МИНИСТЕРСТВО ТРАНСПОРТА РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «РОССИЙСКИЙ УНИВЕРСИТЕТ ТРАНСПОРТА (МИИТ)»

Кафедра ЦТУТП

**Отчёт**

По лабораторной работе №3  
по дисциплине «Корпоративные информационные системы»

Тема: «События»

Вариант №7

Выполнил: Шедания В. М.

Группа: УИС-411

Преподаватель: доц. Кафедры ЦТУТП

Козьяков П. О.

­

Москва 2024 г.

**СОДЕРЖАНИЕ**

[ПОСТАНОВКА ЗАДАЧИ 3](#_Toc183377068)

[ХОД ВЫПОЛНЕНИЯ РАБОТЫ 4](#_Toc183377069)

[ДИАГРАММА ПОСЛЕДОВАТЕЛЬНОСТИ 6](#_Toc183377070)

[РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ 7](#_Toc183377071)

[КОД ПРОГРАММЫ 8](#_Toc183377072)

[ВЫВОД 11](#_Toc183377073)

# **ПОСТАНОВКА ЗАДАЧИ**

Создать систему взаимодействующих классов: класс для хранения и управления списком; класс, определяющий, является ли список симметричным; класс, определяющий сколько различных значений содержится в списке. Классы определения, должны срабатывать при каждом добавлении или удалении элементов. Между классами могут передаваться разные сообщения.

# **ХОД ВЫПОЛНЕНИЯ РАБОТЫ**

В ходе выполнения лабораторной работы была разработана система для управления коллекцией чисел, в которой реализованы механизмы добавления и удаления элементов с использованием событий. В начале работы были созданы две аннотации: @Added и @Removed, предназначенные для квалификации событий добавления и удаления элементов из коллекции. Эти аннотации позволяют отслеживать изменения в списке и взаимодействовать с другими компонентами системы.

Затем был реализован класс CollectionManager, который содержит список целых чисел и методы для добавления и удаления элементов. В этом классе происходит инъекция событий listAddedEvent и listRemovedEvent, которые вызываются при добавлении или удалении элементов из списка. При добавлении нового элемента в список метод addItem вызывает событие listAddedEvent, передавая текущий список, что позволяет другим компонентам реагировать на это изменение. Аналогично, метод removeItem вызывает событие listRemovedEvent, если элемент успешно удален. В обоих случаях выводится состояние списка, чтобы продемонстрировать изменения.

Далее был создан класс SymmetryChecker, который отслеживает изменения в списке и проверяет его симметричность. Этот класс использует метод isSymmetric для анализа списка, чтобы определить, является ли он симметричным. В зависимости от результата проверки, выводится соответствующее сообщение. Реакция на события добавления и удаления обеспечивается методами checkSymmetryOnAdd и checkSymmetryOnRemove, которые обрабатывают события, инициированные классом CollectionManager.

Также была реализована функциональность подсчета уникальных значений в списке через класс UniqueValueCounter. При добавлении или удалении элементов этот класс обновляет множество уникальных значений, сохраняя их в Set. Методы countUniqueOnAdd и countUniqueOnRemove отслеживают изменения в коллекции и выводят текущее количество уникальных значений после каждой операции.

Наконец, в классе Laboratory3 была организована основная логика работы программы. Используя контейнер CDI (Contexts and Dependency Injection), инициализируется экземпляр CollectionManager, через который происходят операции добавления и удаления элементов. Программа добавляет несколько чисел в список, включая дублирующиеся значения, и затем удаляет один из элементов, продемонстрировав тем самым как происходит взаимодействие компонентов и обработка событий. В завершение работы контейнер CDI корректно завершает свою работу, освобождая все ресурсы.

