ГУАП

КАФЕДРА № 42

ОТЧЕТ   
ЗАЩИЩЕН С ОЦЕНКОЙ

ПРЕПОДАВАТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ассистент |  |  |  | Н.А. Янковский |
| должность, уч. степень, звание |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ |
| АРХИТЕКТУРА СИСТЕМЫ. MV-ШАБЛОНЫ |
| по курсу: ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ |
|  |
|  |

РАБОТУ ВЫПОЛНИЛ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| СТУДЕНТ ГР. № | 4121 |  |  |  | Н.С. Савосин |
|  |  |  | подпись, дата |  | инициалы, фамилия |

Санкт-Петербург 2023

# ЦЕЛЬ РАБОТЫ

Вспомнить и применить на практике принципы объектно-ориентированного программирования (ООП), используя основные элементы и понятия ООП: классы, объекты, наследование, инкапсуляция, полиморфизм, абстракция. Внедрить в проект шаблон проектирования MVP.

# ЗАДАНИЕ

Необходимо проект своей курсовой работы, написанной в третьем семестре по дисциплине «Основы программирования» переделать в соответствии со стандартами шаблона MVP (Model-View-Presenter). Если же проект был некачественный либо не соответствует требованиям ниже, то можно и нужно дополнить его либо создать новый проект на любую тематику. Проект должен иметь графический пользовательский интерфейс (User Interface, UI), а также может быть написан на любом языке программирования.

# ОПИСАНИЕ ХОДА РАЗРАБОТКИ И НАЗНАЧЕНИЕ ИСПОЛЬЗУЕМЫХ ТЕХНОЛОГИЙ

В данной лабораторной работе был разработан проект в соответствии со стандартами шаблона MVP (Model-View-Presenter).

MVP - это шаблон проектирования, который разделяет систему на три основных компонента: Model (модель), View (представление) и Presenter (презентер).

Model - это компонент, который содержит данные и бизнес-логику приложения. В этом примере модель представлена интерфейсом UserRepository и его реализацией UserRepositoryImpl.

View - это компонент, который отвечает за отображение данных пользователю. В этом примере представление представлено интерфейсом UserView и его реализацией UserViewImpl.

Presenter - это компонент, который управляет взаимодействием между моделью и представлением. В этом примере презентер представлен классом UserPresenter.

В классе Main создаются экземпляры классов UserRepositoryImpl, UserViewImpl и UserPresenter, и презентер используется для отображения списка пользователей в пользовательском интерфейсе.

Интерфейс SearchStrategy и SortStrategy используются для определения критериев поиска и сортировки пользователей в модели. Класс User репрезентует объект пользователя со свойствами name, age и email.

В UserViewImpl определяется пользовательский интерфейс с кнопками для добавления, удаления, сортировки и поиска пользователей, а также список пользователей, который отображается в пользовательском интерфейсе.

Технологии, используемые в этом коде, включают в себя язык программирования Java, библиотеки Swing и AWT для создания пользовательского интерфейса, а также коллекции Java для хранения и управления списком пользователей.

# ИСХОДНЫЙ КОД ПРОГРАММЫ

Класс Main:

//Добавлен класс Main, содержащий метод main для запуска приложения.  
public class Main {  
 public static void main(String[] args) {  
 //Создан объект userRepository типа UserRepositoryImpl для работы с данными пользователей.  
 UserRepository userRepository = new UserRepositoryImpl();  
 //Создан объект userView типа UserViewImpl для отображения данных пользователей.  
 UserView userView = new UserViewImpl();  
 //Создан объект userPresenter типа UserPresenter, который объединяет представление и хранилище данных пользователей.  
 UserPresenter userPresenter = new UserPresenter(userView, userRepository);  
 }  
}

Интерфейс SearchStrategy:

// Добавлен интерфейс для поиска.  
public interface SearchStrategy {  
 //Интерфейс содержит один метод test, который принимает объект типа User и возвращает логическое значение.  
 boolean test(User user);  
}

Интерфейс SortStrategy:

import java.util.Comparator;  
// SortStrategy расширяет Comparator для возможности сравнения объектов User.  
public interface SortStrategy extends Comparator<User> {  
}

Класс User:

//Создание класса User  
public class User {  
 // Создание приватных полей name, age и email  
 private String name;  
 private int age;  
 private String email;  
 // Создание конструктора класса User с параметрами name, age и email  
 public User(String name, int age, String email) {  
 this.name = name;  
 this.age = age;  
 this.email = email;  
 }  
 // Создание геттеров для полей name, age и email  
 public String getName() {  
 return name;  
 }  
  
 public int getAge() {  
 return age;  
 }  
  
 public String getEmail() {  
 return email;  
 }  
  
 // Создание метода toString() для отладки и возвращения строкового представления объекта User  
 public String toString() {  
 return name + ", " + age + ", " + email;  
 }  
}

Класс UserPresenter:

import java.util.List;  
// Добавлен класс UserPresenter для обработки пользовательского интерфейса и связи с хранилищем данных  
public class UserPresenter {  
 private UserView userView;  
 private UserRepository userRepository;  
 // Создан конструктор класса UserPresenter для инициализации объектов UserView и UserRepository  
 public UserPresenter(UserView userView, UserRepository userRepository) {  
 this.userView = userView;  
 this.userRepository = userRepository;  
 userView.showUserList(userRepository.getUserList());  
 }  
 // Добавлен метод addUser() для добавления нового пользователя в хранилище данных и обновления списка пользователей в пользовательском интерфейсе  
 public void addUser(String name, int age, String email) {  
 User user = new User(name, age, email);  
 userRepository.addUser(user);  
 userView.showUserList(userRepository.getUserList());  
 }  
 // Добавлен метод deleteUser() для удаления пользователя из хранилища данных и обновления списка пользователей в пользовательском интерфейсе  
 public void deleteUser(String userName) {  
 userRepository.deleteUser(userName);  
 userView.showUserList(userRepository.getUserList());  
 }  
 // Добавлен метод sortUserList() для сортировки списка пользователей с помощью переданной стратегии сортировки и обновления списка пользователей в пользовательском интерфейсе  
 public void sortUserList(SortStrategy strategy) {  
 List<User> userList = userRepository.sortUserList(strategy);  
 userView.showUserList(userList);  
 }  
 // Добавлен метод searchUserList() для поиска пользователей в списке с помощью переданной стратегии поиска и обновления списка пользователей в пользовательском интерфейсе  
 public List<User> searchUserList(SearchStrategy strategy) {  
 List<User> userList = userRepository.searchUserList(strategy);  
 userView.showUserList(userList);  
 return userList;  
 }  
 // Добавлен метод showUserList() для отображения списка пользователей в пользовательском интерфейсе  
 public void showUserList() {  
 List<User> userList = userRepository.getUserList();  
 userView.showUserList(userList);  
 }  
 // Добавлен перегруженный метод showUserList() для отображения переданного списка пользователей в пользовательском интерфейсе  
 public void showUserList(List<User> userList) {  
 userList = userRepository.getUserList();  
 userView.showUserList(userList);  
 }  
}

Интерфейс UserRepository:

import java.util.List;  
// Добавлен интерфейс UserRepository  
public interface UserRepository {  
 // Добавлен метод addUser для добавления пользователя  
 void addUser(User user);  
 // Добавлен метод deleteUser для удаления пользователя по имени  
 void deleteUser(String userName);  
 // Добавлен метод getUserList для получения списка пользователей  
 List<User> getUserList();  
 // Добавлен метод searchUserList для поиска пользователей с использованием переданным типом поиска  
 List<User> searchUserList(SearchStrategy strategy);  
 // Добавлен метод sortUserList для сортировки пользователей с использованием переданной типом сортировки  
 List<User> sortUserList(SortStrategy strategy);  
}

