**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 6

# по дисциплине «Введение в разработку программного обеспечения»

на тему: «Принципы и технологии создания дружественных

пользовательских интерфейсов в *Java*»

Выполнил: студент гр. ИТП-21

Бондаренко И.А.

Принял: преподаватель-стажёр

Башаримов Ю.С.

Гомель 2024

Цель работы: улучшение пользовательского опыта за счёт разработки удобного интерфейса и реализации функционала, который упрощает взаимодействие с приложением.

**Ход работы**

**Задание.** 1. Создать новую ветку для графического приложения в ранее созданном *git* репозитории.

2. Используя библиотеку *Java Swing*, создать графический интерфейс для разработанных ранее лабораторных работ 1 и 3.

3. Элементы интерфейса должны позволять пользователю создавать, удалять и обновлять элементы коллекции, созданного в лабораторной работе 5 класса-репозитория.

4. Создать и одобрить пул-реквест на слияние ветки с интерфейсом с главной вет кой.

5. Составить отчет о проделанной работе.

Результат создания ветки *graphic\_application* с помощью команды *git checkout -b graphic\_application* представлен на рисунке 1.



Рисунок 1 – Результат создания ветки *graphic\_application*

Результат использования библиотеки *Java Swing*, создавая графический интерфейс для разработанной ранее лабораторной работы 3 представлен на рисунке 2.

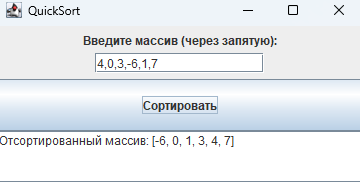


Рисунок 2 – Результат использования библиотеки *Java Swing*

для лабораторной работы 3

Результат использования библиотеки *Java Swing*, создавая графический интерфейс для разработанной ранее лабораторной работы 1 представлен на рисунке 3. Элементы интерфейса должны позволять пользователю создавать, удалять и обновлять элементы коллекции, созданного в лабораторной работе 5 класса-репозитория.

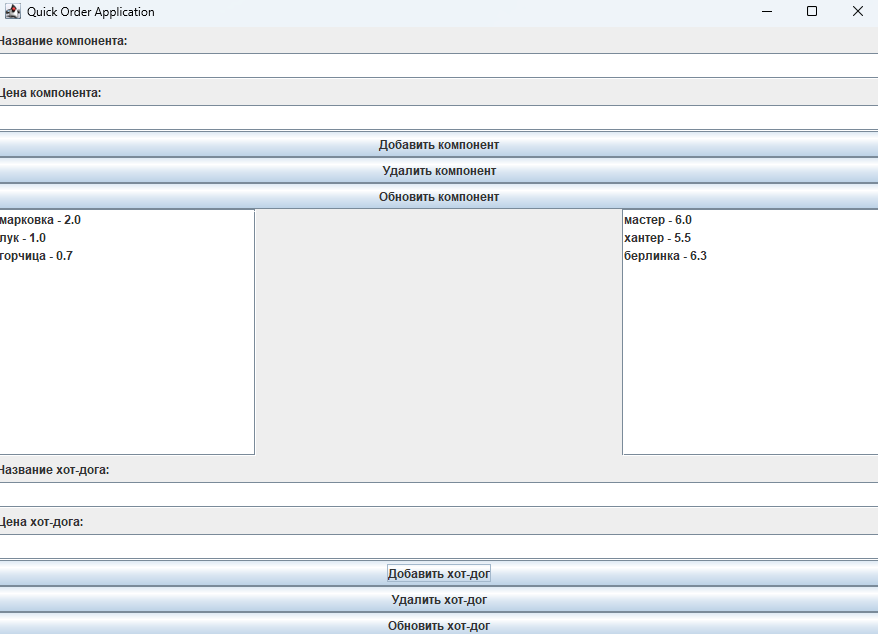
**

Рисунок 3 – Результат использования библиотеки *Java Swing*

для лабораторной работы 5

Результаты выполнения заданий использования библиотеки *Java Swing*, создать графический интерфейс для разработанных ранее лабораторных работ 1 и 3 представлены в приложении А и Б.

Результат запушивания ветки *graphic\_application* в удаленный репозиторий и её результат закоммичивания с помощью команд *git add ., git commit -m "Добавление графического приложения" и git push origin graphic\_application* представлен на рисунке 4.