# **ДИАГРАММА ПОСЛЕДОВАТЕЛЬНОСТИ**

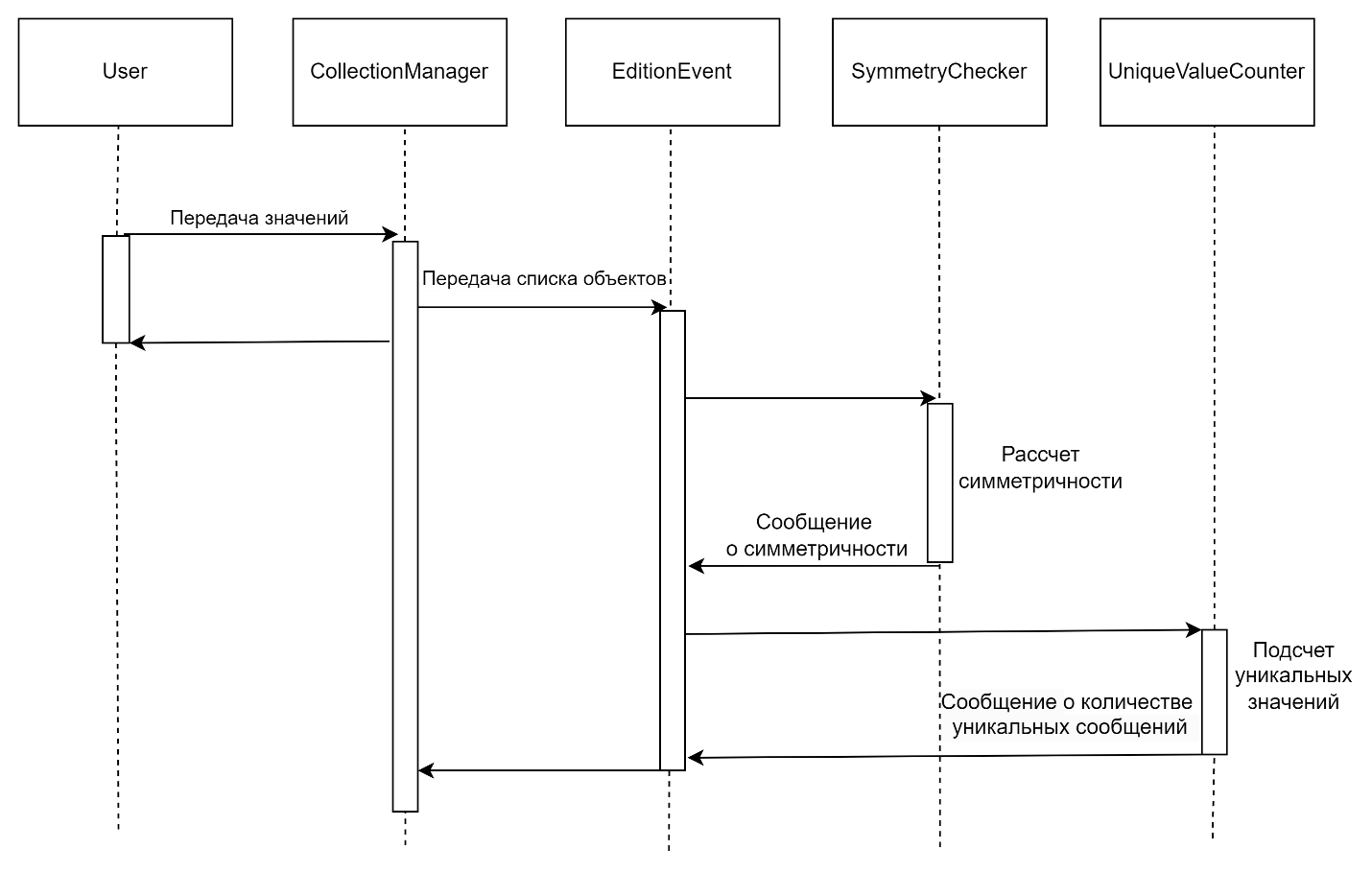


Рисунок 1 – Диаграмма последовательности

# **РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ**

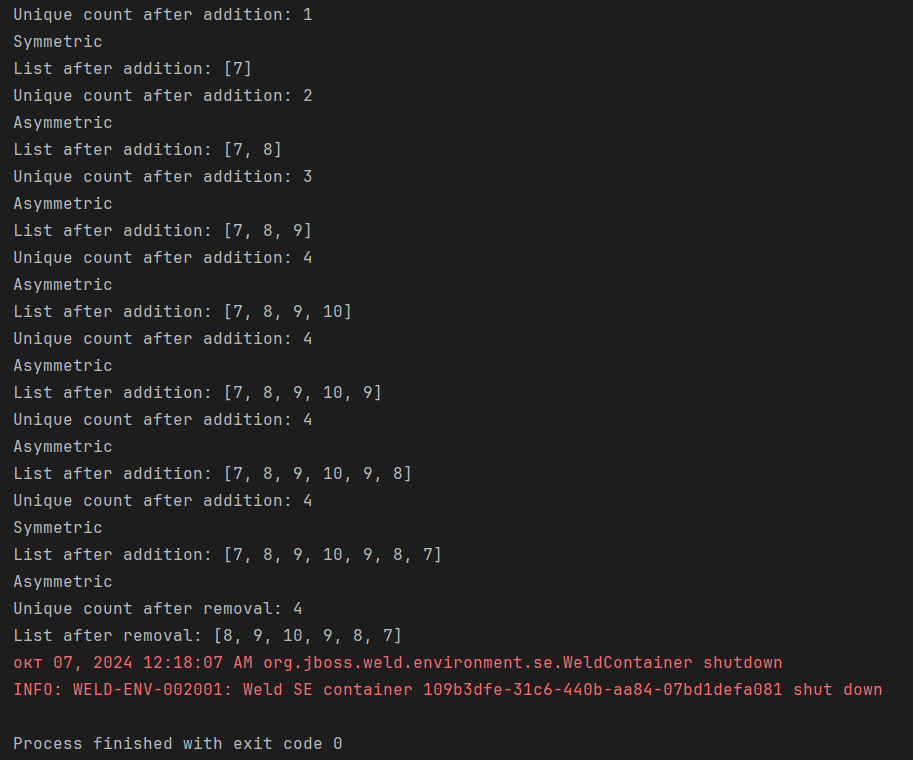


Рисунок 2 – Результат работы программы

# **КОД ПРОГРАММЫ**

package Lab3\_Event.Annotation;  
  
import javax.inject.Qualifier;  
import java.lang.annotation.ElementType;  
import java.lang.annotation.Retention;  
import java.lang.annotation.RetentionPolicy;  
import java.lang.annotation.Target;  
  