Класс UserRepositoryImpl:

import java.util.List;  
import java.util.ArrayList;  
import java.util.Collections;  
import java.util.Iterator;  
  
// Реализация UserRepositoryImpl, имплементирующего UserRepository  
public class UserRepositoryImpl implements UserRepository {  
 private List<User> userList;  
 // Конструктор UserRepositoryImpl заполняющий userList  
 public UserRepositoryImpl() {  
 userList = new ArrayList<>();  
 }  
 // Реализован метод addUser в UserRepositoryImpl для добавления пользователя в список пользователей.  
 @Override  
 public void addUser(User user) {  
 userList.add(user);  
 }  
 // Реализован метод deleteUser в UserRepositoryImpl для удаления пользователя из списка пользователей.  
 @Override  
 public void deleteUser(String userName) {  
 Iterator<User> iterator = this.getUserList().iterator();  
 while (iterator.hasNext()) {  
 User user = iterator.next();  
 if (user.getName().equals(userName)) {  
 iterator.remove();  
 break;  
 }  
 }  
 }  
 // Реализован метод getUserList в UserRepositoryImpl для получения списка всех пользователей.  
 @Override  
 public List<User> getUserList() {  
 return userList;  
 }  
 // Реализован метод searchUserList в UserRepositoryImpl для поиска пользователей по определенному критерию.  
 @Override  
 public List<User> searchUserList(SearchStrategy strategy) {  
 List<User> result = new ArrayList<>();  
 for (User user : userList) {  
 if (strategy.test(user)) {  
 result.add(user);  
 }  
 }  
 return result;  
 }  
 // Реализован метод sortUserList в UserRepositoryImpl для сортировки списка пользователей по определенному критерию.  
 @Override  
 public List<User> sortUserList(SortStrategy strategy) {  
 List<User> result = new ArrayList<>(userList);  
 Collections.*sort*(result, strategy);  
 return result;  
 }  
  
  
}

Интерфейс UserView:

import java.util.List;  
// Добавлен интерфейс UserView  
public interface UserView {  
 // Добавлен метод showUserList в интерфейс UserView  
 void showUserList(List<User> userList);  
 // Добавлен метод showErrorMessage в интерфейс UserView  
 void showErrorMessage(String errorMessage);  
 // Добавлен метод showSuccessMessage в интерфейс UserView  
 void showSuccessMessage(String message);  
}

Класс UserViewImpl:

import javax.swing.\*;  
import javax.swing.event.\*;  
import java.awt.\*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.awt.event.AdjustmentEvent;  
import java.awt.event.AdjustmentListener;  
import java.util.List;  
public class UserViewImpl implements UserView, ActionListener, ChangeListener, AdjustmentListener {  
 private JFrame frame;  
 private JPanel panel;  
 private JList<User> listArea;  
 private JButton addButton;  
 private JButton deleteButton;  
 private JButton sortButton;  
 private JButton searchButton;  
 private JButton showListButton;  
 private UserPresenter presenter;  
 private List<User> currentList;  
  
 public UserViewImpl() {  
 frame = new JFrame("User Management System");  
 frame.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 frame.setSize(600, 500);  
  
 panel = new JPanel(new BorderLayout());  
  
 // Добавление списка пользователей в центр панели, внутри панели прокрутки  
 listArea = new JList<User>();  
 JScrollPane scrollPane = new JScrollPane(listArea);  
 panel.add(scrollPane, BorderLayout.*CENTER*);  
  
 // Добавление панели кнопок в нижнюю часть панели  
 JPanel buttonPanel = new JPanel(new FlowLayout(FlowLayout.*CENTER*, 10, 10));  
 panel.add(buttonPanel, BorderLayout.*SOUTH*);  
  
 // Добавление кнопок на панель кнопок  
 addButton = new JButton("Add User");  
 addButton.addActionListener(this);  
 buttonPanel.add(addButton);  
  
 deleteButton = new JButton("Delete User");  
 deleteButton.addActionListener(this);  
 buttonPanel.add(deleteButton);  
  
 sortButton = new JButton("Sort");  
 sortButton.addActionListener(this);  
 buttonPanel.add(sortButton);  
  
 searchButton = new JButton("Search Users");  
 searchButton.addActionListener(this);  
 buttonPanel.add(searchButton);  
  
 showListButton = new JButton("Show User List");  
 showListButton.addActionListener(this);  
 buttonPanel.add(showListButton);  
  
 panel.add(new JScrollPane(listArea), BorderLayout.*CENTER*);  
  
 frame.setContentPane(panel);  
 frame.setVisible(true);  
  
 presenter = new UserPresenter(this, new UserRepositoryImpl());  
 }  
  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 // Добавление пользователя  
 if (e.getSource() == addButton) {  
 String name = JOptionPane.*showInputDialog*("Enter name:");  
 String ageStr = JOptionPane.*showInputDialog*("Enter age:");  
 String email = JOptionPane.*showInputDialog*("Enter email:");  
  
 try {  
 int age = Integer.*parseInt*(ageStr);  
 presenter.addUser(name, age, email);  
 } catch (NumberFormatException ex) {  
 showErrorMessage("Invalid age: " + ageStr);  
 }  
 // Удаление пользователя  
 } else if (e.getSource() == deleteButton) {  
 if (currentList.isEmpty()) {  
 JOptionPane.*showMessageDialog*(frame, "No users to delete.");  
 return;  
 }  
  
 String[] userNames = currentList.stream().map(User::getName).toArray(String[]::new);  
 String selectedUser = (String) JOptionPane.*showInputDialog*(frame, "Select user to delete:", "Delete", JOptionPane.*PLAIN\_MESSAGE*, null, userNames, null);  
 if (selectedUser != null) {  
 // Удаление выбранного пользователя из списка  
 presenter.deleteUser(selectedUser);  
 }  
 // Сортировка списка пользователей  
 } else if (e.getSource() == sortButton) {  
 // Показ дилога с выбором типа сортировки  
 String[] sortTypes = {"Name", "Age", "Email"};  
 int sortTypeIndex = JOptionPane.*showOptionDialog*(frame, "Select sort type:", "Sort", JOptionPane.*DEFAULT\_OPTION*, JOptionPane.*PLAIN\_MESSAGE*, null, sortTypes, sortTypes[0]);  
  
 // Показ дилога с выбором типа сортировки по возрастанию или убыванию  
 String[] sortDirections = {"Ascending", "Descending"};  
 int sortDirectionIndex = JOptionPane.*showOptionDialog*(frame, "Select sort direction:", "Sort", JOptionPane.*DEFAULT\_OPTION*, JOptionPane.*PLAIN\_MESSAGE*, null, sortDirections, sortDirections[0]);  
  
