Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

«Санкт-Петербургский государственный электротехнический университет

“ЛЭТИ” им.В.И.Ульянова (Ленина) »

Кафедра МОЭВМ

**ОТЧЕТ**

**по лабораторно-практической работе № 2, 3, 4, 5**

**по дисциплине «Объектно - ориентированное программирование на языке Java»**

Выполнил Баймухамедов Р. Р.

Преподаватель Павловский М.Г

Факультет КТИ

Группа № 3311

Подпись преподавателя \_\_\_\_\_\_\_\_\_\_

Санкт-Петербург

2024 г

**Цель работы**

1. Знакомство с правилами построения экранной формы приложения
2. Знакомство со способами подключения слушателей событий к графическим компонентам пользовательского интерфейса
3. Знакомство с механизмом обработки исключений в языке Java.
4. Знакомство с организацией обмена данными между объектами экранной формы и файлом.

**Репозиторий**

[https://github.com/brick1ng5654/course-2/tree/main/OOP/lab\_0](https://github.com/brick1ng5654/course-2/tree/main/OOP/lab_02)5

В этом репозитории находятся исходные файлы проекта:

В /src/edu/java/lab05 находится файл CinemaList.java

В /doc находится документация, сгенерированная JavaDoc

Ссылка на видеоотчёт

ht

**Описание назначения экранной формы**

Инструментальная панель JToolBar размещена менеджером граничного размещения BorderLayout.NORTH сразу под линией границы окна. Она состоит из кнопок JButton: “Сохранить список фильмов”, “Добавить фильм”, “Удалить фильм”, “Редактировать данные”, “Загрузить данные”, “Распечатать”

Таблица JTable состоит из трёх столбцов: “Режиссёр”, “Фильм”, “Просмотрен”. Таблица находится в центре (на всем свободном месте) экранной формы.

Компонент поиска состоит из выпадающего списка режиссёров, поле ввода и кнопки поиска, реализованных с помощью JComboBox, JTextField, JButton.

Макет экранной формы представлен на рисунке 2.1

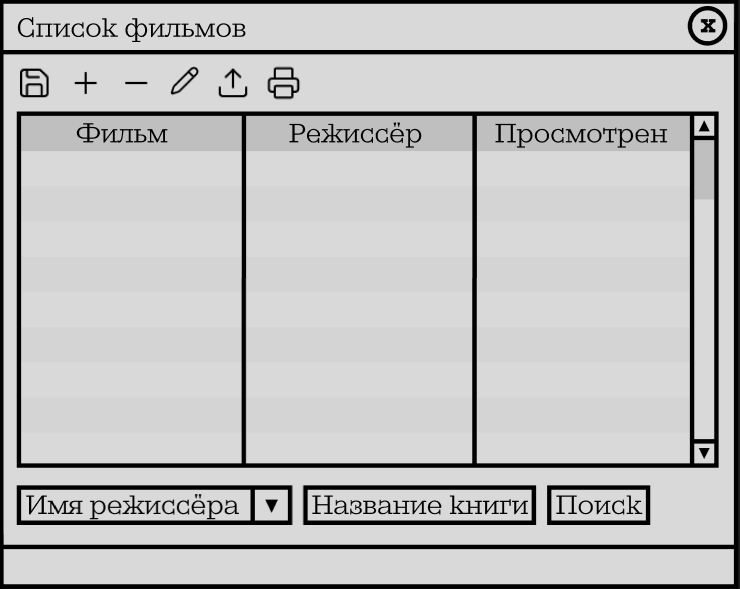


Рисунок 2.1

**Комментарий к макетной форме**

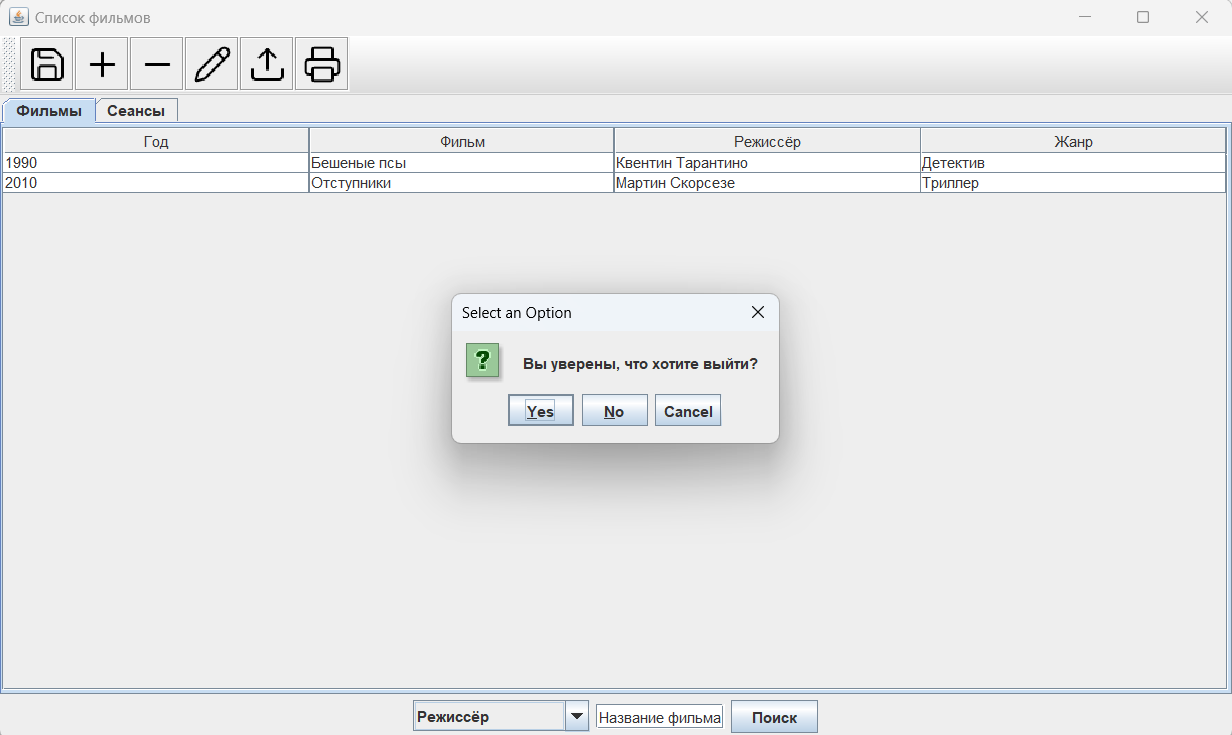
Макетная форма была доработана, добавилась новая панель таблицы с сеансами с такими столбцами: “Фильм”, “Дата”, “Время”, “Продано билетов”. Также была изменена панель таблицы с фильмами на “Год”, “Фильм”, “Режиссёр”, “Жанр”

**Описание действий, которые должны реализовать слушатели**

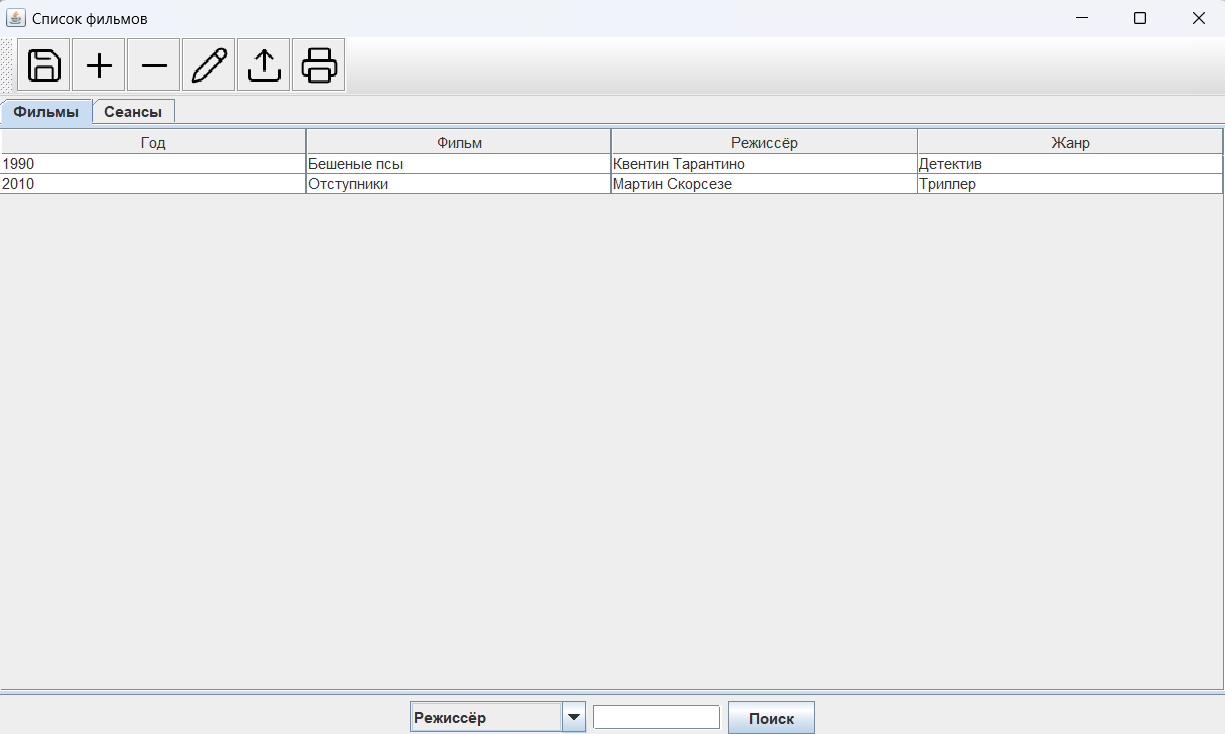
Поскольку кнопки на инструментальной панель связаны с чтением и сохранением файла, который будет реализован в лабораторной работе позднее, то слушатели были добавлены к следующим действиям:

— Закрытие окна (При нажатии на кнопку закрытия приложения, всплывёт подтверждение действия). Работа слушателя продемонстрирована на примере 3.1.

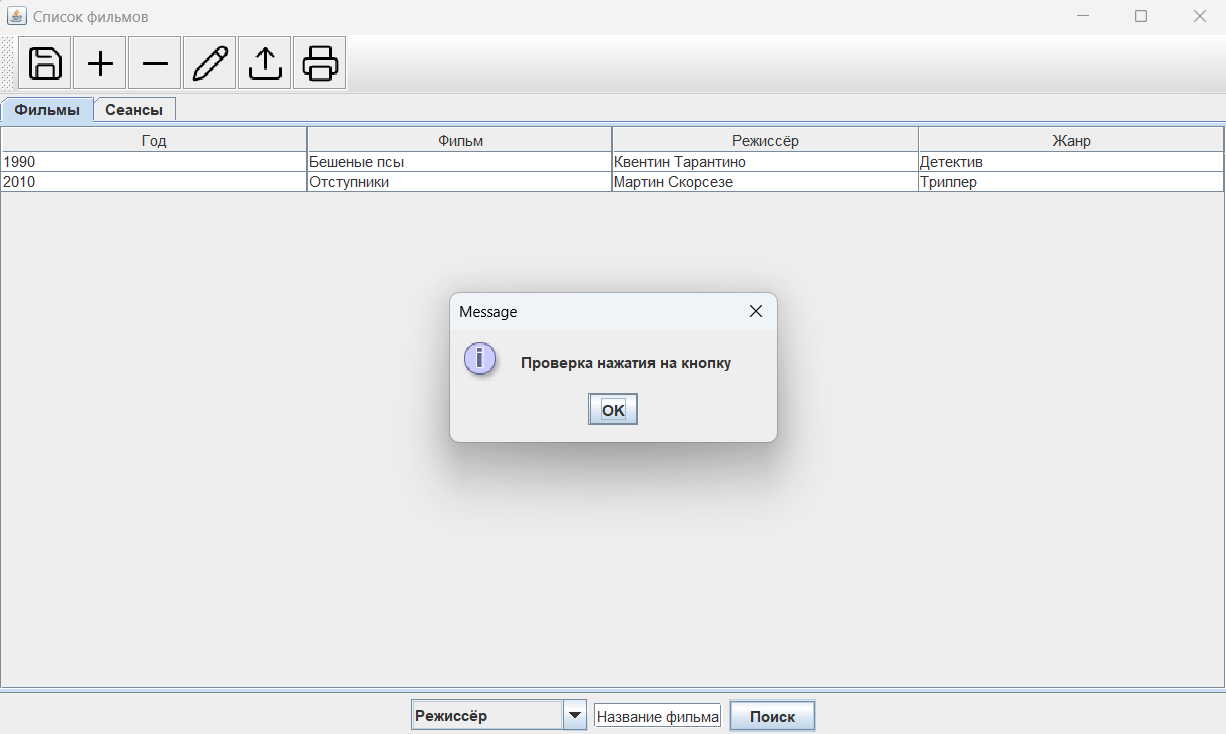
* Фокусировка на поле ввода при поиске (При фокусировке, т.е. нажатии и удержании его на поле ввода, текст поле ввода становится пустым, ожидая ввода пользователя. Если же пользователь ничего не ввёл, а после переместил фокус на другой элемент, то полю присвоится исходный текст). Работа слушателя продемонстрирована на примере 3.2.
* Проверка нажатия на кнопку (При нажатии на кнопку “Поиск” всплывёт окно с уведомлением о нажатии на кнопку). Работа слушателя продемонстрирована на примере 3.3.



* Пример 3.1



Пример 3.2



Пример 3.3

**Добавление нового функционала**

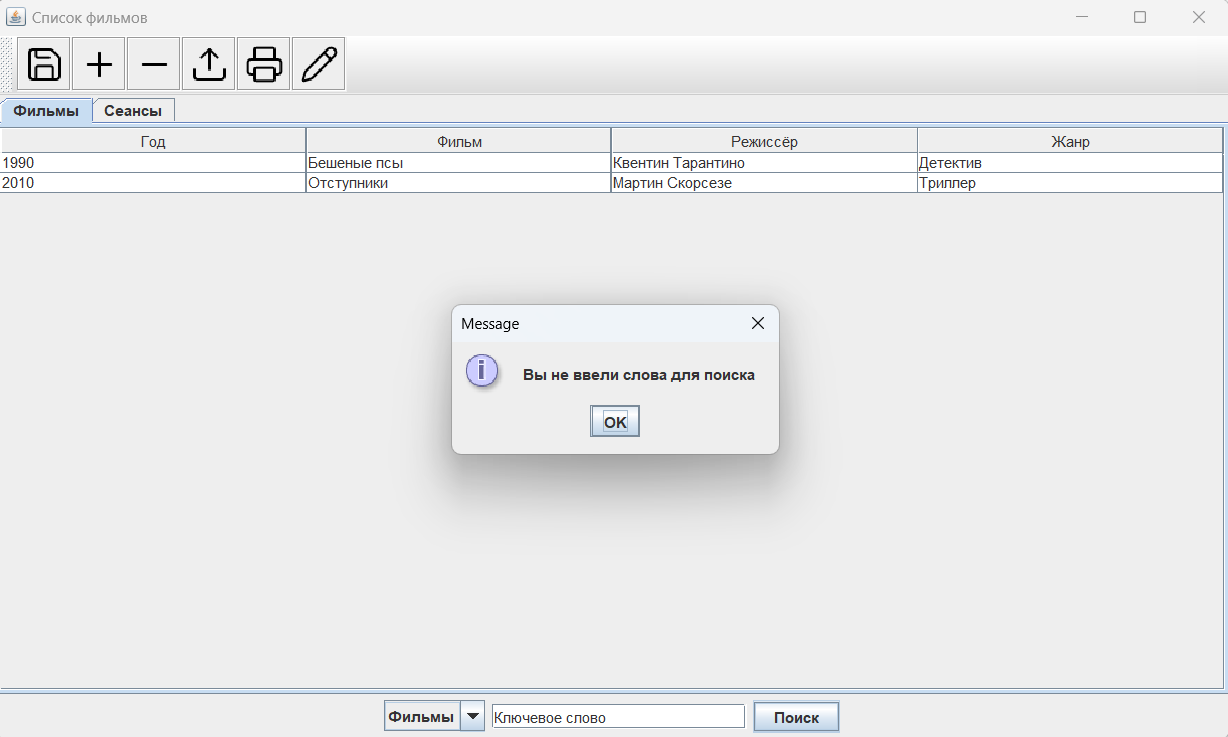
На данном этапе добавлен функционал кнопкам “Добавить данные” и “Удалить данные”.