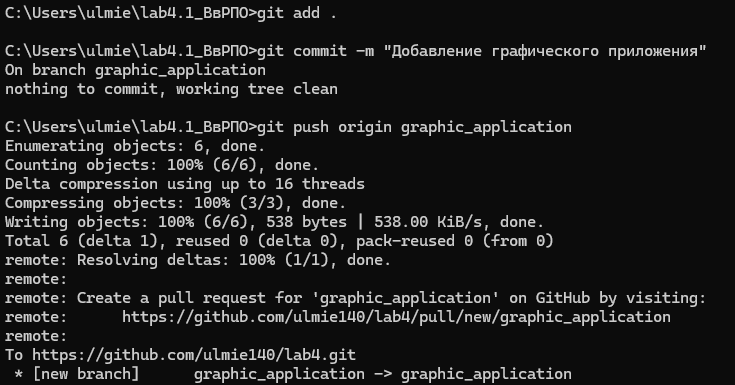


Рисунок 4 – Результат запушивания ветки *graphic\_application*

в удаленный репозиторий

Результат создания и одобрения пул-реквеста на слияние ветки *graphic\_application* с веткой *main* представлен на рисунке 5.



Рисунок 5 – Результат создания и одобрения пул-реквеста на слияние ветки *graphic\_application* с веткой *main*

Вывод: в ходе выполнения лабораторной работы мы улучшили пользовательский опыт за счёт разработки удобного интерфейса и реализации функционала, который упрощает взаимодействие с приложением. Современные инструменты и технологии, такие как системы контроля версий и инструменты для статического анализа, играют ключевую роль в поддержании высоких стандартов разработки. Внедрение этих практик способствует более эффективной командной работе и снижению технического долга.

**ПРИЛОЖЕНИЕ А**

(обязательное)

**Текст программы**

***Bun.java*:**

package by.bondar.component;  
  
*/\*\*  
 \* Класс для булки.  
 \*/*public class Bun extends Component {  
 public Bun() {  
 super("Булка", 1.0);  
 }  
}

***Cabbage.java*:**

package by.bondar.component;  
  
*/\*\*  
 \* Класс для капусты.  
 \*/*public class Cabbage extends Component {  
 public Cabbage() {  
 super("Капуста", 0.7);  
 }  
}

***Carrot.java*:**

package by.bondar.component;  
*/\*\*  
 \* Класс для морковки.  
 \*/*public class Carrot extends Component {  
 public Carrot() {  
 super("Марковка", 0.7);  
 }  
}

***Sausage.java*:**

package by.bondar.component;  
*/\*\*  
 \* Класс для сосиски.  
 \*/*public class Sausage extends Component {  
 public Sausage() {  
 super("Сосиска", 1.1);  
 }  
}

***Component.java*:**

package by.bondar.component;  
  
*/\*\*  
 \* Абстрактный класс для всех компонентов.  
 \*/*public abstract class Component {  
 protected String name;  
 protected double price;  
  
 public Component(String name, double price) {  
 this.name = name;  
 this.price = price;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public double getPrice() {  
 return price;  
 }  
}

***Berlinka.java*:**

package by.bondar.hotdog;  
  
*/\*\*  
 \* Класс Berlinka, наследующий HotDog.  
 \*/*public class Berlinka extends HotDog {  
 public Berlinka() {  
 super("Berlinka", 4.5);  
 }  
}

***HunterDog.java*:**

package by.bondar.hotdog;  
  
*/\*\*  
 \* Класс HunterDog, наследующий HotDog.  
 \*/*public class HunterDog extends HotDog {  
 public HunterDog() {  
 super("Hunter Dog", 5.0);  
 }  
}

***MasterDog.java*:**

package by.bondar.hotdog;  
  
*/\*\*  
 \* Класс MasterDog, наследующий HotDog.  
 \*/*public class MasterDog extends HotDog {  
 public MasterDog() {  
 super("Master Dog", 6.0);  
 }  
}

***HotDog.java*:**

package by.bondar.hotdog;  
  
*/\*\*  
 \* Абстрактный класс для всех видов хот-догов.  
 \*/*public abstract class HotDog {  
 protected String name;  
 protected double price;  
  
 public HotDog(String name, double price) {  
 this.name = name;  
 this.price = price;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public double getPrice() {  
 return price;  
 }  
}