 // Сортировка по выбранным фильтрам  
 SortStrategy sortStrategy = null;  
 switch (sortTypeIndex) {  
 case 0:  
 sortStrategy = (sortDirectionIndex == 0) ? new NameAscendingSort() : new NameDescendingSort();  
 break;  
 case 1:  
 sortStrategy = (sortDirectionIndex == 0) ? new AgeAscendingSort() : new AgeDescendingSort();  
 break;  
 case 2:  
 sortStrategy = (sortDirectionIndex == 0) ? new EmailAscendingSort() : new EmailDescendingSort();  
 break;  
 default:  
 break;  
 }  
 if (sortStrategy != null) {  
 presenter.sortUserList(sortStrategy);  
 }  
 // Поиск пользователей  
 } else if (e.getSource() == searchButton) {  
 String searchStr = JOptionPane.*showInputDialog*("Enter search string:");  
 List<User> userList = presenter.searchUserList(user -> user.getName().contains(searchStr) ||  
 user.getEmail().contains(searchStr));  
 showUserList(userList);  
 // Показать список пользователей  
 } else if (e.getSource() == showListButton) {  
 presenter.showUserList();  
 }  
 }  
 // Рализация метода showUserList для вывода нынешнего спика пользователей  
 @Override  
 public void showUserList(List<User> userList) {  
 currentList = userList;  
 DefaultListModel<User> model = new DefaultListModel<>();  
 for (User user : userList) {  
 model.addElement(user);  
 }  
 listArea.setModel(model);  
 }  
 // Рализация метода showErrorMessage для вывода сообщения об ошибке  
 @Override  
 public void showErrorMessage(String message) {  
 JOptionPane.*showMessageDialog*(frame, message, "Error", JOptionPane.*ERROR\_MESSAGE*);  
 }  
 // Рализация метода showSuccessMessage для вывода сообщения об успешной выполненой операции  
 @Override  
 public void showSuccessMessage(String message) {  
 JOptionPane.*showMessageDialog*(frame, message, "Success", JOptionPane.*INFORMATION\_MESSAGE*);  
 }  
 // Добавлен класс NameAscendingSort для сортировки пользователей по имени в порядке возрастания  
 private static class NameAscendingSort implements SortStrategy {  
 @Override  
 public int compare(User u1, User u2) {  
 return u1.getName().compareTo(u2.getName());  
 }  
 }  
 // Добавлен класс NameDescendingSort для сортировки пользователей по имени в порядке убывания  
 private static class NameDescendingSort implements SortStrategy {  
 @Override  
 public int compare(User u1, User u2) {  
 return u2.getName().compareTo(u1.getName());  
 }  
 }  
 // Добавлен класс AgeAscendingSort для сортировки пользователей по возрасту в порядке возрастания  
 private static class AgeAscendingSort implements SortStrategy {  
 @Override  
 public int compare(User u1, User u2) {  
 return Integer.*compare*(u1.getAge(), u2.getAge());  
 }  
 }  
 // Добавлен класс AgeDescendingSort для сортировки пользователей по возрасту в порядке убывания  
 private static class AgeDescendingSort implements SortStrategy {  
 @Override  
 public int compare(User u1, User u2) {  
 return Integer.*compare*(u2.getAge(), u1.getAge());  
 }  
 }  
 // Добавлен класс EmailAscendingSort для сортировки пользователей по электронной почте в порядке возрастания  
 private static class EmailAscendingSort implements SortStrategy {  
 @Override  
 public int compare(User u1, User u2) {  
 return u1.getEmail().compareTo(u2.getEmail());  
 }  
 }  
 // Добавлен класс EmailDescendingSort для сортировки пользователей по электронной почте в порядке убывания  
 private static class EmailDescendingSort implements SortStrategy {  
 @Override  
 public int compare(User u1, User u2) {  
 return u2.getEmail().compareTo(u1.getEmail());  
 }  
 }  
 // Реализован метод stateChanged для отслеживания изменений состояния  
 @Override  
 public void stateChanged(ChangeEvent e) {  
 }  
 // Реализован метод adjustmentValueChanged для отслеживания изменений значения скролла.  
 @Override  
 public void adjustmentValueChanged(AdjustmentEvent e) {  
  
 }  
}

# РЕЗУЛЬТАТЫ РАБОТЫ ПРОГРАММЫ С ПРИМЕРАМИ

На рисунке 1 представлен вид формы, который получается в ходе выполнения кода.

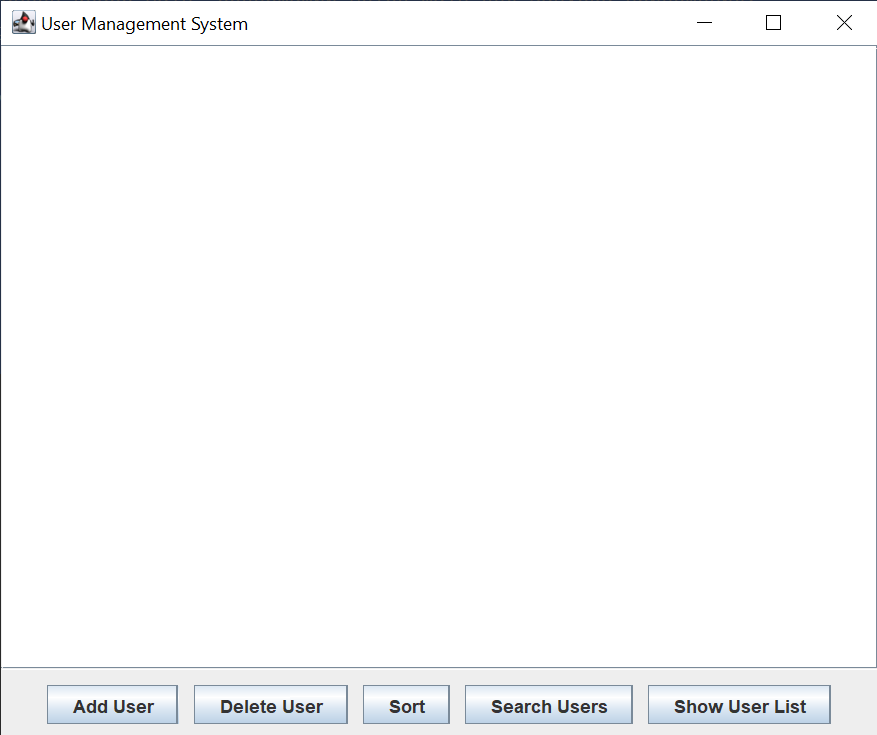


Рисунок 1 – Вид формы для написанного проекта

На рисунках 2 – 4 представлено заполнение полей пользователя с помощью кнопки Add User.

