NAMIBIA UNIVERSITY OF SCIENCE AND TECHNOLOGY

Bachelor of computer science and informatics

Name: Haitula melkisedek	223011878
Stefanus Daniel I N	222103426
NANGOLO DROTHEA G	223039985
SHIKONGO HELVI	224081659
HAIMBODI ELIASER ND	223067261
ALFEUS ROSALIA	224009893

```
import java.awt.*;

import java.awt.event.*;

import javax.sound.sampled.*;

import javax.swing.*;

import javax.swing.event.*;

import javax.swing.table.DefaultTableModel;

import javax.swing.table.TableRowSorter;

import java.io.*;

import java.net.URL;

import java.util.ArrayList;

import java.util.List;


class Contact implements Serializable {

  private static final long serialVersionUID = 1L;

  private String category;

  private String name;

  private String phone;


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

    this.category = category;

    this.name = name;

    this.phone = phone;

  }
```

```
    public String getCategory() {
        return category;
    }

    public String getName() {
        return name;
    }

    public String getPhone() {
        return phone;
    }
}

public class Phonebook implements ActionListener {
    JFrame frame;
    JTextField nameField, phoneField, searchInputField;
    JTable contactTable;
    DefaultTableModel contactTableModel;
    TableRowSorter<DefaultTableModel> sorter;
    JButton addButton, viewButton, searchButton, deleteButton, updateButton;
    JComboBox<String> categoryOptions;
    JPanel inputPanel, outputPanel, keyboardPanel, buttonPanel;
    boolean isSearchingByName;

    private static final String CONTACTS_FILE = "contacts.ser";
    private static final String SOUND_FILE = "sound.wav"; // Ensure the sound file is in the
resources folder

    public Phonebook() {
        // Create JFrame
        frame = new JFrame("Contact Book");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.getContentPane().setBackground(Color.GREEN); // Change to green

        // Create text fields and buttons
        nameField = new JTextField();
        phoneField = new JTextField();
        searchInputField = new JTextField();

        addButton = new JButton("Add Contact");
        viewButton = new JButton("View Contacts");
        searchButton = new JButton("Search");
        deleteButton = new JButton("Delete Selected Contacts");
        updateButton = new JButton("Update Contact");

        // Set button background color
        addButton.setBackground(Color.LIGHT_GRAY);
        viewButton.setBackground(Color.LIGHT_GRAY);
        searchButton.setBackground(Color.LIGHT_GRAY);
        deleteButton.setBackground(Color.LIGHT_GRAY);
        updateButton.setBackground(Color.LIGHT_GRAY);
        // Add tooltips
        addButton.setToolTipText("Add a new contact to the phonebook.");
        viewButton.setToolTipText("View all contacts in the phonebook.");
        searchButton.setToolTipText("Search for contacts by name or phone number.");
        deleteButton.setToolTipText("Delete selected contacts from the phonebook.");
        updateButton.setToolTipText("Update the selected contact's details.");

        // Create panels
        inputPanel = new JPanel(new GridLayout(6, 2));
        outputPanel = new JPanel(new BorderLayout());
        keyboardPanel = new JPanel(new GridLayout(3, 9));
        buttonPanel = new JPanel(new GridLayout(5, 1));

        // Set the background color of input, output, and button panels to green
        inputPanel.setBackground(Color.GREEN);
        outputPanel.setBackground(Color.GREEN);
        buttonPanel.setBackground(Color.GREEN);
        keyboardPanel.setBackground(Color.GREEN);

        // Initialize the contact table
        contactTableModel = new DefaultTableModel(new Object[]{"Select", "Category",
"Name", "Phone"}, 0) {
            Class[] types = new Class[]{Boolean.class, String.class, String.class, String.class};

            @Override
            public Class<?> getColumnClass(int columnIndex) {
                return types[columnIndex];
            }
        };
        contactTable = new JTable(contactTableModel);
        contactTable.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);

        // Create a TableRowSorter and set it to the contactTable
        sorter = new TableRowSorter<>(contactTableModel);
        contactTable.setRowSorter(sorter);

        // Initialize JScrollPane for the contact table
        JScrollPane scrollPane = new JScrollPane(contactTable);

scrollPane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED);
        scrollPane.setPreferredSize(new Dimension(600, 400));

