

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 № 4**  
**«Обработка исключений»**  
**по дисциплине «Объектно - ориентированное**  
**программирование на языке Java»**

Выполнил Сапронов К.Д.

Факультет КТИ

Группа № 3311

Подпись преподавателя \_\_\_\_\_

Санкт-Петербург

2024 г

## Цель работы

Знакомство с механизмом обработки исключений в языке Java.

## Описание задания

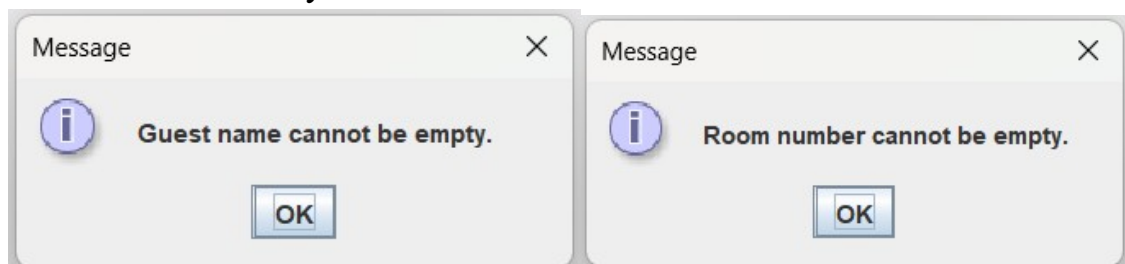
1. Проанализировать методы приложения и определить ошибочные ситуации, которые необходимо в них контролировать.
2. Написать 1-2 класса собственных исключений.
3. Задать спецификацию исключений и разработать код методов, где генерируются исключительные ситуации.
4. Разработать код методов, где контролируются и обрабатываются исключительные ситуации.

## Исключительные ситуации и их обработка.

Для обработки исключительных ситуаций был создан класс собственных исключений `EmptyFieldException`.

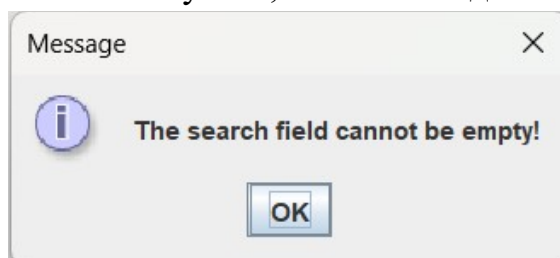
### 1. Добавление нового гостя в таблицу

При добавлении нового гостя пользователь может оставить поле для ввода новых данных пустым. Для обработки этой ситуации необходимо проверить все поля на то, пустые ли они, и если да, то вывести сообщение о неверном вводе в соответствующем поле.



### 2. Поиск

Поле для поиска аналогично проверяется на наличие введенных данных. Если оно пустое, также выводится диалоговое окно с сообщением.



## Ссылки

[https://drive.google.com/drive/folders/1SkmitiMaArA7aWjd8Q5cThXVNP8ws\\_N4?usp=drive\\_link](https://drive.google.com/drive/folders/1SkmitiMaArA7aWjd8Q5cThXVNP8ws_N4?usp=drive_link)

В этой папке будут находиться все лабораторные работы

В папке lab4 находятся этот отчет, видеоотчет и папка lab04, в которой находятся файлы проекта и документация javadoc.