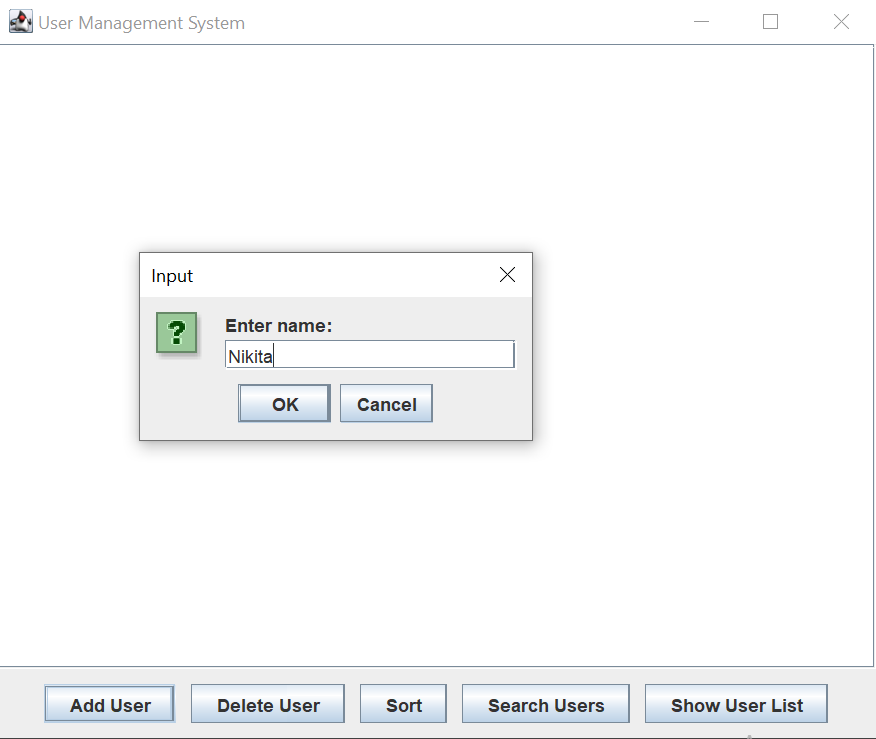


Рисунок 2 – Работа кнопки Add User поле name

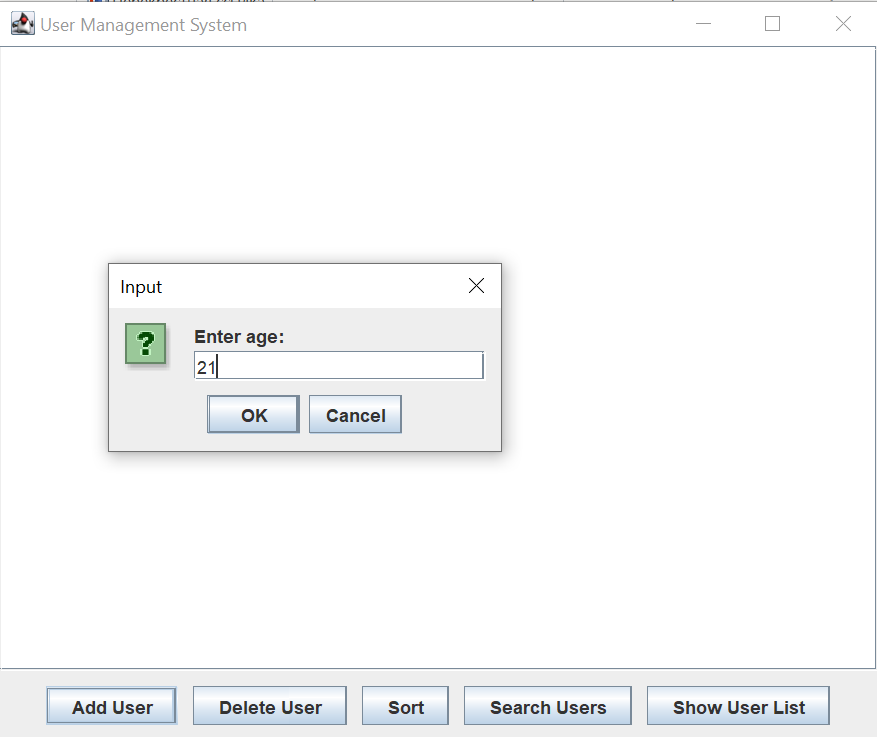


Рисунок 3 – Работа кнопки Add User поле age

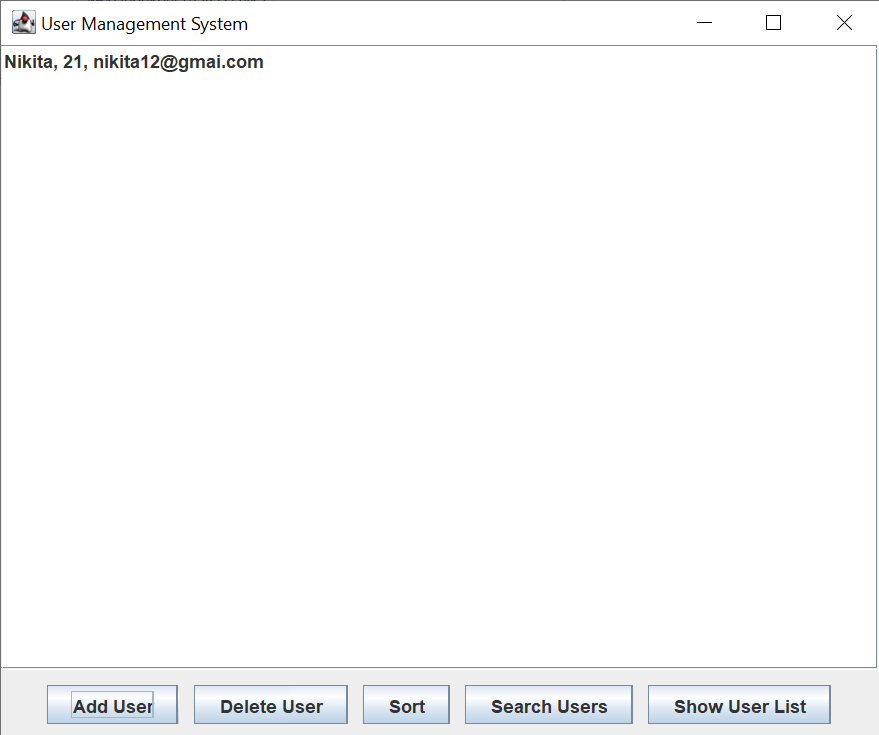


Рисунок 4 – Результат работы кнопки Add User

На рисунках 5 – 6 представлен пример того, если в поле age записать не числовое значение.