**Перечень ситуаций, которые контролируются с помощью исключений**

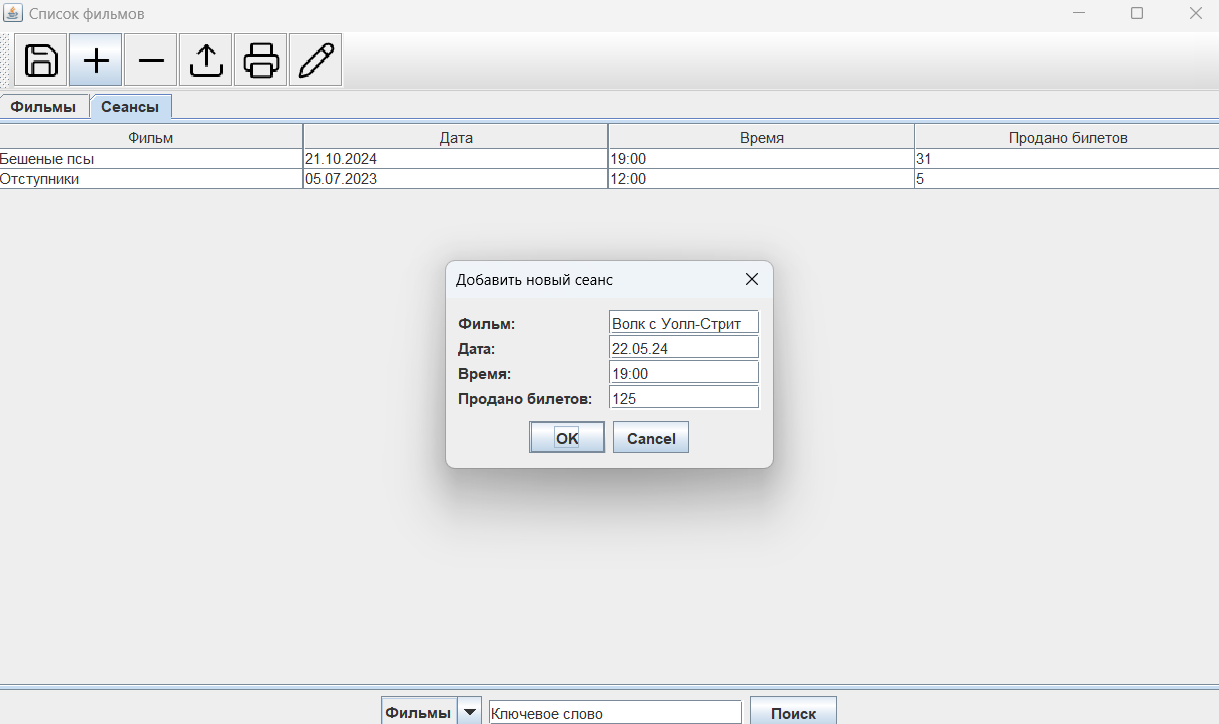
— Попытка нажатия кнопки поиска, при не введённом в поле ввода значении (пример 1)

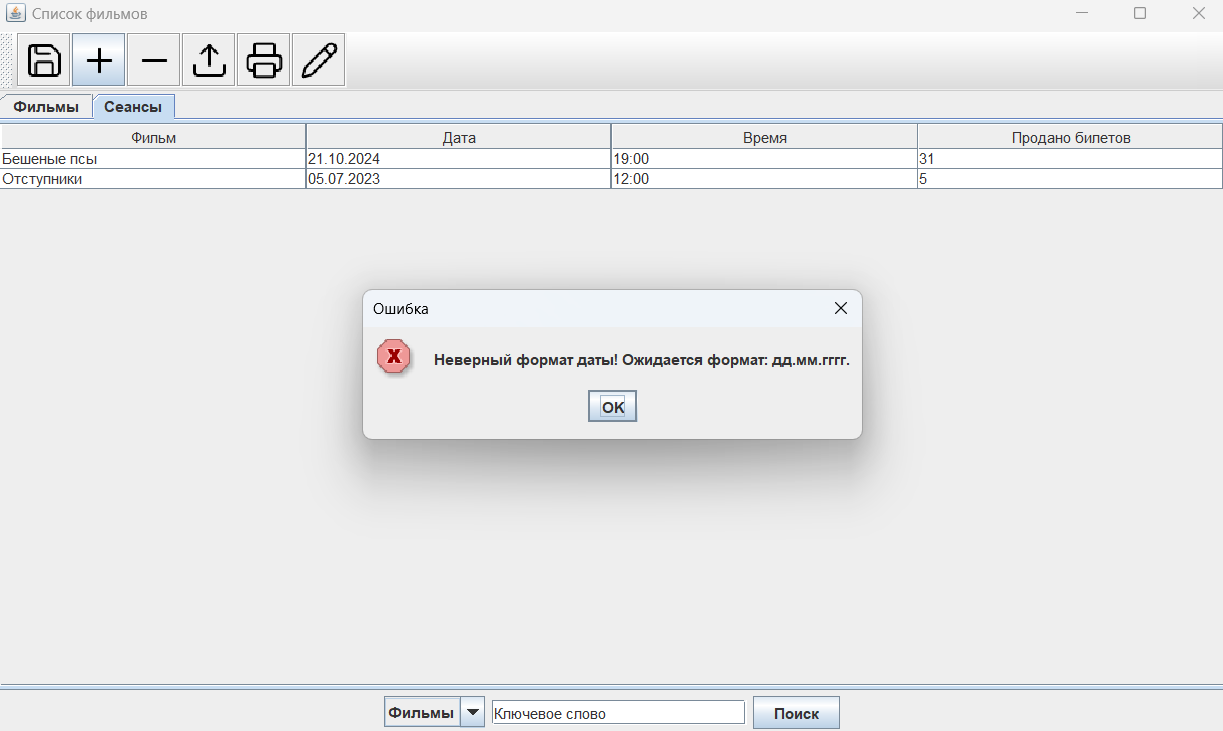
— Попытка ввода значения неправильного формата времени, даты или количества проданных билетов при добавлении данных. (пример 4.2, 4.3 и 4.4 соответственно)