        // Add components to input panel
        inputPanel.add(new JLabel("Name:"));
        inputPanel.add(nameField);
        inputPanel.add(new JLabel("Phone:"));
        inputPanel.add(phoneField);
        inputPanel.add(new JLabel("Category:"));

        // Category selection options
        categoryOptions = new JComboBox<>(new String[]{"Friends", "Family", "Work"});
```

```
inputPanel.add(categoryOptions);

inputPanel.add(new JLabel("Search:"));
inputPanel.add(searchInputField);

// Add action listeners for buttons
addButton.addActionListener(this);
viewButton.addActionListener(this);
searchButton.addActionListener(this);
deleteButton.addActionListener(this);
updateButton.addActionListener(this);

// Add buttons to buttonPanel vertically
buttonPanel.add(addButton);
buttonPanel.add(searchButton);
buttonPanel.add(deleteButton);
buttonPanel.add(updateButton);
buttonPanel.add(viewButton);

// Create keyboard buttons
createKeyboardButtons();

// Add panels to frame
frame.setLayout(new BorderLayout());
frame.add(inputPanel, BorderLayout.NORTH);
frame.add(buttonPanel, BorderLayout.WEST);
```

```
    frame.add(outputPanel, BorderLayout.CENTER);

    frame.add(keyboardPanel, BorderLayout.SOUTH);

    outputPanel.add(scrollPane, BorderLayout.CENTER);


    // Load contacts from file when the application starts

    loadContacts();


    // Frame settings

    frame.setSize(600, 600);

    frame.setVisible(true);


    // Save contacts when the application is closing

    frame.addWindowListener(new WindowAdapter() {

        public void windowClosing(WindowEvent e) {

            saveContacts();

            System.exit(0);

        }

    });


    // Add keyboard shortcuts

    addKeyboardShortcuts();

}


// Create keyboard buttons A-Z

private void createKeyboardButtons() {

    for (char letter = 'A'; letter <= 'Z'; letter++) {
        JButton button = new JButton(String.valueOf(letter));
        button.addActionListener(new ActionListener() {
          @Override
          public void actionPerformed(ActionEvent e) {
            nameField.setText(nameField.getText() + button.getText());
          }
        });
        keyboardPanel.add(button);
      }
    }

    public static void main(String[] args) {
      new Phonebook();
    }

    public void actionPerformed(ActionEvent e) {
      if (e.getSource() == addButton) {
        addContact();
      } else if (e.getSource() == viewButton) {
        viewContacts();
      } else if (e.getSource() == searchButton) {
        searchContacts();
      } else if (e.getSource() == deleteButton) {
        deleteContacts();
      } else if (e.getSource() == updateButton) {
        updateContact();
        }
    }

    private void viewContacts() {
        // Create a JDialog to display the contacts
        JDialog viewDialog = new JDialog(frame, "View Contacts", true);
        viewDialog.setLayout(new BorderLayout());

        // Create a new JTable to show contacts
        JTable viewContactsTable = new JTable(contactTableModel); // Use the same model
        viewContactsTable.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);

        // Create a JScrollPane for the JTable
        JScrollPane scrollPane = new JScrollPane(viewContactsTable);

scrollPane.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED);

        // Add scrollPane to the dialog
        viewDialog.add(scrollPane, BorderLayout.CENTER);

        // Add button to close the dialog
        JButton closeButton = new JButton("Close");
        closeButton.addActionListener(e -> viewDialog.dispose());
        viewDialog.add(closeButton, BorderLayout.SOUTH);

        // Set dialog size and make it visible
        viewDialog.setSize(400, 300); // Adjust size as needed
        viewDialog.setLocationRelativeTo(frame);
        viewDialog.setVisible(true);
    }

    private void addKeyboardShortcuts() {
        // Add keyboard shortcuts
        addButton.setMnemonic(KeyEvent.VK_A); // Alt + A
        deleteButton.setMnemonic(KeyEvent.VK_D); // Alt + D
        updateButton.setMnemonic(KeyEvent.VK_U); // Alt + U
        viewButton.setMnemonic(KeyEvent.VK_V); // Alt + V
        searchButton.setMnemonic(KeyEvent.VK_S); // Alt + S

        // Enter key to add a contact
        nameField.getInputMap().put(KeyStroke.getKeyStroke(KeyEvent.VK_ENTER, 0),
"addContact");
        nameField.getActionMap().put("addContact", new AbstractAction() {
            @Override
            public void actionPerformed(ActionEvent e) {
                addContact();
            }
        });

        // Delete key to remove selected contacts