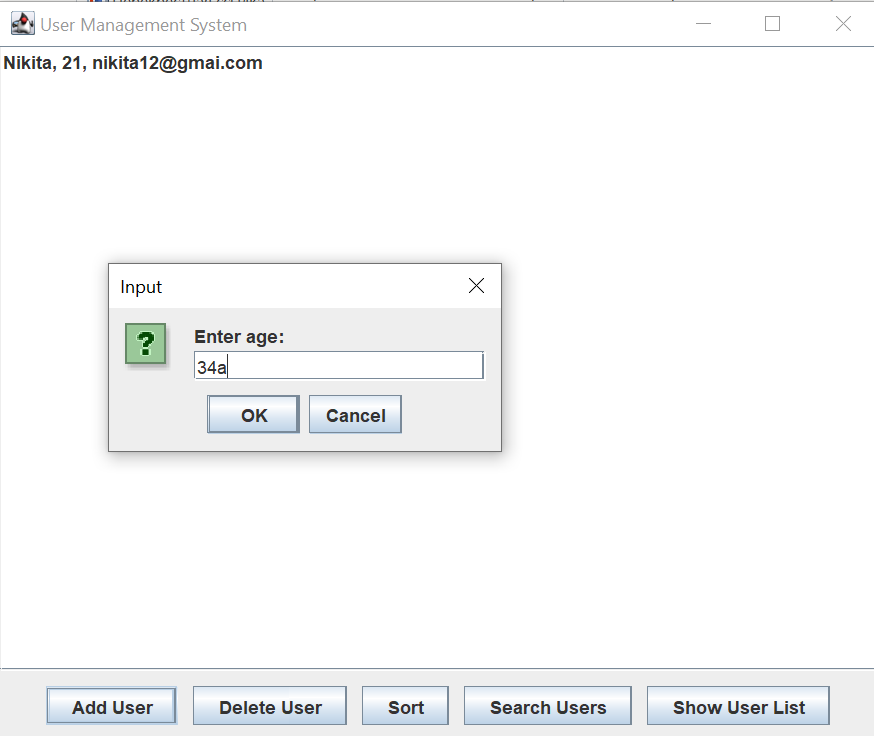


Рисунок 5 – Работа кнопки Add User поле name

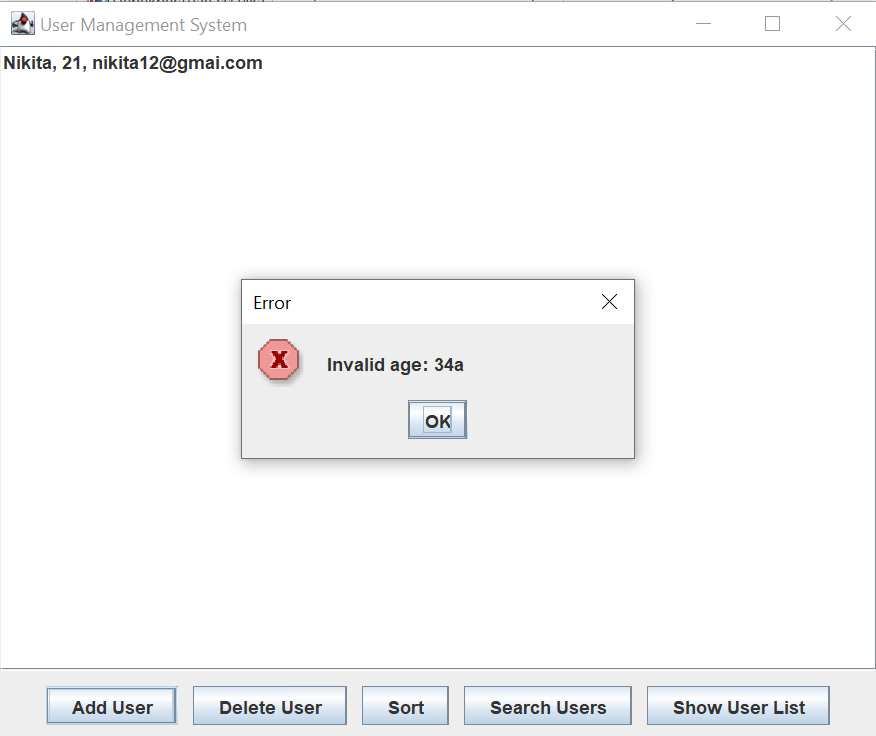


Рисунок 6 – Работа кнопки Add User поле name

На рисунках 7 – 9 показана сортировка списка пользователя. Были выбраны такие параметры: Name, Ascending. Сортировка была выполнена с помощью кнопки Sort.