***QuickOrderApp.java*:**

package by.bondar;  
  
import by.bondar.component.\*;  
import by.bondar.HotDog.\*;  
import by.bondar.component.Component;  
  
import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
  
public class QuickOrderApp extends JFrame {  
 private JList<Component> componentList;  
 private JList<HotDog> hotDogList;  
 private DefaultListModel<Component> componentListModel;  
 private DefaultListModel<HotDog> hotDogListModel;  
 private JTextField componentNameField, componentPriceField, hotDogNameField, hotDogPriceField;  
  
 public QuickOrderApp() {  
 setTitle("Quick Order Application");  
 setSize(900, 650); // Задаем размеры окна  
 setDefaultCloseOperation(*EXIT\_ON\_CLOSE*);  
 setLayout(new BorderLayout());  
  
 // Создаем модели списков  
 componentListModel = new DefaultListModel<>();  
 hotDogListModel = new DefaultListModel<>();  
  
 // Создаем списки  
 componentList = new JList<>(componentListModel);  
 hotDogList = new JList<>(hotDogListModel);  
  
 // Создаем панели для ввода компонентов  
 JPanel componentPanel = new JPanel(new GridLayout(0, 1));  
 componentNameField = new JTextField();  
 componentPriceField = new JTextField();  
 componentPanel.add(new JLabel("Название компонента:"));  
 componentPanel.add(componentNameField);  
 componentPanel.add(new JLabel("Цена компонента:"));  
 componentPanel.add(componentPriceField);  
 JButton addComponentButton = new JButton("Добавить компонент");  
 JButton removeComponentButton = new JButton("Удалить компонент");  
 JButton updateComponentButton = new JButton("Обновить компонент");  
 componentPanel.add(addComponentButton);  
 componentPanel.add(removeComponentButton);  
 componentPanel.add(updateComponentButton);  
  
 // Создаем панели для ввода хот-догов  
 JPanel hotDogPanel = new JPanel(new GridLayout(0, 1));  
 hotDogNameField = new JTextField();  
 hotDogPriceField = new JTextField();  
 hotDogPanel.add(new JLabel("Название хот-дога:"));  
 hotDogPanel.add(hotDogNameField);  
 hotDogPanel.add(new JLabel("Цена хот-дога:"));  
 hotDogPanel.add(hotDogPriceField);  
 JButton addHotDogButton = new JButton("Добавить хот-дог");  
 JButton removeHotDogButton = new JButton("Удалить хот-дог");  
 JButton updateHotDogButton = new JButton("Обновить хот-дог");  
 hotDogPanel.add(addHotDogButton);  
 hotDogPanel.add(removeHotDogButton);  
 hotDogPanel.add(updateHotDogButton);  
  
 // Добавляем списки и панели в главную форму  
 add(new JScrollPane(componentList), BorderLayout.*WEST*);  
 add(new JScrollPane(hotDogList), BorderLayout.*EAST*);  
 add(componentPanel, BorderLayout.*NORTH*);  
 add(hotDogPanel, BorderLayout.*SOUTH*);  
  
 // Обработчики событий  
 addComponentButton.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 addComponent();  
 }  
 });  
  
 removeComponentButton.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 removeComponent();  
 }  
 });  
  
 updateComponentButton.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 updateComponent();  
 }  
 });  
  
 addHotDogButton.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 addHotDog();  
 }  
 });  
  
 removeHotDogButton.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 removeHotDog();  
 }  
 });  
  
 updateHotDogButton.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 updateHotDog();  
 }  
 });  
 }  
  
 private void addComponent() {  
 String name = componentNameField.getText();  
 double price = Double.*parseDouble*(componentPriceField.getText());  
 Component component = new Component(name, price) {};  
 componentListModel.addElement(component);  
 componentNameField.setText("");  
 componentPriceField.setText("");  
 }  
  
 private void removeComponent() {  
 Component selected = componentList.getSelectedValue();  
 if (selected != null) {  
 componentListModel.removeElement(selected);  
 }  
 }  
  
 private void updateComponent() {  
 Component selected = componentList.getSelectedValue();  
 if (selected != null) {  
 String name = componentNameField.getText();  
 double price = Double.*parseDouble*(componentPriceField.getText());  
 Component updatedComponent = new Component(name, price) {};  
 componentListModel.setElementAt(updatedComponent, componentList.getSelectedIndex());  
 }  
 }  
  