        contactTable.getInputMap().put(KeyStroke.getKeyStroke(KeyEvent.VK_DELETE, 0),
"deleteContact");
```

```
        contactTable.getActionMap().put("deleteContact", new AbstractAction() {
            @Override
            public void actionPerformed(ActionEvent e) {
                deleteContacts();
            }
        });

        // Escape key to exit the application

frame.getRootPane().getInputMap().put(KeyStroke.getKeyStroke(KeyEvent.VK_ESCAPE, 0),
"closeApp");
        frame.getRootPane().getActionMap().put("closeApp", new AbstractAction() {
            @Override
            public void actionPerformed(ActionEvent e) {
                saveContacts();
                System.exit(0);
            }
        });
    }

    private void addContact() {
        String name = nameField.getText().trim();
        String phone = phoneField.getText();
        String category = categoryOptions.getSelectedItem().toString();

        // Validate name
        if (!validateName(name)) {
            JOptionPane.showMessageDialog(frame, "Please enter a valid name (1-50
characters, no digits).");
            return;
        }

        if (validatePhoneNumber(phone)) {
            if (!name.isEmpty()) {
                contactTableModel.addRow(new Object[]{false, category, name, phone}); // false
for checkbox
                nameField.setText("");
                phoneField.setText("");
                saveContacts(); // Save contacts after adding
                playSound(); // Play sound on addition of contact
            } else {
                JOptionPane.showMessageDialog(frame, "Please enter a name.");
            }
        } else {
            JOptionPane.showMessageDialog(frame, "Please enter a valid phone number (only
digits, 1 to 10 characters).");
        }
    }

    private void searchContacts() {
        String[] options = {"Search by Name", "Search by Phone Number"};
        int choice = JOptionPane.showOptionDialog(frame, "Choose search option:",
            "Search Options", JOptionPane.DEFAULT_OPTION,
JOptionPane.QUESTION_MESSAGE,
            null, options, options[0]);

        String searchTitle;
        String initialFieldValue = "";
        if (choice == 0) {
            searchTitle = "Search by Name";
            isSearchingByName = true; // Set searching by name
        } else {
            searchTitle = "Search by Phone Number";
            isSearchingByName = false; // Set searching by phone number
        }

        // Create a new dialog for user input
        String searchQuery = JOptionPane.showInputDialog(frame, searchTitle + ":",
initialFieldValue);
        if (searchQuery != null) {
            searchInputField.setText(searchQuery); // Set search query text field
            updateSearchResults(); // Update results based on the search input
        }
    }

    private void deleteContacts() {
        // Collect selected rows for deletion
        ArrayList<Integer> rowsToDelete = new ArrayList<>();
        for (int i = 0; i < contactTableModel.getRowCount(); i++) {
            Boolean isSelected = (Boolean) contactTableModel.getValueAt(i, 0);
            if (isSelected != null && isSelected) {
            rowsToDelete.add(i);
            }
        }

        if (!rowsToDelete.isEmpty()) {
            for (int i = rowsToDelete.size() - 1; i >= 0; i--) { // Remove in reverse order
            contactTableModel.removeRow(rowsToDelete.get(i));
            }
            JOptionPane.showMessageDialog(frame, "Selected contacts deleted successfully.");
            saveContacts(); // Save contacts after deletion
            playSound(); // Play sound on deletion of contacts
        } else {
            JOptionPane.showMessageDialog(frame, "Please select contacts to delete.");
        }
    }

    private void updateContact() {
        int selectedRow = contactTable.getSelectedRow();
        if (selectedRow != -1) {
            String currentCategory = contactTableModel.getValueAt(selectedRow, 1).toString();
            String currentName = contactTableModel.getValueAt(selectedRow, 2).toString();
            String currentPhone = contactTableModel.getValueAt(selectedRow, 3).toString();
            // Show dialogs for updating
            String newName = JOptionPane.showInputDialog(frame, "Edit Name:",
    currentName);
            String newPhone = JOptionPane.showInputDialog(frame, "Edit Phone:",
    currentPhone);
            String newCategory = (String) JOptionPane.showInputDialog(frame,
                "Edit Category:", "Category",