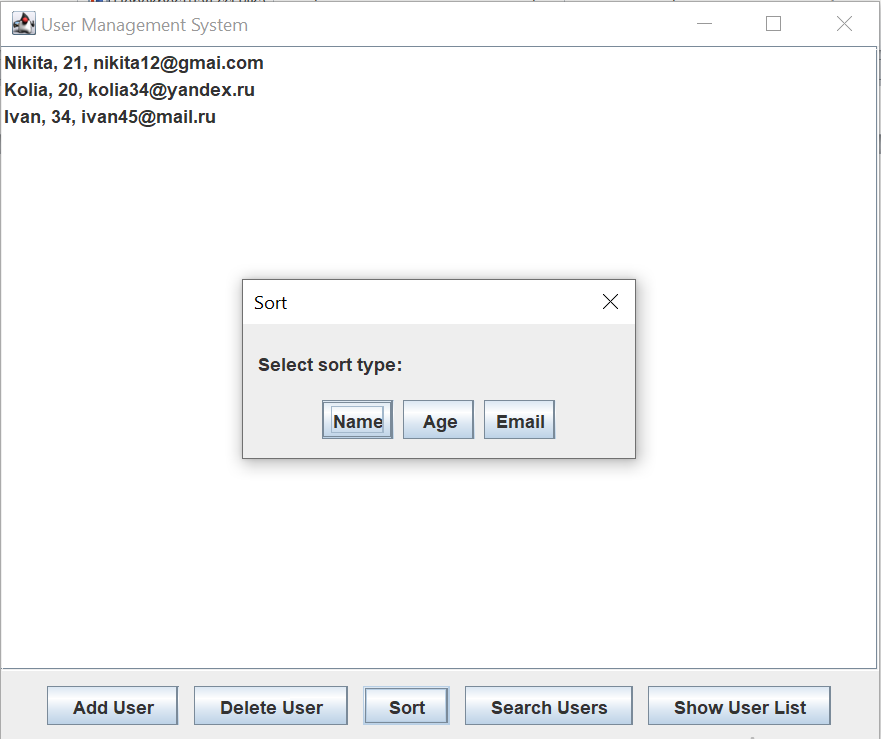


Рисунок 7 – Работа кнопки Sort тип сортировки

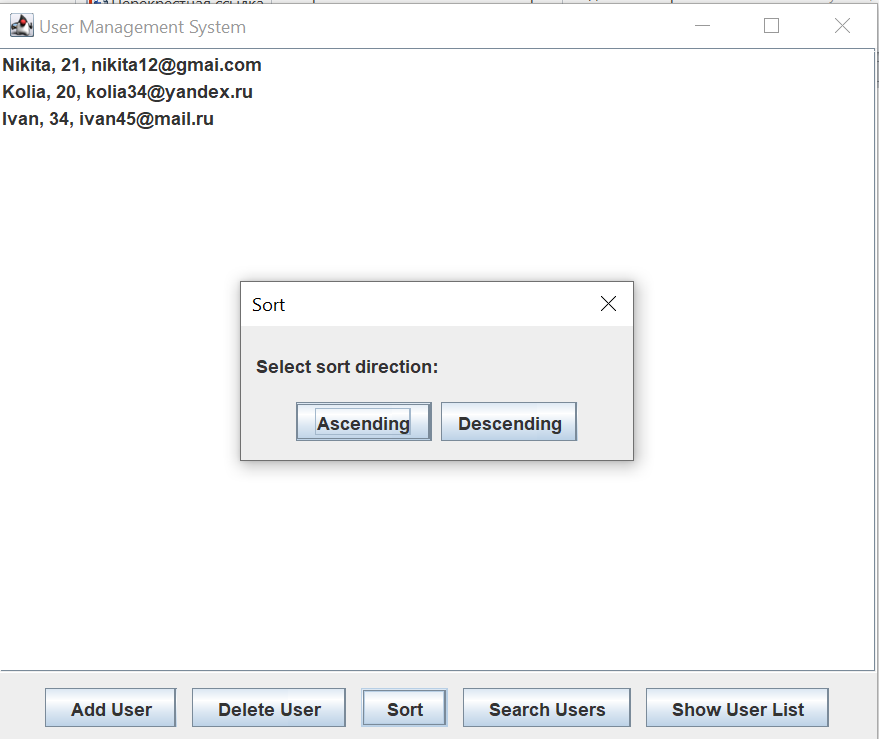


Рисунок 8 – Работа кнопки Sort тип сортировки

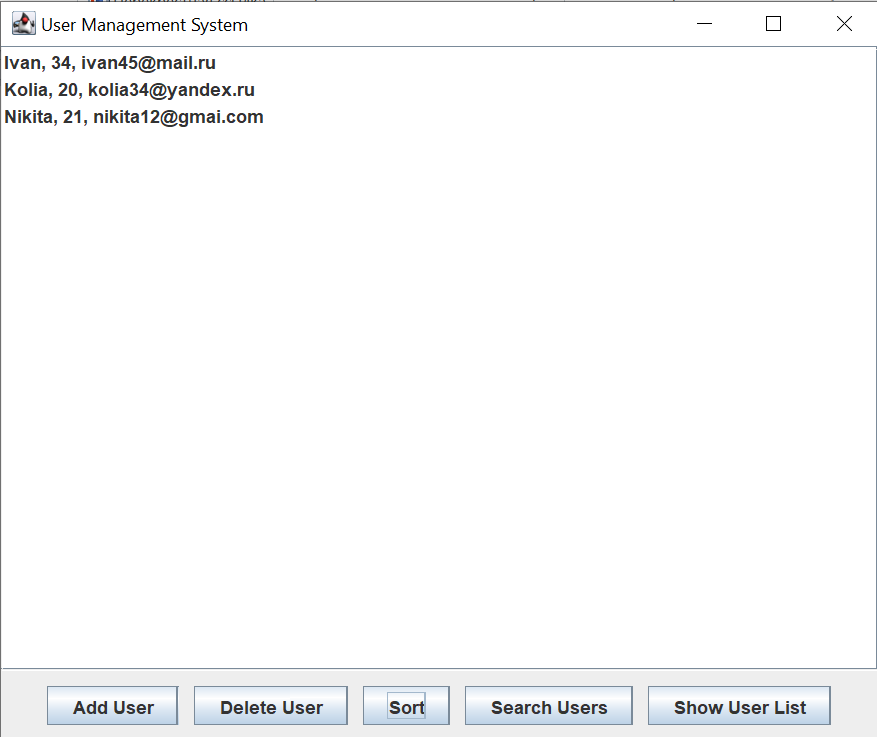


Рисунок 9 – Результат работы кнопки Sort

На рисунке 10 представлена работа кнопки Show User List, которая возвращает нынешний список пользователей.

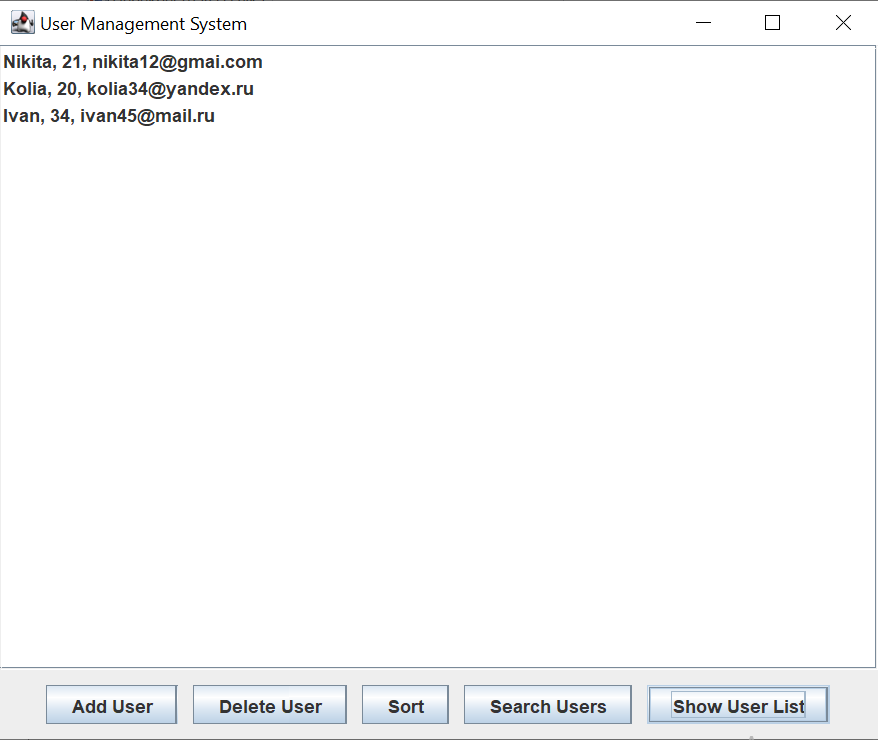


Рисунок 10 – Результат работы кнопки Show User List

На рисунках 11 – 15 представлен поиск пользователей. Поиск осуществляется по полям Name или Email. Поиск может выполняться по частичному совпадению.