 private void addHotDog() {  
 String name = hotDogNameField.getText();  
 double price = Double.*parseDouble*(hotDogPriceField.getText());  
 HotDog hotDog = new HotDog(name, price) {};  
 hotDogListModel.addElement(hotDog);  
 hotDogNameField.setText("");  
 hotDogPriceField.setText("");  
 }  
  
 private void removeHotDog() {  
 HotDog selected = hotDogList.getSelectedValue();  
 if (selected != null) {  
 hotDogListModel.removeElement(selected);  
 }  
 }  
  
 private void updateHotDog() {  
 HotDog selected = hotDogList.getSelectedValue();  
 if (selected != null) {  
 String name = hotDogNameField.getText();  
 double price = Double.*parseDouble*(hotDogPriceField.getText());  
 HotDog updatedHotDog = new HotDog(name, price) {};  
 hotDogListModel.setElementAt(updatedHotDog, hotDogList.getSelectedIndex());  
 }  
 }  
  
 public static void main(String[] args) {  
 SwingUtilities.*invokeLater*(() -> {  
 QuickOrderApp app = new QuickOrderApp();  
 app.setVisible(true);  
 });  
 }  
}

***Repo.java:***

package by.bondar;  
  
import java.util.List;  
  
public interface Repo<T> {  
 void add(T t);  
 void remove(T t);  
 void update(T oldT, T newT);  
 List<T> getAll();  
}

***ComponentRepo.java:***

package by.bondar;  
  
import by.bondar.component.Component;  
  
public interface ComponentRepo extends Repo<Component> {  
}

***HotDogRepo.java:***

package by.bondar;  
  
import by.bondar.HotDog.HotDog;  
  
public interface HotDogRepo extends Repo<HotDog> {  
}

***CompRepoImpl.java:***

package by.bondar;  
  
import by.bondar.component.Component;  
  
import java.util.ArrayList;  
import java.util.List;  
  
*/\*\*  
 \* Репозиторий для компонентов.  
 \*/*public class CompRepoImpl implements ComponentRepo {  
 private List<Component> components = new ArrayList<>();  
  
 @Override  
 public void add(Component component) {  
 components.add(component);  
 System.*out*.println("Добавлен компонент: " + component.getName());  
 }  
  
 @Override  
 public void remove(Component component) {  
 if (components.remove(component)) {  
 System.*out*.println("Удален компонент: " + component.getName());  
 } else {  
 System.*out*.println("Компонент не найден: " + component.getName());  
 }  
 }  
  
 @Override  
 public void update(Component oldComponent, Component newComponent) {  
 int index = components.indexOf(oldComponent);  
 if (index != -1) {  
 components.set(index, newComponent);  
 System.*out*.println("Изменен компонент: " + oldComponent.getName() + " на " + newComponent.getName());  
 } else {  
 System.*out*.println("Компонент не найден: " + oldComponent.getName());  
 }  
 }  
  
 @Override  
 public List<Component> getAll() {  
 return new ArrayList<>(components);  
 }  
}

***HotDogrepoImpl.java:***

package by.bondar;  
  
import by.bondar.HotDog.HotDog;  
  
import java.util.ArrayList;  
import java.util.List;  
  
*/\*\*  
 \* Репозиторий для хот-догов.  
 \*/*public class HotDogRepoImpl implements HotDogRepo {  
 private List<HotDog> hotDogs = new ArrayList<>();  
  
 @Override  
 public void add(HotDog hotDog) {  
 hotDogs.add(hotDog);  
 System.*out*.println("Добавлен хот-дог: " + hotDog.getName());  
 }  
  
 @Override  
 public void remove(HotDog hotDog) {  
 if (hotDogs.remove(hotDog)) {  
 System.*out*.println("Удален хот-дог: " + hotDog.getName());  
 } else {  
 System.*out*.println("Хот-дог не найден: " + hotDog.getName());  
 }  
 }  
  
 @Override  
 public void update(HotDog oldHotDog, HotDog newHotDog) {  
 int index = hotDogs.indexOf(oldHotDog);  
 if (index != -1) {  
 hotDogs.set(index, newHotDog);  
 System.*out*.println("Изменен хот-дог: " + oldHotDog.getName() + " на " + newHotDog.getName());  
 } else {  
 System.*out*.println("Хот-дог не найден: " + oldHotDog.getName());  
 }  
 }  
  