**Скриншоты, иллюстрирующие работу обработчиков ситуаций**

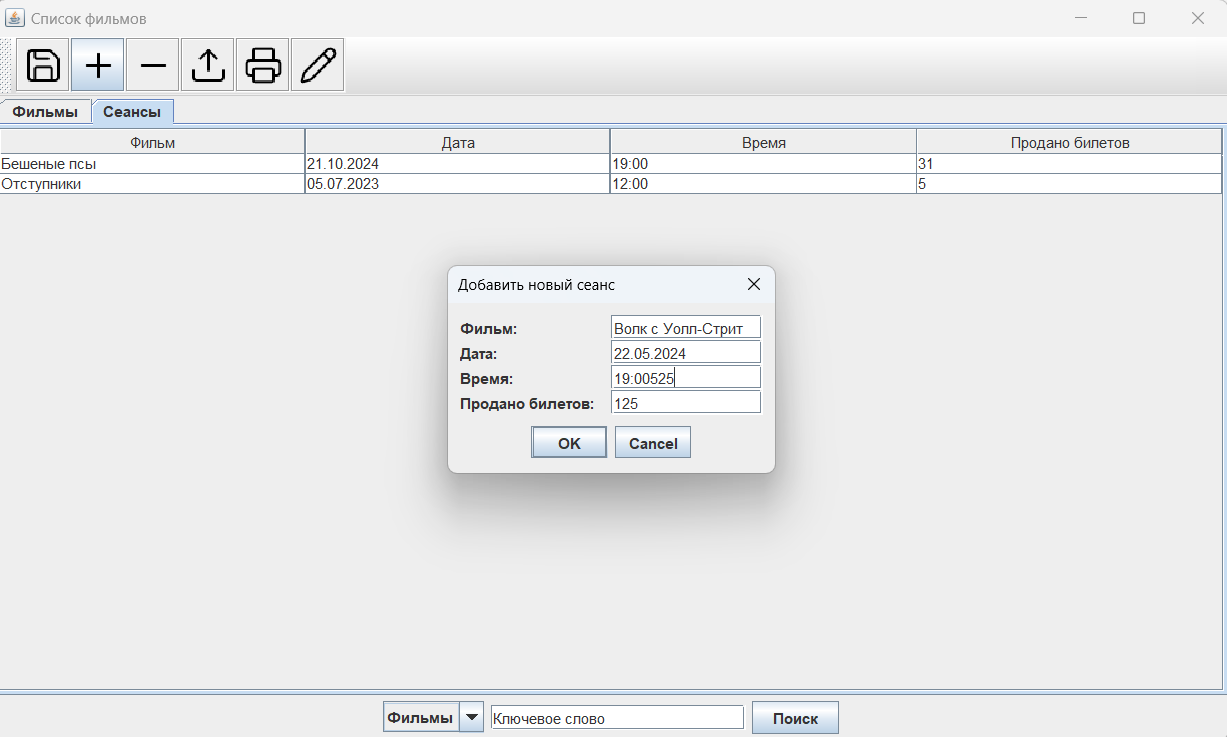


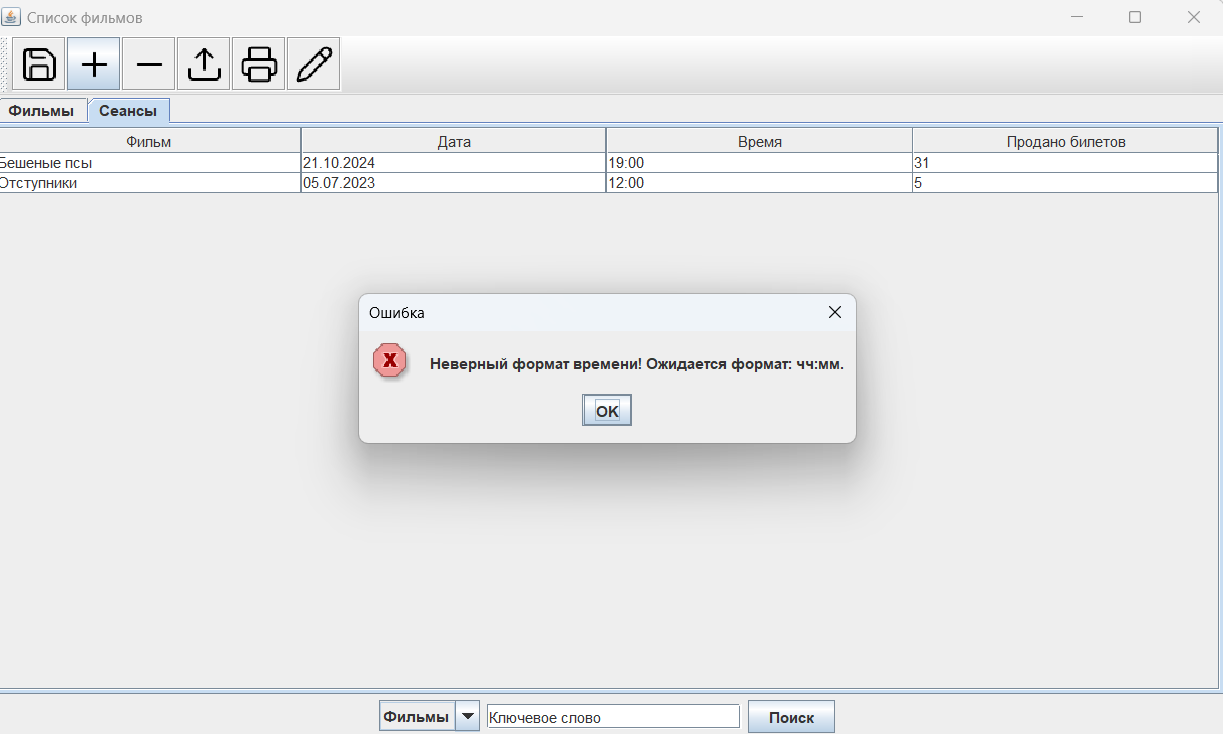
*Пример 4.1*



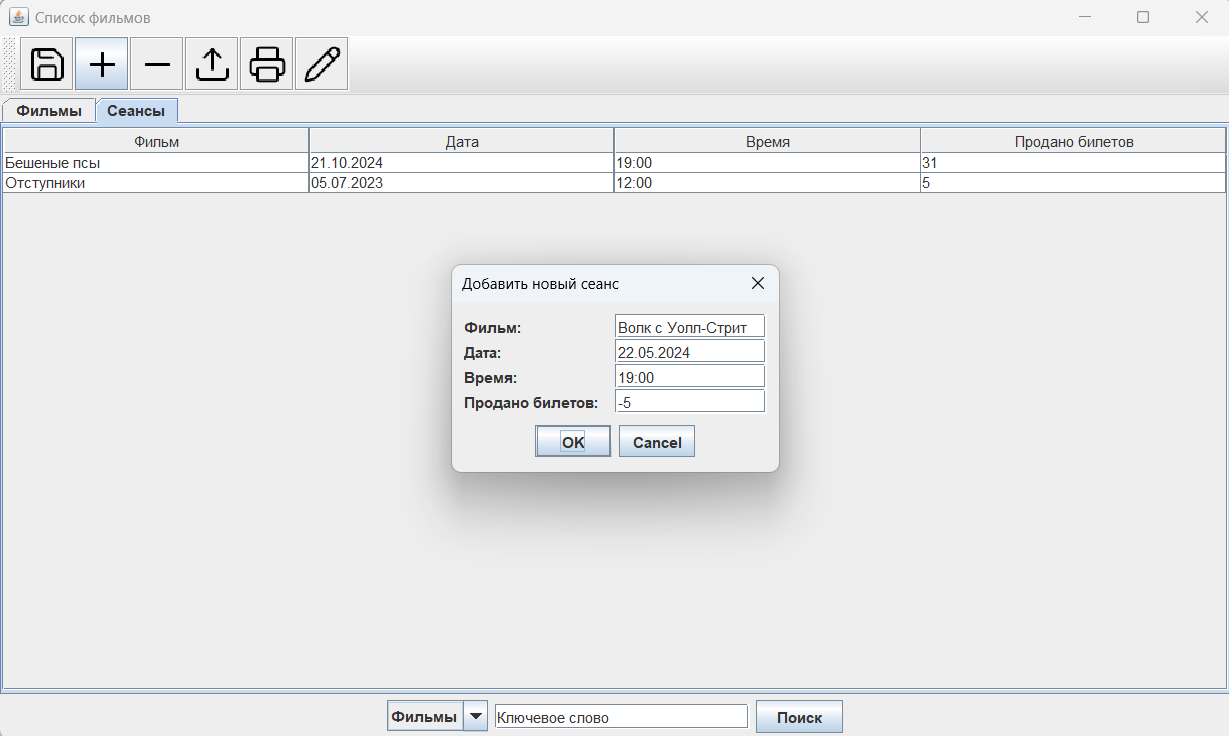


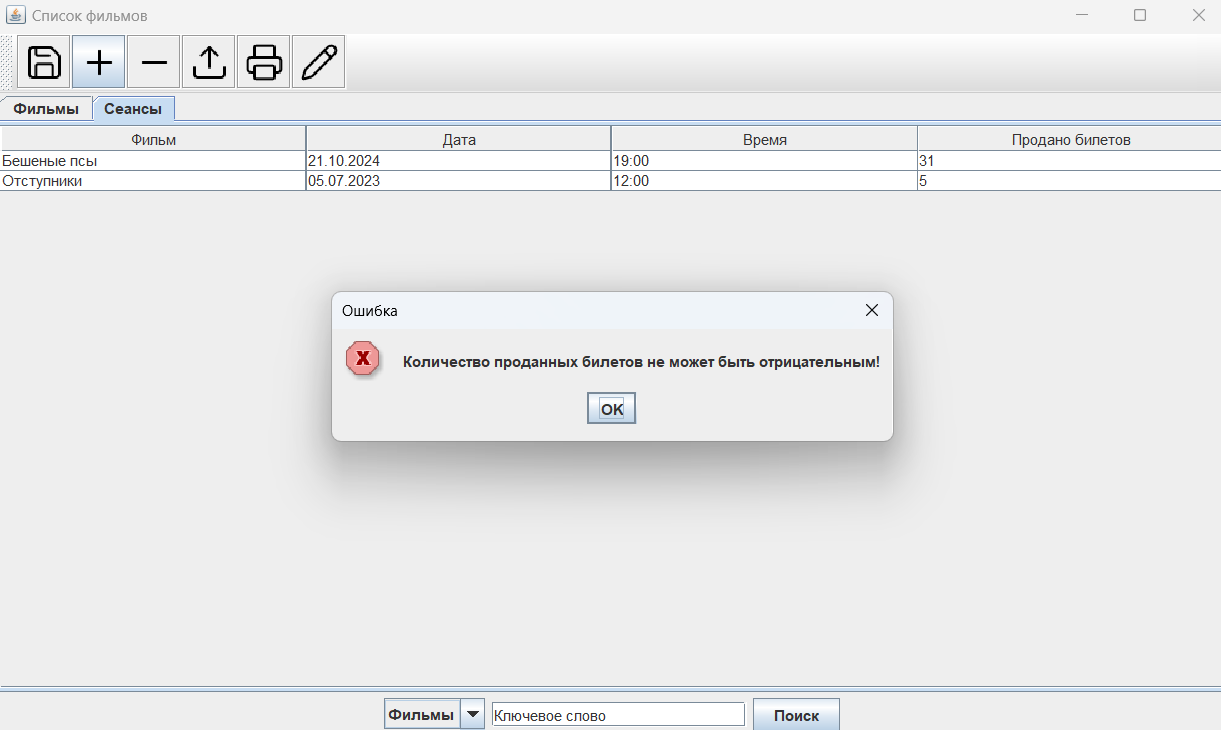
*Пример 4.2*





*Пример 4.3*

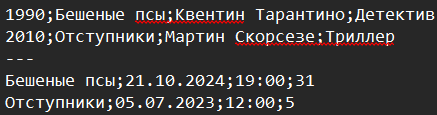