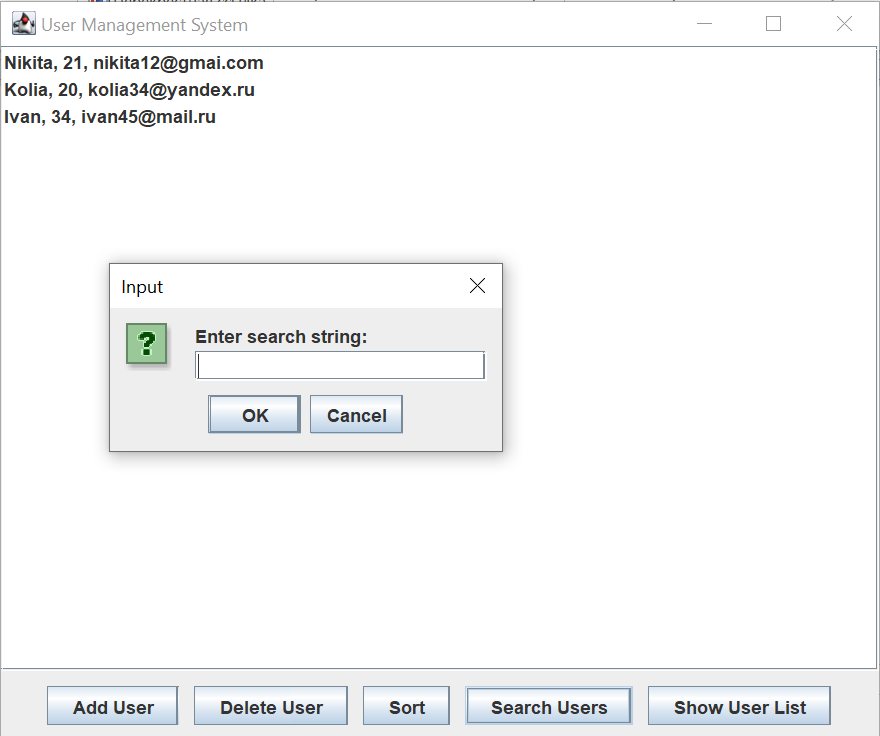


Рисунок 11 – Работа кнопки Search Users

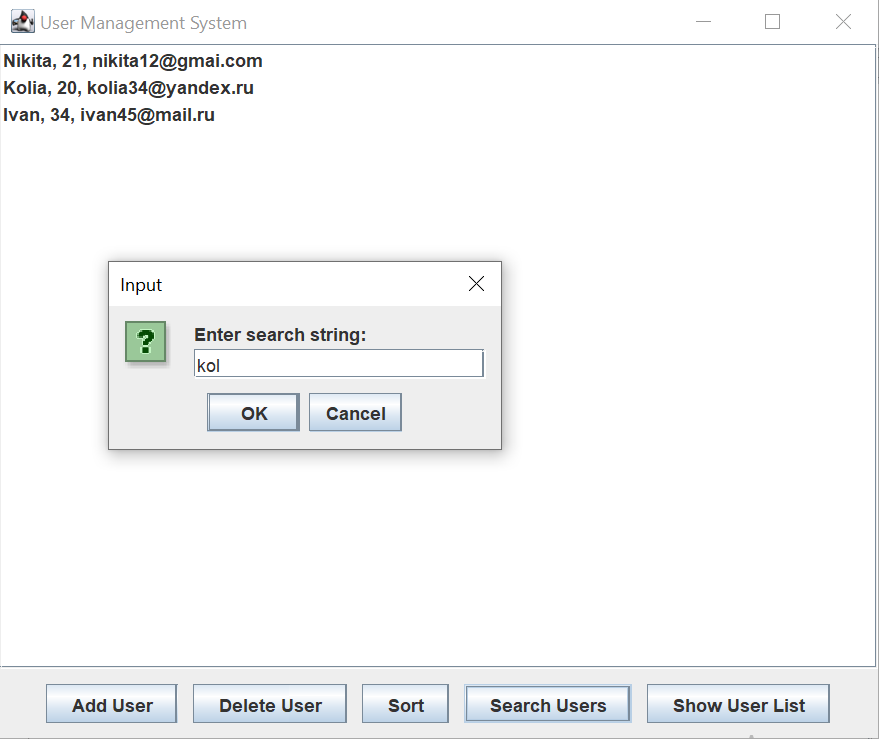


Рисунок 12 – Работа кнопки Search Users

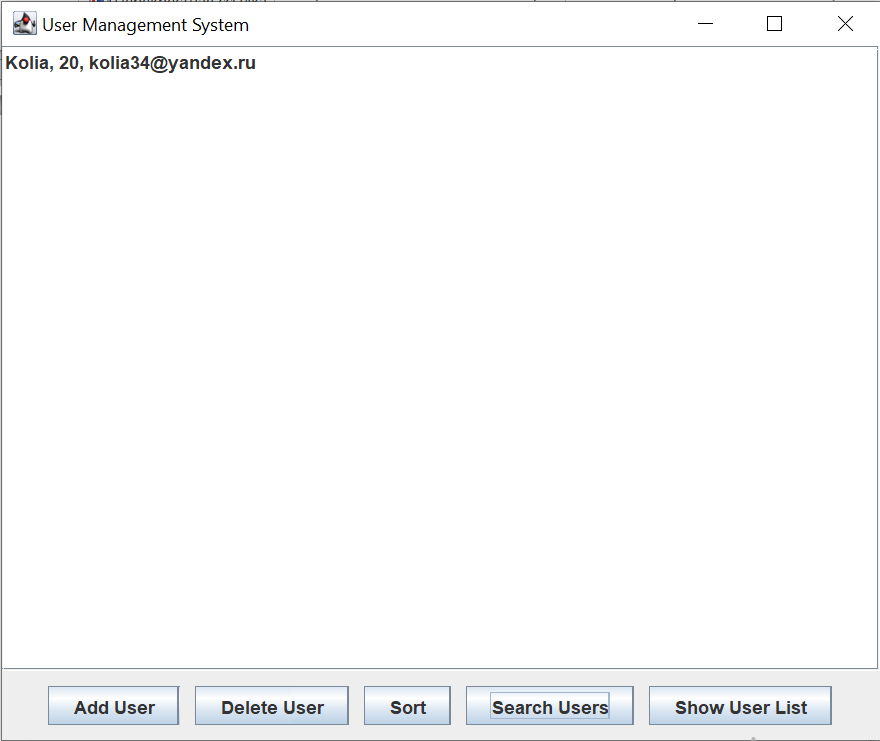


Рисунок 13 – Результат работы кнопки Search Users

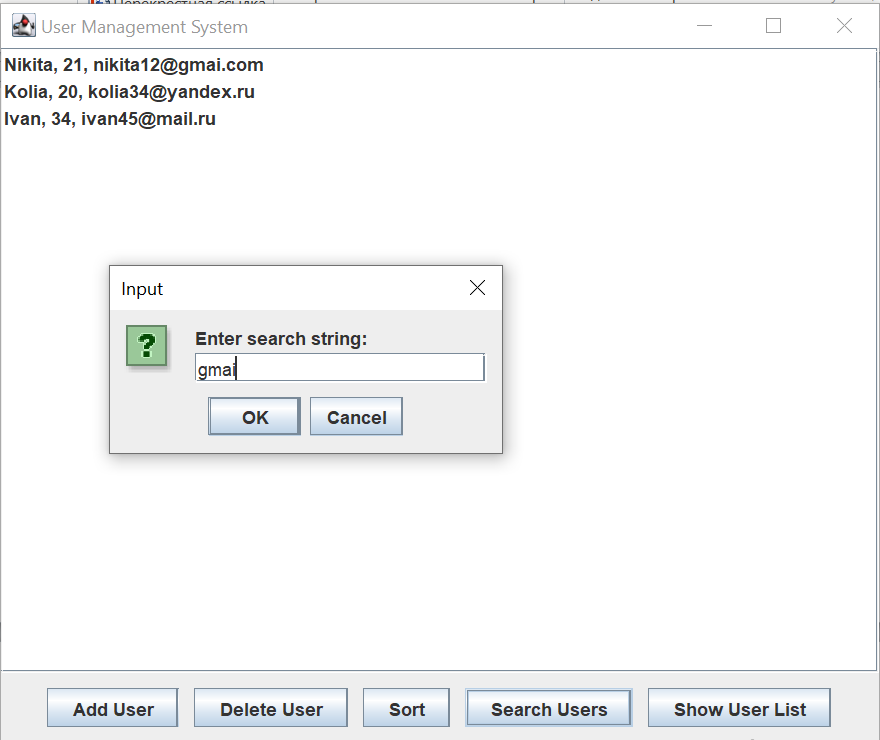


Рисунок 14 – Работа кнопки Search Users

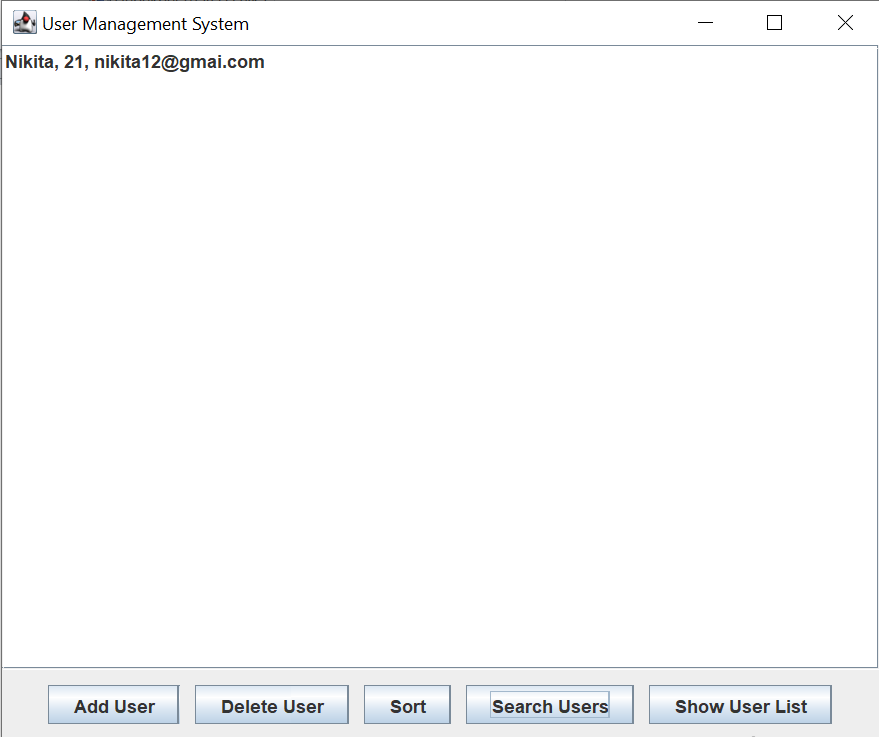


Рисунок 15 – Результат работы кнопки Search Users

На рисунках 16 – 18 представлена работа кнопки Delete User. В диалогов окне выбирается пользователь которого надо удалить из списка.

