of providing an intuitive and efficient way to manage contact information. Through user-friendly features such as inserting, updating, deleting, and searching for contacts, the program simplifies personal or professional contact management. The inclusion of validation mechanisms ensures data integrity, while the confirmation prompt for deletion prevents accidental loss of information. Overall, this project demonstrates effective use of Java and Swing for building functional desktop applications, with potential for future enhancements such as sorting and importing/exporting contact lists.