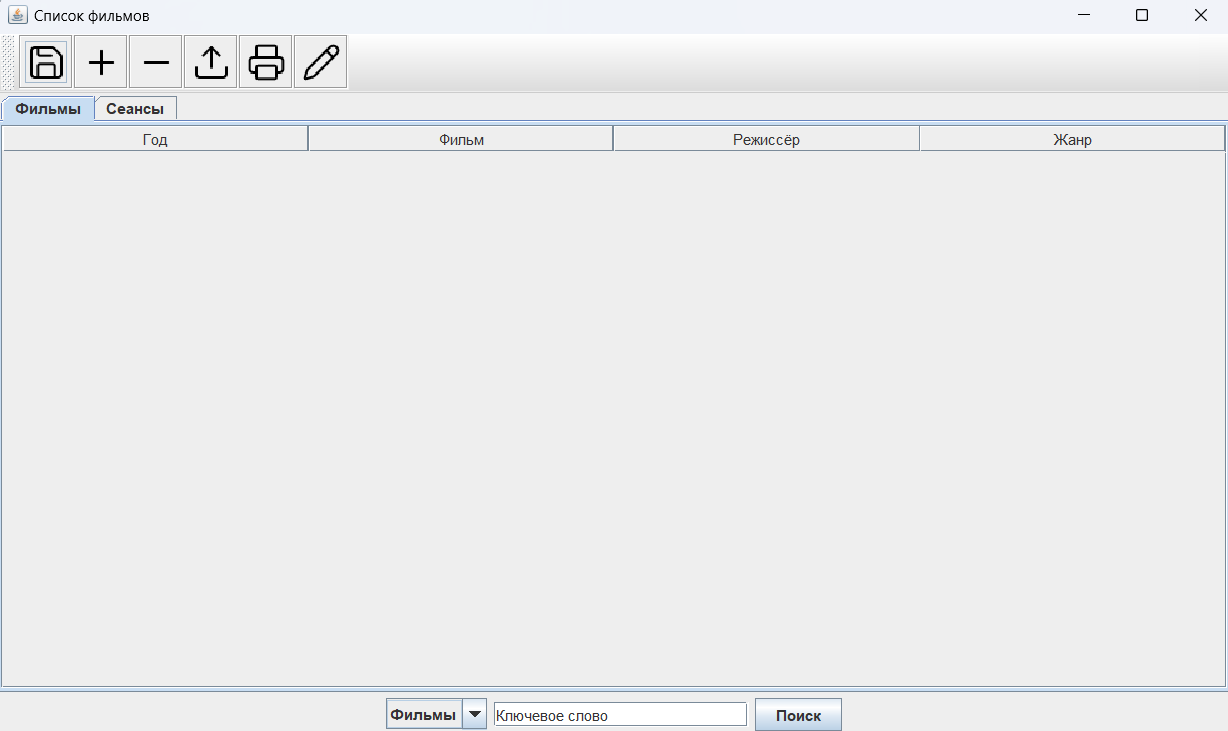
*Пример 4.4*

**Распечатки содержимого файлов с данными до и после внесения изменений**

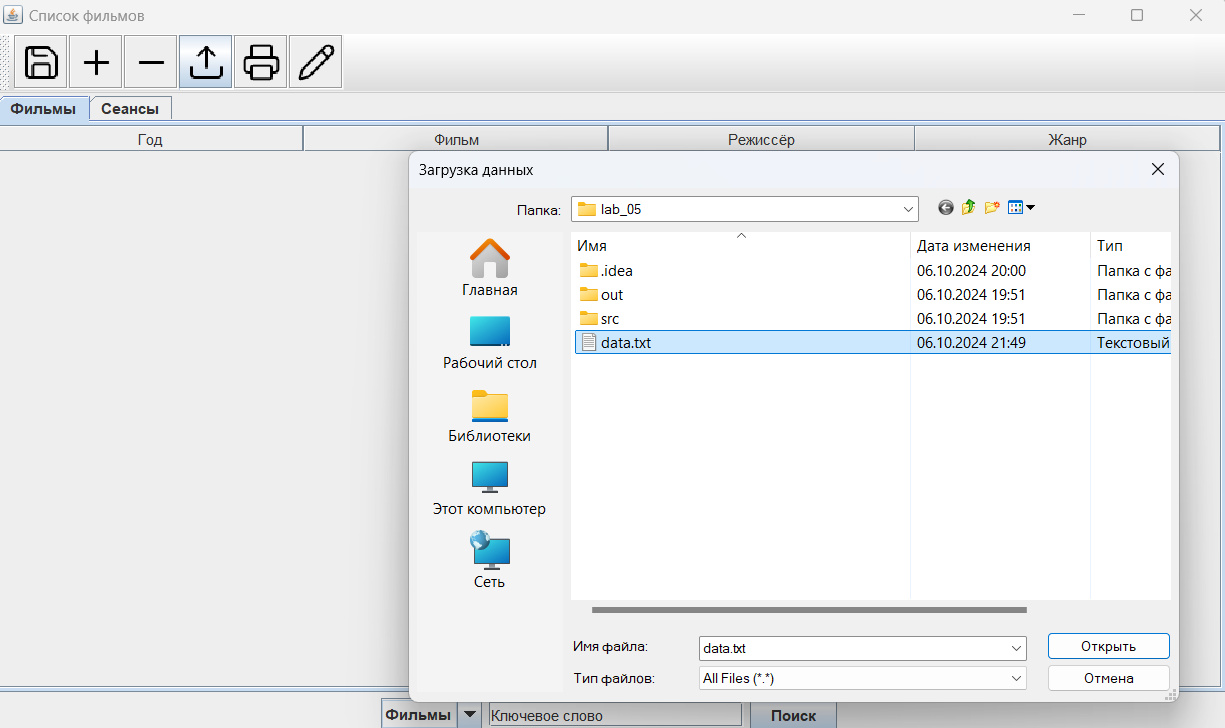
Процесс загрузки файла, внесения в него изменения и сохранения файла показан ниже. Также продемонстрирован исходный файл с данными до и после работы программы.