## Текст программы

```
package lab04;

import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Comparator;

/**
 * Custom exception that is thrown when an input field is empty.
 */
class EmptyFieldException extends Exception {
    /**
     * Constructs an EmptyFieldException with the specified error message.
     *
     * @param message the error message for the exception
     */
    public EmptyFieldException(String message) {
        super(message);
    }
}

/**
 * This class creates the GUI for the Hotel Guest List application.
 * It builds the interface with menus, toolbar, search panel, and table to display guest data.
 */
class GUI {
    private JFrame frame;
    private JMenuBar menuBar;
    private JMenu fileMenu, sortMenu;
    private JMenuItem openItem, saveItem, roomItem, nameItem;
    private JToolBar toolBar;
    private JButton saveButton, addButton, deleteButton;
    private JButton searchButton;
    private JComboBox<String> searchType;
    private JTextField searchField;
    private JTable dataTable;
    private JScrollPane tableScrollPane;
    private DefaultTableModel tableModel;

    /**
     * This method sets up the GUI and displays it.
     * It creates the main window with menus, toolbar, search panel, and a table for displaying guest data.
     */
    public void buildAndShowGUI() {
```

```

// Create main frame (window) for the application
frame = new JFrame("Hotel - Guest List");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setSize(800, 600);

// Create menu bar and file menu with Open and Save items
menuBar = new JMenuBar();
fileMenu = new JMenu("File");
openItem = new JMenuItem("Open");
saveItem = new JMenuItem("Save");
fileMenu.add(openItem);
fileMenu.add(saveItem);
menuBar.add(fileMenu);

// Create sort menu with sorting options: Name and Room
sortMenu = new JMenu("Sort by");
roomItem = new JMenuItem("Room");
nameItem = new JMenuItem("Name");
sortMenu.add(nameItem);
sortMenu.add(roomItem);
menuBar.add(sortMenu);

// Set menu bar to the frame
frame.setJMenuBar(menuBar);

// Create toolbar with Save, Add, and Delete buttons
toolBar = new JToolBar();
saveButton = new JButton("Save");
addButton = new JButton("Add");
deleteButton = new JButton("Delete");
toolBar.add(saveButton);
toolBar.add(addButton);
toolBar.add(deleteButton);
frame.add(toolBar, BorderLayout.NORTH); // Add toolbar to the top (north) of the
frame

// Create a search panel with a combo box for selecting search type, text field,
and search button
JPanel searchPanel = new JPanel();
searchType = new JComboBox<>(new String[]{"Name", "Room", "Date"}); // Dropdown
for search criteria
searchField = new JTextField(15); // Text field for inputting search terms
searchButton = new JButton("Search");
searchPanel.add(new JLabel("Search by:"));
searchPanel.add(searchType);
searchPanel.add(searchField);
searchPanel.add(searchButton);
frame.add(searchPanel, BorderLayout.SOUTH); // Add search panel to the bottom
(south) of the frame

// Create table to display guest data with columns for Name, Room, Check-in Date,
and Check-out Date
String[] columns = {"Name", "Room", "Check-in Date", "Check-out Date"};
Object[][] data = {
    {"John Doe", "106", "07.10.2024", "21.10.2024"},
    {"Jane Doe", "228", "21.10.2024", "28.10.2024"}
};
tableModel = new DefaultTableModel(data, columns);
dataTable = new JTable(tableModel); // Initialize table with data and column
headers
tableScrollPane = new JScrollPane(dataTable); // Add scroll functionality to the
table
frame.add(tableScrollPane, BorderLayout.CENTER); // Add table in the center of
the frame

// Adding event listeners

```

```

        addListeners();

        // Make the frame visible
        frame.setVisible(true);
    }

    /**
     * Adds event listeners to the GUI components.
     * Listeners handle button clicks, menu item selections, and search actions.
     */
    private void addListeners() {
        // Listener for the Save button (stub)
        saveButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                // Example functionality: Display a message when Save button is clicked
                JOptionPane.showMessageDialog(frame, "Data saved successfully!");
            }
        });

        // Listener for the Add button
        addButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                try {
                    String name = JOptionPane.showInputDialog("Enter guest name:");
                    String room = JOptionPane.showInputDialog("Enter room number:");
                    String checkIn = JOptionPane.showInputDialog("Enter check-in date
(dd.mm.yyyy):");
                    String checkOut = JOptionPane.showInputDialog("Enter check-out date
(dd.mm.yyyy):");

                    // Call method to check for empty values
                    validateGuestInput(name, room, checkIn, checkOut);

                    // Add new guest if all checks are passed
                    tableModel.addRow(new Object[]{name, room, checkIn, checkOut});
                    JOptionPane.showMessageDialog(frame, "Guest added successfully!");
                } catch (EmptyFieldException ex) {
                    JOptionPane.showMessageDialog(frame, ex.getMessage());
                }
            }
        });

        // Listener for the Delete button
        deleteButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                int selectedRow = dataTable.getSelectedRow();
                int confirm = JOptionPane.showConfirmDialog(frame, "Are you sure you want
to delete the selected row?", "Delete", JOptionPane.YES_NO_OPTION);
                if (confirm == JOptionPane.YES_OPTION) {
                    // Remove the selected row from the table
                    tableModel.removeRow(selectedRow);
                }
            }
        });

        // Listener for the Search button
        searchButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                try {
                    checkSearchField();
                    // Perform search if search field isn't empty