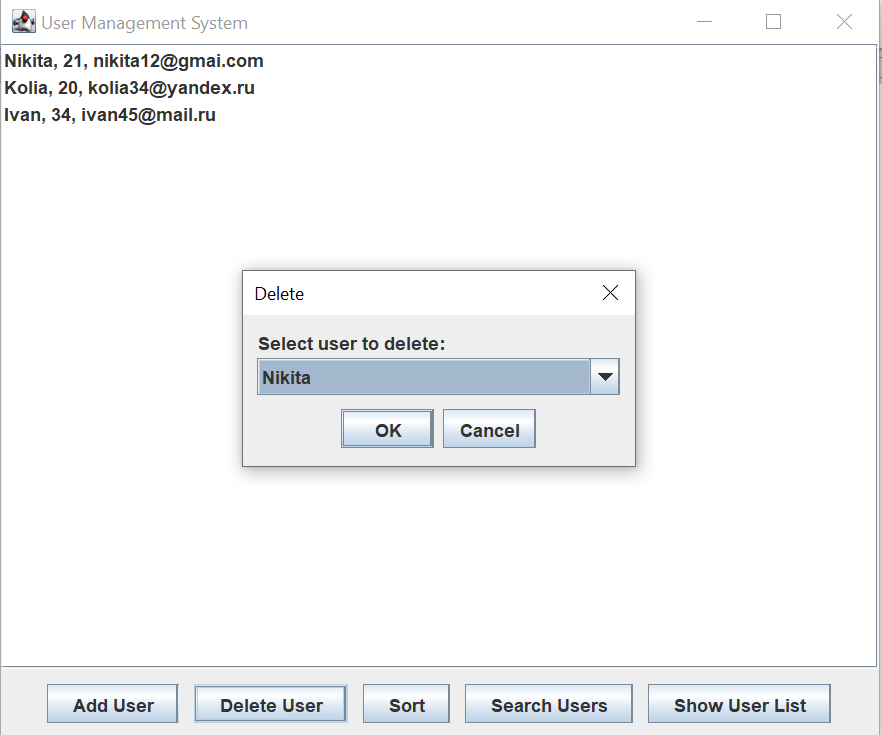


Рисунок 16 – Работа кнопки Delete User

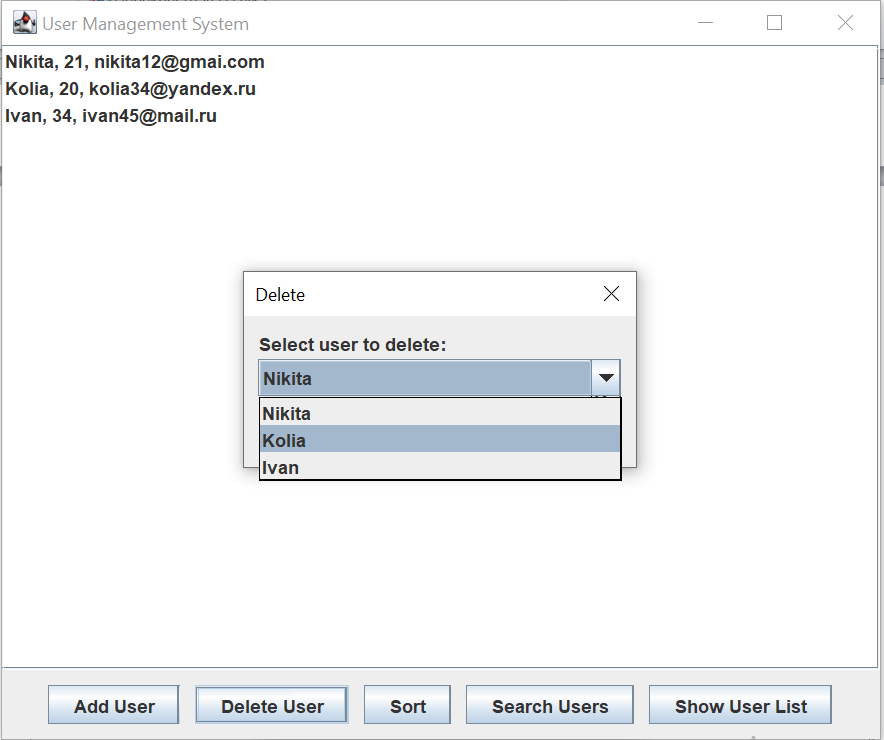


Рисунок 17 – Работа кнопки Delete User

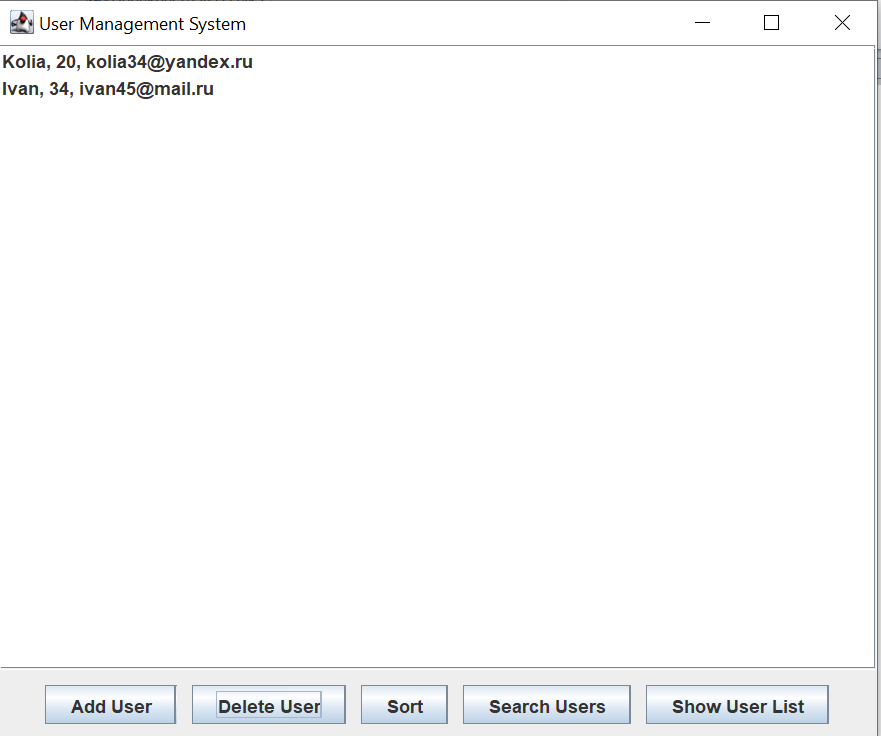


Рисунок 18 – Результат работы кнопки Delete User

# ВЫВОД

Созданный проект представляет собой систему управления пользователями. В ней реализованы три интерфейса: SearchStrategy, SortStrategy и UserView, а также классы User, UserRepository, UserRepositoryImpl и UserPresenter.

User - это модель данных, которая хранит информацию о пользователе. UserRepository - это интерфейс для управления базой данных пользователей. UserRepositoryImpl - это реализация UserRepository, использующая ArrayList для хранения пользователей. UserView - это интерфейс для отображения пользователей в графическом интерфейсе пользователя. UserViewImpl - это реализация UserView, которая использует Swing для создания графического интерфейса.

UserPresenter - это класс, который связывает вместе UserView и UserRepositoryImpl. Он обрабатывает события от пользовательского интерфейса и взаимодействует с UserRepositoryImpl для получения и сохранения данных пользователей.

В Main-классе создается экземпляр UserPresenter и запускается приложение.

Эта система управления пользователями позволяет добавлять, удалять и просматривать пользователей, а также выполнять поиск и сортировку пользователей. Она может быть использована в качестве основы для более сложных систем управления пользователями, которые могут быть разработаны для различных предприятий и организаций.