 @Override  
 public List<HotDog> getAll() {  
 return new ArrayList<>(hotDogs);  
 }  
}

**ПРИЛОЖЕНИЕ Б**

(обязательное)

**Текст программы**

***QuickSort.java*:**

package by.bondar;  
  
*/\*\*  
 \* Класс для реализации алгоритма быстрой сортировки.  
 \*/*public class QuickSort {  
  
 */\*\*  
 \* Метод для сортировки массива.  
 \*/* public void quickSort(int[] array, int low, int high) {  
 if (low < high) {  
 int pivotIndex = partition(array, low, high); // Разделение массива  
 quickSort(array, low, pivotIndex - 1); // Сортировка левой части  
 quickSort(array, pivotIndex + 1, high); // Сортировка правой части  
 }  
 }  
  
 */\*\*  
 \* Метод для разделения массива на две части.  
 \*/* private int partition(int[] array, int low, int high) {  
 int pivot = array[high]; // Опорный элемент  
 int i = low - 1; // Индекс меньшего элемента  
  
 for (int j = low; j < high; j++) {  
 if (array[j] <= pivot) { // Сравнение текущего элемента с опорным  
 i++; // Увеличиваем индекс меньшего элемента  
 swap(array, i, j); // Обмен элементов  
 }  
 }  
 swap(array, i + 1, high); // Перемещение опорного элемента в правильное место  
 return i + 1; // Возврат индекса опорного элемента  
 }  
  
 */\*\*  
 \* Метод для обмена элементов массива.  
 \*/* private void swap(int[] array, int i, int j) {  
 int temp = array[i];  
 array[i] = array[j];  
 array[j] = temp;  
 }  
}

***QoickSortSwingApp.java*:**

package by.bondar;  
  
import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.util.Arrays;  
  
public class QuickSortSwingApp extends JFrame {  
 private JTextField inputField;  
 private JTextArea resultArea;  
 private JButton sortButton;  
  
 public QuickSortSwingApp() {  
 setTitle("QuickSort");  
 setSize(380, 250); // Уменьшенный размер окна  
 setDefaultCloseOperation(*EXIT\_ON\_CLOSE*);  
 setLayout(new GridLayout(4, 1)); // Используем сетку для удобства  
  
 // Ввод массива  
 JPanel inputPanel = new JPanel();  
 inputPanel.add(new JLabel("Введите массив (через запятую):"));  
 inputField = new JTextField(15); // Уменьшенная ширина  
 inputPanel.add(inputField);  
 add(inputPanel);  
  
 // Кнопка сортировки  
 sortButton = new JButton("Сортировать");  
 add(sortButton);  
  
 // Область для отображения результатов  
 resultArea = new JTextArea();  
 resultArea.setEditable(false);  
 resultArea.setLineWrap(true);  
 resultArea.setWrapStyleWord(true);  
 add(new JScrollPane(resultArea)); // Добавляем прокрутку  
  
 // Обработчик события кнопки сортировки  
 sortButton.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 sortArray();  
 }  
 });  
 }  
  
 private void sortArray() {  
 String inputText = inputField.getText();  
 try {  
 // Преобразуем ввод в массив целых чисел  
 String[] stringArray = inputText.split(",");  
 int[] array = Arrays.*stream*(stringArray)  
 .map(String::trim)  
 .mapToInt(Integer::*parseInt*)  
 .toArray();  
  
 // Сортируем массив  
 QuickSort quickSort = new QuickSort();  
 quickSort.quickSort(array, 0, array.length - 1);  
  
 // Отображаем результат  
 resultArea.setText("Отсортированный массив: " + Arrays.*toString*(array));  
 } catch (NumberFormatException e) {  
 resultArea.setText("Ошибка ввода. Введите целые числа, разделенные запятыми.");  
 } catch (Exception e) {  
 resultArea.setText("Ошибка: " + e.getMessage());  
 }  
 }  
  
 public static void main(String[] args) {  
 SwingUtilities.*invokeLater*(() -> {  
 QuickSortSwingApp app = new QuickSortSwingApp();  
 app.setVisible(true);  
 });  
 }  
}