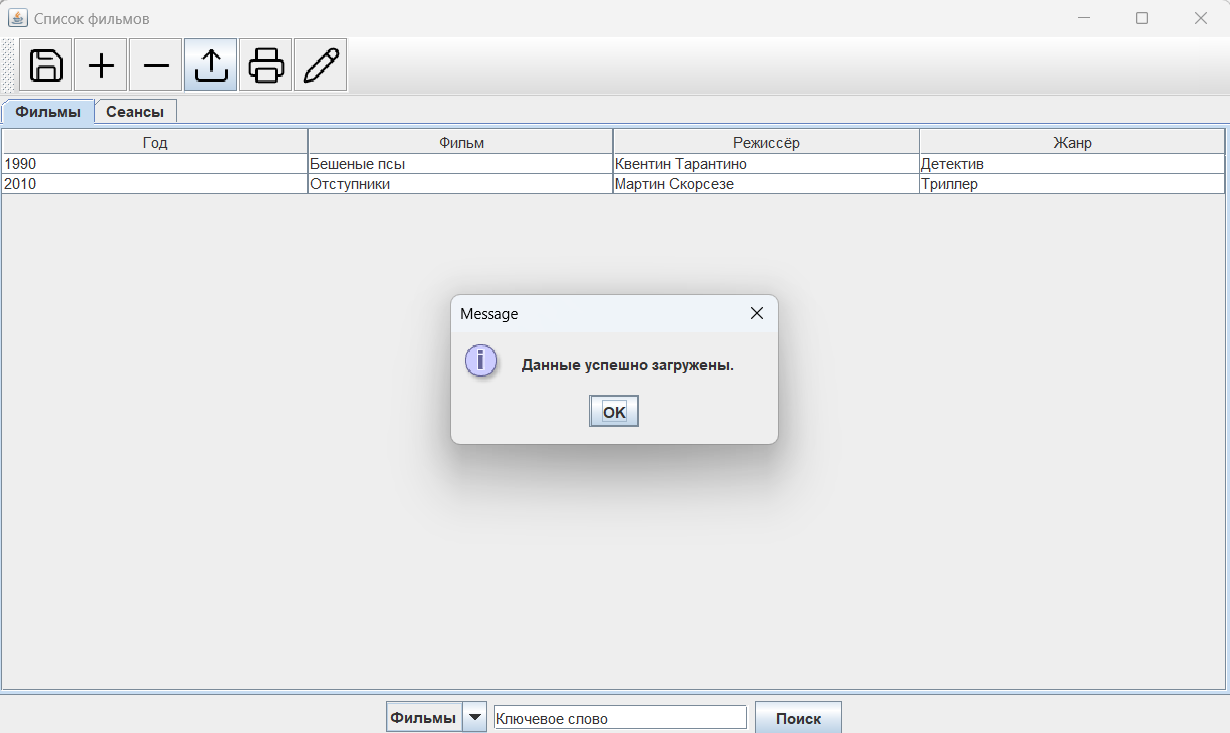
Файл data.txt до работы программы



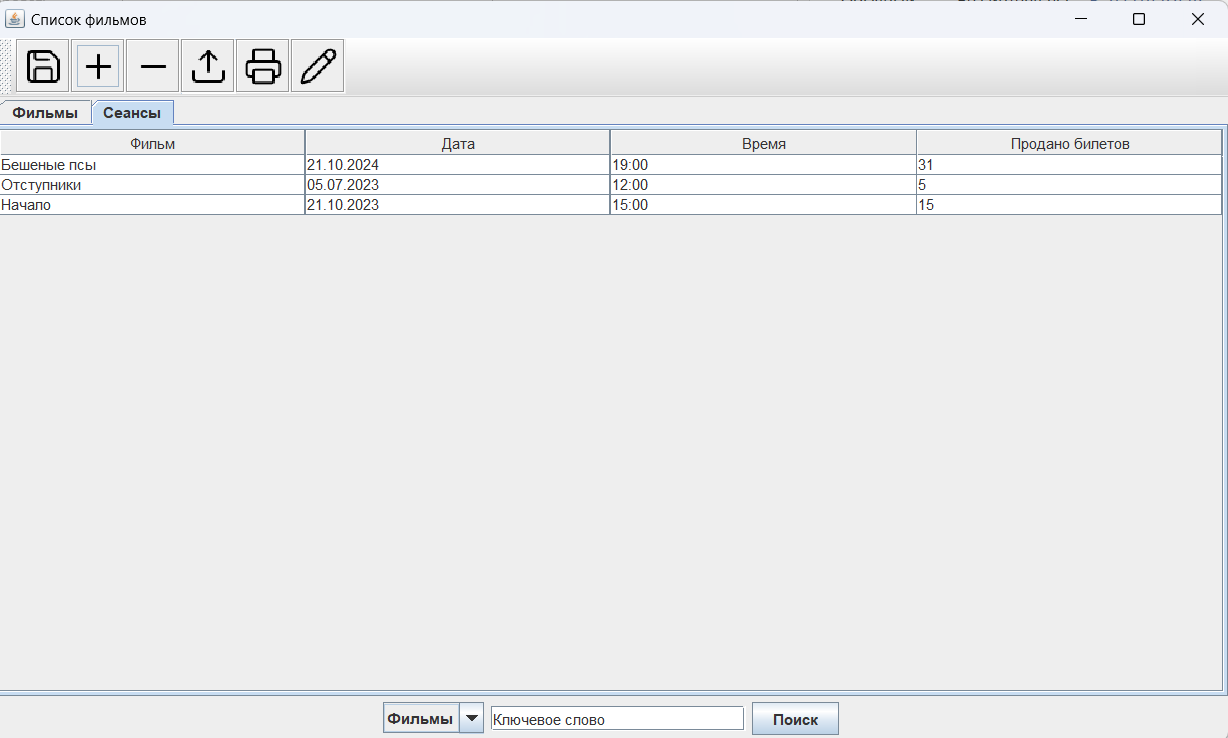
Программа без загрузки данных



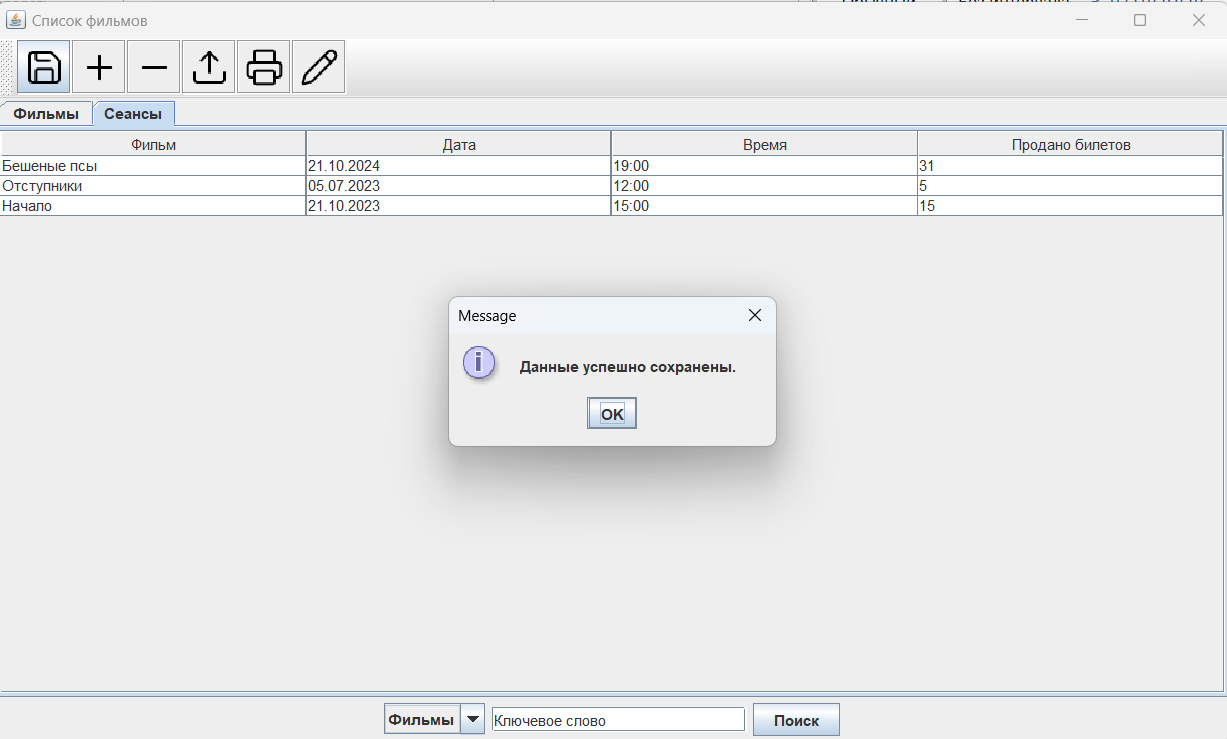
Выбор файла с данными



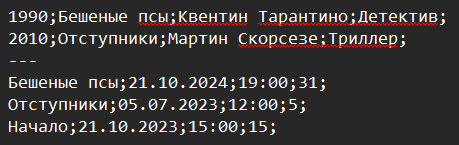
Данные успешно загружены



Внесены новые данные



Сохранение новых данных



Файл data.txt после работы программы

**Текст программы**

package edu.java.lab05;  
  