                JOptionPane.QUESTION_MESSAGE,
                null,
                new String[]{"Friends", "Family", "Work"},
                currentCategory);

            // Validate new name
            if (newName != null && !validateName(newName.trim())) {
            JOptionPane.showMessageDialog(frame, "Please enter a valid name (1-50
    characters, no digits).");
            return;
            }

            if (newPhone != null && validatePhoneNumber(newPhone)) {
            String updatedName = (newName != null && !newName.isEmpty()) ? newName :
    currentName;
            String updatedPhone = (newPhone != null && !newPhone.isEmpty()) ? newPhone :
    currentPhone;
            String updatedCategory = (newCategory != null) ? newCategory : currentCategory;

            contactTableModel.setValueAt(updatedCategory, selectedRow, 1);
```

```
            contactTableModel.setValueAt(updatedName, selectedRow, 2);
            contactTableModel.setValueAt(updatedPhone, selectedRow, 3);
            JOptionPane.showMessageDialog(frame, "Contact updated successfully.");
            saveContacts(); // Save contacts after updating
            playSound(); // Play sound on updating of contact
          } else {
            JOptionPane.showMessageDialog(frame, "Please enter a valid phone number (only
digits, 1 to 10 characters).");
          }
        } else {
          JOptionPane.showMessageDialog(frame, "Please select a contact to update.");
        }
    }

    private void updateSearchResults() {
        String searchQuery = searchInputField.getText().trim().toLowerCase();
        ArrayList<Object[]> results = new ArrayList<>();

        for (int i = 0; i < contactTableModel.getRowCount(); i++) {
          String contactName = contactTableModel.getValueAt(i, 2).toString().toLowerCase();
          String contactPhone = contactTableModel.getValueAt(i, 3).toString();

          if (isSearchingByName) { // Search by Name
            if (contactName.startsWith(searchQuery)) {
              results.add(new Object[]{
                    false,
                    contactTableModel.getValueAt(i, 1),
                    contactTableModel.getValueAt(i, 2),
                    contactTableModel.getValueAt(i, 3)
              });
            }
          } else { // Search by Phone Number
            if (contactPhone.startsWith(searchQuery)) {
              results.add(new Object[]{
                    false,
                    contactTableModel.getValueAt(i, 1),
                    contactTableModel.getValueAt(i, 2),
                    contactTableModel.getValueAt(i, 3)
              });
            }
          }
        }

        // Clear the table and repopulate with filtered results
        contactTableModel.setRowCount(0);
        for (Object[] contact : results) {
          contactTableModel.addRow(contact);
        }

        // Scroll to the top of the table if needed after search
        if (!results.isEmpty()) {
          contactTable.scrollRectToVisible(contactTable.getCellRect(0, 0, true));
        } else {
          contactTableModel.addRow(new Object[]{false, "No contacts found", "", ""});
        }
      }

      private boolean validatePhoneNumber(String phone) {
        return phone.matches("\\d{1,10}"); // Must be digits only and between 1 to 10
    characters long
      }

      private boolean validateName(String name) {
        return name.matches("^[a-zA-Z\\s]{1,50}$"); // Only letters and spaces, 1-50
    characters
      }

      private void saveContacts() {
        try (ObjectOutputStream oos = new ObjectOutputStream(new
    FileOutputStream(CONTACTS_FILE))) {
          List<Contact> contacts = getContacts();
          oos.writeObject(contacts);
        } catch (FileNotFoundException e) {
          JOptionPane.showMessageDialog(frame, "Contacts file not found. Creating a new
    file.");
        } catch (IOException e) {
          JOptionPane.showMessageDialog(frame, "Error saving contacts: " + e.getMessage());
        }
      }
      @SuppressWarnings("unchecked")
      private void loadContacts() {
        try (ObjectInputStream ois = new ObjectInputStream(new
    FileInputStream(CONTACTS_FILE))) {
          List<Contact> contacts = (List<Contact>) ois.readObject();
          for (Contact contact : contacts) {
            contactTableModel.addRow(new Object[]{false, contact.getCategory(),
    contact.getName(), contact.getPhone()});
          }
        } catch (FileNotFoundException e) {
          // File does not exist; it's okay, we just start with an empty phonebook.