```

```

        String searchTerm = searchField.getText();
        String searchCriteria = (String) searchType.getSelectedItem();
        // Iterate over table rows to find matching data
        for (int i = 0; i < dataTable.getRowCount(); i++) {
            String value = (String) dataTable.getValueAt(i,
searchType.getSelectedIndex());
            if (value.toLowerCase().contains(searchTerm.toLowerCase())) {
                // Highlight the row if it matches the search criteria
                dataTable.setRowSelectionInterval(i, i);
                JOptionPane.showMessageDialog(frame, "Match found: " +
value);
            }
            return;
        }
        JOptionPane.showMessageDialog(frame, "No matches found.");
    } catch (EmptyFieldException ex) {
        JOptionPane.showMessageDialog(frame, ex.getMessage());
    }
}

});

// Listener for the "Open" menu item
openItem.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        // Functionality: Load data from a file
        JOptionPane.showMessageDialog(frame, "File opened successfully!");
    }
});

// Listener for the "Sort by Name" menu item
nameItem.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        sortTable(0); // Sort by Name (first column)
    }
});

// Listener for the "Sort by Room" menu item
roomItem.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        sortTable(1); // Sort by Room (second column)
    }
});
}

/**
 * Sorts the table based on the specified column index.
 *
 * @param columnIndex the column index to sort by
 */
private void sortTable(int columnIndex) {
    // Get the row count and table data
    int rowCount = tableModel.getRowCount();
    Object[][] tableData = new Object[rowCount][tableModel.getColumnCount()];

    for (int i = 0; i < rowCount; i++) {
        for (int j = 0; j < tableModel.getColumnCount(); j++) {
            tableData[i][j] = tableModel.getValueAt(i, j);
        }
    }

    // Sort the data by the specified column
    java.util.Arrays.sort(tableData, Comparator.comparing(o ->

```

```

o[columnIndex].toString()));

        // Clear the table and re-add the sorted data
        tableModel.setRowCount(0);
        for (Object[] row : tableData) {
            tableModel.addRow(row);
        }

        JOptionPane.showMessageDialog(frame, "Table sorted by " +
tableModel.getColumnName(columnIndex));
    }

    /**
     * Checks if the search field is empty and throws an exception if it is.
     *
     * @throws EmptyFieldException if the search field is empty
     */
    private void checkSearchField() throws EmptyFieldException {
        String searchTerm = searchField.getText();
        if (searchTerm.isEmpty()) {
            throw new EmptyFieldException("The search field cannot be empty!");
        }
        // Logic to search through table rows
    }

    /**
     * Validates the guest input fields to ensure they are not empty.
     *
     * @param name      The name of the guest.
     * @param room      The room number.
     * @param checkIn   The check-in date.
     * @param checkOut  The check-out date.
     * @throws EmptyFieldException if any of the input fields are empty.
     */
    private void validateGuestInput(String name, String room, String checkIn, String
checkOut) throws EmptyFieldException {
        if (name == null || name.trim().isEmpty()) {
            throw new EmptyFieldException("Guest name cannot be empty.");
        }
        if (room == null || room.trim().isEmpty()) {
            throw new EmptyFieldException("Room number cannot be empty.");
        }
        if (checkIn == null || checkIn.trim().isEmpty()) {
            throw new EmptyFieldException("Check-in date cannot be empty.");
        }
        if (checkOut == null || checkOut.trim().isEmpty()) {
            throw new EmptyFieldException("Check-out date cannot be empty.");
        }
    }
}

    /**
     * This is the main class that starts the Hotel Guest List application.
     * It uses SwingUtilities.invokeLater to ensure thread safety when creating the GUI.
     */
    public class HotelGUI {
        public static void main(String[] args) {
            SwingUtilities.invokeLater(() -> new GUI().buildAndShowGUI());
        }
    };

```