// Подключение графических библиотек  
import java.awt.\*;  
import javax.swing.\*;  
import javax.swing.table.DefaultTableModel;  
import java.awt.event.\*;  
import java.io.BufferedWriter;  
import java.io.FileWriter;  
import java.io.IOException;  
import java.io.FileNotFoundException;  
import java.io.BufferedReader;  
import java.io.FileReader;  
  
  
public class CinemaList {  
 // Объявления графических компонентов  
 private JFrame cinemaList;  
 private DefaultTableModel modelMovies, modelSessions;  
 private JButton save, add, delete, upload, print, edit;  
 private JToolBar toolBar;  
 private JScrollPane scrollMovies, scrollSessions;  
 private JTable tableMovies, tableSessions;  
 private JComboBox director;  
 private JTextField wordToFind;  
 private JButton filter;  
  
 public void show() {  
 // Создание окна  
 cinemaList = new JFrame("Список фильмов"); // Название приложения  
 cinemaList.setSize(1000, 600); // Ширина и высота окна  
 cinemaList.setLocation(100, 100); // Начальное положение  
 cinemaList.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 cinemaList.addWindowListener(new WindowAdapter() { // Слушатель на закрытие окна с подтвердждением выхода  
 public void windowClosing(WindowEvent e) {  
 int confirm = JOptionPane.*showConfirmDialog*(cinemaList, "Вы уверены, что хотите выйти?");  
 if (confirm == JOptionPane.*YES\_OPTION*) {  
 cinemaList.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 } else {  
 cinemaList.setDefaultCloseOperation(JFrame.*DO\_NOTHING\_ON\_CLOSE*);  
 }  
 }  
 });  
  
 toolBar = new JToolBar("Панель инструментов"); // Создание панели инструментов  
 cinemaList.setLayout(new BorderLayout()); // Размещение панели инструментов  
 cinemaList.add(toolBar, BorderLayout.*NORTH*); // Начальное положение панели инструментов  
  
 // Создание кнопок, прикрепление иконок, настройка подсказок и добавление кнопок на панель инструментов  
 save = new JButton(new ImageIcon("src/img/save.png"));  
 save.setToolTipText("Сохранить данные");  
 toolBar.add(save);  
 save.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 saveToFile();  
 }  
 });  
  
 add = new JButton(new ImageIcon("src/img/add.png"));  
 add.setToolTipText("Добавить данные");  
 toolBar.add(add);  
  
 delete = new JButton(new ImageIcon("src/img/delete.png"));  
 delete.setToolTipText("Удалить данные");  
 toolBar.add(delete);  
  
 upload = new JButton(new ImageIcon("src/img/upload.png"));  
 upload.setToolTipText("Загрузить данные");  
 toolBar.add(upload);  
 upload.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 loadFromFile();  
 }  
 });  
  
  
  
 print = new JButton(new ImageIcon("src/img/print.png"));  
 print.setToolTipText("Распечатать");  
 toolBar.add(print);  
  
 edit = new JButton(new ImageIcon("src/img/edit.png"));  
 edit.setToolTipText("Редактировать данные");  
 toolBar.add(edit);  
  
 // Создание таблицы для фильмов  
 String [] columnsMovies = {"Год", "Фильм", "Режиссёр", "Жанр"};  
 String [][] dataMovies = {};  
 modelMovies= new DefaultTableModel(dataMovies, columnsMovies);  
 tableMovies = new JTable(modelMovies);  
 scrollMovies = new JScrollPane(tableMovies);  
  
 // Создание таблицы для сеансов  
 String [] columnsSessions = {"Фильм", "Дата", "Время", "Продано билетов"};  
 String [][] dataSessions = {};  
 modelSessions= new DefaultTableModel(dataSessions, columnsSessions);  
 tableSessions = new JTable(modelSessions);  
 scrollSessions = new JScrollPane(tableSessions);  
  
 // Создание вкладок с таблицами  
 JTabbedPane tabbedPane = new JTabbedPane();  
 tabbedPane.addTab("Фильмы", scrollMovies);  
 tabbedPane.addTab("Сеансы", scrollSessions);  
 cinemaList.add(tabbedPane, BorderLayout.*CENTER*); // Размещение таблиц  
  
 // Подготовка компонентов поиска  
 director = new JComboBox(new String[]{"Фильмы", "Сеансы"});  
  
 wordToFind = new JTextField("Ключевое слово", 20);  
 wordToFind.addFocusListener(new FocusAdapter() {  
 public void focusGained(FocusEvent e) {  
 if (wordToFind.getText().equals("Ключевое слово")) {  
 wordToFind.setText(""); // Очистить поле при получении фокуса  
 }  
 }  
 public void focusLost(FocusEvent e) {  
 if (wordToFind.getText().isEmpty()) {  
 wordToFind.setText("Ключевое слово"); // Вернуть текст, если поле пустое  
 }  
 }  
 });  
  
 filter = new JButton("Поиск");  
 filter.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent arg0) { // Обработка исключения - пустая строка  
 try { checkName(wordToFind);  
 }  
 catch(NullPointerException ex) {  
 JOptionPane.*showMessageDialog*(cinemaList, ex.toString());  
 }  
 catch(InvalidFindException myEx) {  
 JOptionPane.*showMessageDialog*(null, myEx.getMessage());  
 }}});  
  
 // Добавление компонентов на панель  
 JPanel filterPanel = new JPanel();  
 filterPanel.add(director); // Добавление на панель поиска  
 filterPanel.add(wordToFind);  
 filterPanel.add(filter);  
 // Размещение панели поиска внизу окна  
 cinemaList.add(filterPanel, BorderLayout.*SOUTH*);  
  
 add.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 if (tabbedPane.getSelectedIndex() == 0) { // Если выбрана вкладка "Фильмы"  
 addMovie();  
 } else if (tabbedPane.getSelectedIndex() == 1) { // Если выбрана вкладка "Сеансы"  
 addSession();  
 }  
 }  
 });  
 delete.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 int selectedTab = tabbedPane.getSelectedIndex(); // Определяем, какая вкладка выбрана  
  
 if (selectedTab == 0) { // Если выбрана вкладка "Фильмы"  
 int selectedRow = tableMovies.getSelectedRow();  
 if (selectedRow != -1) { // Проверяем, выбрана ли строка  
 modelMovies.removeRow(selectedRow); // Удаляем выбранную строку из таблицы фильмов  
 } else {  
 JOptionPane.*showMessageDialog*(cinemaList, "Пожалуйста, выберите фильм для удаления.");  
 }  
 } else if (selectedTab == 1) { // Если выбрана вкладка "Сеансы"  
 int selectedRow = tableSessions.getSelectedRow();  
 if (selectedRow != -1) { // Проверяем, выбрана ли строка  
 modelSessions.removeRow(selectedRow); // Удаляем выбранную строку из таблицы сеансов  
 } else {  
 JOptionPane.*showMessageDialog*(cinemaList, "Пожалуйста, выберите сеанс для удаления.");  
 }  
 }  
 }  
 });  
  
  
 // Визуализация экранной формы  
 cinemaList.setVisible(true);  
 }  
  
 private class InvalidFindException extends Exception { // Исключение для неверного ввода поиска  
 public InvalidFindException(String message) {  
 super (message);  
 }}  
 private void checkName (JTextField bName) throws InvalidFindException,NullPointerException { // Метод проверки поля поиска  
 String sName = bName.getText();  
 if (sName.contains("Ключевое слово")) throw new InvalidFindException("Вы не ввели слова для поиска");  
 if (sName.isEmpty()) throw new NullPointerException();  
 }  
  
 class InvalidDateException extends Exception { // Исключение для неверного формата даты  
 public InvalidDateException(String message) {  
 super(message);  
 }  
 }  
 private void checkDate(String date) throws InvalidDateException { // Метод проверки формата даты  
 if (!date.matches("\\d{2}\\.\\d{2}\\.\\d{4}")) {  
 throw new InvalidDateException("Неверный формат даты! Ожидается формат: дд.мм.гггг.");  
 }  
 }  
  
 class InvalidTimeException extends Exception { // Исключение для неверного формата времени  
 public InvalidTimeException(String message) {  
 super(message);  
 }  
 }  
 private void checkTime(String time) throws InvalidTimeException { // Метод проверки формата времени  
 if (!time.matches("\\d{2}:\\d{2}")) {  
 throw new InvalidTimeException("Неверный формат времени! Ожидается формат: чч:мм.");  
 }  
 }  
  
 class InvalidTicketCountException extends Exception { // Исключение для неверного формата количества билетов  
 public InvalidTicketCountException(String message) {  
 super(message);  
 }  
 }  
 private void checkTicketCount(String ticketsSold) throws InvalidTicketCountException, NumberFormatException { // Метод проверки количества проданных билетов  
 try {  
 int count = Integer.*parseInt*(ticketsSold);  
 if (count < 0) {  
 throw new InvalidTicketCountException("Количество проданных билетов не может быть отрицательным!");  
 }  
 } catch (NumberFormatException ex) {  
 throw new InvalidTicketCountException("Количество проданных билетов должно быть числом!");  
 }  
 }  
  