          JOptionPane.showMessageDialog(frame, "No existing contacts found. Starting with
    an empty phonebook.");
        } catch (IOException e) {
          JOptionPane.showMessageDialog(frame, "Error loading contacts: " +
    e.getMessage());
        } catch (ClassNotFoundException e) {
          JOptionPane.showMessageDialog(frame, "Error in contact data: " + e.getMessage());
        }
      }

      private List<Contact> getContacts() {
        List<Contact> contacts = new ArrayList<>();
        for (int i = 0; i < contactTableModel.getRowCount(); i++) {
          String category = contactTableModel.getValueAt(i, 1).toString();
          String name = contactTableModel.getValueAt(i, 2).toString();
```

```
            String phone = contactTableModel.getValueAt(i, 3).toString();
            contacts.add(new Contact(category, name, phone));
        }
        return contacts;
    }

    private void playSound() {
        try {
            // Use class loader to get the sound file from resources
            URL soundURL = getClass().getClassLoader().getResource(SOUND_FILE);
            if (soundURL == null) {
                JOptionPane.showMessageDialog(frame, "Sound file not found: " + SOUND_FILE);
                return;
            }

            // Load the audio input stream
            AudioInputStream audioInputStream =
AudioSystem.getAudioInputStream(soundURL);
            Clip clip = AudioSystem.getClip();
            clip.open(audioInputStream);
            clip.start();
        } catch (UnsupportedAudioFileException | IOException | LineUnavailableException e) {
            JOptionPane.showMessageDialog(frame, "Error playing sound: " + e.getMessage());
        }
    }
}
```

Pseudocode

```
START Phonebook Application

  CREATE JFrame (main window)
  SET background color to green

  CREATE text fields (nameField, phoneField, searchInputField)
  CREATE buttons (addButton, viewButton, searchButton, deleteButton, updateButton)
  SET button tooltips and background colors

  CREATE JComboBox for category options (Friends, Family, Work)

  CREATE JPanel layout for input fields, buttons, keyboard, and output
  SET background colors of panels to green

  CREATE JTable for displaying contacts with columns: [Select, Category, Name, Phone]
  ADD TableRowSorter for filtering contacts

  ADD components to panels (input fields, buttons, table, keyboard)
  ADD panels to the JFrame

  ON application start:
    CALL loadContacts to load contacts from the file

  ON window close:
    CALL saveContacts to save contacts to the file
```

```
CREATE keyboard buttons for A-Z
  WHEN button clicked:
    APPEND corresponding letter to nameField

SET keyboard shortcuts for buttons:
  Alt + A for add contact
  Alt + D for delete contact
  Alt + U for update contact
  Alt + V for view contacts
  Alt + S for search contact
  Enter to add contact when nameField is focused
  Delete to remove selected contacts when JTable is focused
  Escape to close application

FUNCTION addContact():
  GET name, phone, and category from input fields
  IF name is valid AND phone is valid:
    ADD contact to the JTable
    CLEAR input fields
    CALL saveContacts to save contacts to the file
    PLAY success sound
  ELSE:
    SHOW error message

FUNCTION viewContacts():
  CREATE JDialog to display all contacts in a JTable
  ADD close button to close the dialog

FUNCTION searchContacts():
  ASK user to choose between "Search by Name" or "Search by Phone Number"
  GET search query from the user
  FILTER contacts based on search query and display in the table

FUNCTION deleteContacts():
  FOR each contact in the JTable:
    IF selected for deletion:
        REMOVE from the JTable
  CALL saveContacts to save the updated contacts
  PLAY success sound

FUNCTION updateContact():
  IF contact is selected in the JTable:
    GET current contact details
    ASK user to edit details (name, phone, category)
    IF new details are valid:
        UPDATE the contact in the JTable
        CALL saveContacts to save updated contacts
        PLAY success sound
  ELSE:
    SHOW error message
```

```
 FUNCTION saveContacts():
   SAVE all contacts from JTable to a file (contacts.ser)

 FUNCTION loadContacts():
   LOAD contacts from the file (contacts.ser)
   ADD loaded contacts to the JTable
   IF file does not exist:
     START with an empty phonebook

 FUNCTION playSound():
   PLAY sound when a contact is added, updated, or deleted

END Phonebook Application
```