 // Метод для добавления новой строки в таблицу фильмов  
 private void addMovie() {  
 // Создаем панель для ввода данных  
 JPanel inputPanel = new JPanel(new GridLayout(4, 2));  
 JTextField yearField = new JTextField();  
 JTextField movieField = new JTextField();  
 JTextField directorField = new JTextField();  
 JTextField genreField = new JTextField();  
  
 inputPanel.add(new JLabel("Год:"));  
 inputPanel.add(yearField);  
 inputPanel.add(new JLabel("Фильм:"));  
 inputPanel.add(movieField);  
 inputPanel.add(new JLabel("Режиссер:"));  
 inputPanel.add(directorField);  
 inputPanel.add(new JLabel("Жанр:"));  
 inputPanel.add(genreField);  
 boolean flag = false;  
  
 while(!flag){  
 // Открываем диалоговое окно для ввода данных  
 int result = JOptionPane.*showConfirmDialog*(cinemaList, inputPanel,  
 "Добавить новый фильм", JOptionPane.*OK\_CANCEL\_OPTION*, JOptionPane.*PLAIN\_MESSAGE*);  
  
 // Если пользователь нажал "OK", то добавляем строку в таблицу  
 if (result == JOptionPane.*OK\_OPTION*) {  
 // Проверка на пустые поля (можно добавить исключение здесь)  
 if (yearField.getText().isEmpty() || movieField.getText().isEmpty() ||  
 directorField.getText().isEmpty() || genreField.getText().isEmpty()) {  
 JOptionPane.*showMessageDialog*(cinemaList, "Все поля должны быть заполнены",  
 "Ошибка", JOptionPane.*ERROR\_MESSAGE*);  
 } else {  
 // Добавление строки в таблицу фильмов  
 modelMovies.addRow(new Object[]{yearField.getText(), movieField.getText(),  
 directorField.getText(), genreField.getText()});  
 flag = true;  
 }  
 } else {  
 flag = true;  
 }  
 }  
 }  
  
 // Метод для добавления новой строки в таблицу сеансов  
 private void addSession() {  
 // Создаем панель для ввода данных  
 JPanel inputPanel = new JPanel(new GridLayout(4, 2));  
 JTextField movieField = new JTextField();  
 JTextField dateField = new JTextField();  
 JTextField timeField = new JTextField();  
 JTextField ticketsSoldField = new JTextField();  
  
 inputPanel.add(new JLabel("Фильм:"));  
 inputPanel.add(movieField);  
 inputPanel.add(new JLabel("Дата:"));  
 inputPanel.add(dateField);  
 inputPanel.add(new JLabel("Время:"));  
 inputPanel.add(timeField);  
 inputPanel.add(new JLabel("Продано билетов:"));  
 inputPanel.add(ticketsSoldField);  
 boolean flag = false;  
  
 // Открываем диалоговое окно для ввода данных  
 while (!flag) {  
 int result = JOptionPane.*showConfirmDialog*(cinemaList, inputPanel,  
 "Добавить новый сеанс", JOptionPane.*OK\_CANCEL\_OPTION*, JOptionPane.*PLAIN\_MESSAGE*);  
  
 // Если пользователь нажал "OK", то добавляем строку в таблицу  
 if (result == JOptionPane.*OK\_OPTION*) {  
 if (dateField.getText().isEmpty() || movieField.getText().isEmpty() ||  
 timeField.getText().isEmpty() || ticketsSoldField.getText().isEmpty()) {  
 JOptionPane.*showMessageDialog*(cinemaList, "Все поля должны быть заполнены",  
 "Ошибка", JOptionPane.*ERROR\_MESSAGE*);  
 } else {  
 try {  
 // Проверяем формат даты  
 checkDate(dateField.getText());  
 // Проверяем формат времени  
 checkTime(timeField.getText());  
 // Проверяем формат количества билетов  
 checkTicketCount(ticketsSoldField.getText());  
  
 // Если все проверки пройдены, добавляем строку в таблицу сеансов  
 modelSessions.addRow(new Object[]{  
 movieField.getText(),  
 dateField.getText(),  
 timeField.getText(),  
 ticketsSoldField.getText()  
 });  
 flag = true; // Выходим из цикла  
  
 } catch (InvalidDateException | InvalidTimeException | InvalidTicketCountException ex) {  
 // Выводим сообщение об ошибке  
 JOptionPane.*showMessageDialog*(cinemaList, ex.getMessage(),  
 "Ошибка", JOptionPane.*ERROR\_MESSAGE*);  
 }  
 }  
 } else {  
 flag = true; // Cancel option  
 }  
 }  
 }  
  
 private void saveToFile() {  
 // Открытие диалогового окна для выбора файла  
 FileDialog saveDialog = new FileDialog(cinemaList, "Сохранение данных", FileDialog.*SAVE*);  
 saveDialog.setFile("\*.txt");  
 saveDialog.setVisible(true);  
  
 String directory = saveDialog.getDirectory();  
 String filename = saveDialog.getFile();  
 if (directory == null || filename == null) return; // Пользователь нажал "Отмена"  
  
 String filePath = directory + filename;  
  
 try (BufferedWriter writer = new BufferedWriter(new FileWriter(filePath))) {  
 // Сохранение данных из таблицы фильмов  
 for (int i = 0; i < modelMovies.getRowCount(); i++) {  
 for (int j = 0; j < modelMovies.getColumnCount(); j++) {  
 writer.write(modelMovies.getValueAt(i, j).toString());  
 writer.write(";"); // Используем табуляцию как разделитель  
 }  
 writer.newLine(); // Переход на новую строку  
 }  
 // Добавляем разделитель между таблицами  
 writer.write("---");  
 writer.newLine();  
 // Сохранение данных из таблицы сеансов  
 for (int i = 0; i < modelSessions.getRowCount(); i++) {  
 for (int j = 0; j < modelSessions.getColumnCount(); j++) {  
 writer.write(modelSessions.getValueAt(i, j).toString());  
 writer.write(";");  
 }  
 writer.newLine();  
 }  
 JOptionPane.*showMessageDialog*(cinemaList, "Данные успешно сохранены.");  
 } catch (IOException ex) {  
 ex.printStackTrace();  
 JOptionPane.*showMessageDialog*(cinemaList, "Ошибка при сохранении файла.");  
 }  
 }  
  
 private void loadFromFile() {  
 FileDialog loadDialog = new FileDialog(cinemaList, "Загрузка данных", FileDialog.*LOAD*);  
 loadDialog.setFile("\*.txt");  
 loadDialog.setVisible(true);  
  
 String directory = loadDialog.getDirectory();  
 String filename = loadDialog.getFile();  
 if (directory == null || filename == null) return; // Пользователь нажал "Отмена"  
  
 String filePath = directory + filename;  
  
 try (BufferedReader reader = new BufferedReader(new FileReader(filePath))) {  
 // Очистка таблиц перед загрузкой  
 modelMovies.setRowCount(0);  
 modelSessions.setRowCount(0);  
  
 String line;  
 boolean isSessionsTable = false;  
 while ((line = reader.readLine()) != null) {  
 if (line.equals("---")) {  
 isSessionsTable = true; // Переходим к таблице сеансов  
 continue;  
 }  
 String[] data = line.split(";");  
 if (!isSessionsTable) {  
 modelMovies.addRow(data);  
 } else {  
 modelSessions.addRow(data);  
 }  
 }  
 JOptionPane.*showMessageDialog*(cinemaList, "Данные успешно загружены.");  
 } catch (FileNotFoundException ex) {  
 ex.printStackTrace();  
 JOptionPane.*showMessageDialog*(cinemaList, "Файл не найден.");  
 } catch (IOException ex) {  
 ex.printStackTrace();  
 JOptionPane.*showMessageDialog*(cinemaList, "Ошибка при чтении файла.");  
 }  
 }  
  
 public static void main(String[] args) {  
 // Создание и отображение экранной формы  
 new CinemaList().show();  
 }  
}