@Qualifier  
@Retention(RetentionPolicy.RUNTIME)  
@Target({ElementType.FIELD, ElementType.TYPE, ElementType.METHOD, ElementType.PARAMETER})  
public @interface Added {  
}

package Lab3\_Event.Annotation;  
  
import javax.inject.Qualifier;  
import java.lang.annotation.ElementType;  
import java.lang.annotation.Retention;  
import java.lang.annotation.RetentionPolicy;  
import java.lang.annotation.Target;  
  
@Qualifier  
@Retention(RetentionPolicy.RUNTIME)  
@Target({ElementType.FIELD, ElementType.TYPE, ElementType.METHOD, ElementType.PARAMETER})  
public @interface Removed {  
}

package Lab3\_Event.Util;  
  
import Lab3\_Event.Annotation.Added;  
import Lab3\_Event.Annotation.Removed;  
import lombok.Getter;  
  
import javax.enterprise.event.Event;  
import javax.inject.Inject;  
import java.util.ArrayList;  
import java.util.List;  
public class CollectionManager {  
 @Getter  
 private List<Integer> list = new ArrayList<>();  
  
 @Inject  
 @Added  
 private Event<List<Integer>> listAddedEvent;  
  
 @Inject  
 @Removed  
 private Event<List<Integer>> listRemovedEvent;  
  
 public void addItem(Integer item) {  
 list.add(item);  
 listAddedEvent.fire(new ArrayList<>(list));  
 System.out.println("List after addition: " + list); // Print the list after addition  
 }  
  
 public void removeItem(Integer item) {  
 if (list.contains(item)) {  
 list.remove(item);  
 listRemovedEvent.fire(new ArrayList<>(list));  
 System.out.println("List after removal: " + list); // Print the list after removal  
 } else {  
 System.out.println("Item " + item + " not found in the list."); // Optional: Notify if the item is not found  
 }  
 }  
  
}

package Lab3\_Event.Util;  
  
import Lab3\_Event.Annotation.Added;  
import Lab3\_Event.Annotation.Removed;  
  
import javax.enterprise.event.Observes;  
import java.util.List;  
  
public class SymmetryChecker {  
  
 // Method to handle the event when items are added  
 public void checkSymmetryOnAdd(@Observes @Added List<Integer> list) {  
 System.out.println(outputSymmetric(isSymmetric(list)));  
 }  
  
 // Method to handle the event when items are removed  
 public void checkSymmetryOnRemove(@Observes @Removed List<Integer> list) {  
 System.out.println(outputSymmetric(isSymmetric(list)));  
 }  
  
 public boolean isSymmetric(List<Integer> list) {  
 if (list == null || list.size() == 0) {  
 return true; // Empty list is symmetric  
 }  
 int n = list.size();  
 for (int i = 0; i < n / 2; i++) {  
 if (!list.get(i).equals(list.get(n - 1 - i))) {  
 return false; // Asymmetry  
 }  
 }  
 return true; // List is symmetric  
 }  
  
 public String outputSymmetric(boolean symmetric) {  
 return symmetric ? "Symmetric" : "Asymmetric";  
 }  
}  
package Lab3\_Event.Util;  
  
import Lab3\_Event.Annotation.Added;  
import Lab3\_Event.Annotation.Removed;  
  
import javax.enterprise.event.Observes;  
import java.util.HashSet;  
import java.util.List;  
import java.util.Set;  
  
public class UniqueValueCounter {  
  
 // Set to keep track of unique values  
 private final Set<Integer> uniqueValues = new HashSet<>();  
  
 // Method to handle the event when items are added  
 public void countUniqueOnAdd(@Observes @Added List<Integer> list) {  
 uniqueValues.addAll(list); // Add all items from the list to the set  
 System.out.println("Unique count after addition: " + uniqueValues.size());  
 }  
  
 // Method to handle the event when items are removed  
 public void countUniqueOnRemove(@Observes @Removed List<Integer> list) {  
 uniqueValues.addAll(list); // Add all items from the list to the set  
 System.out.println("Unique count after removal: " + uniqueValues.size());  
 }  
}  
package Lab3\_Event;  
  
import Lab3\_Event.Util.CollectionManager;  
import org.jboss.weld.environment.se.Weld;  
import org.jboss.weld.environment.se.WeldContainer;  
  
public class Laboratory3 {  
 protected static Weld weld;  
  
 protected static WeldContainer container;  
  
 public static void main(String args[]) throws Exception {  
 weld = new Weld();  
 container = weld.initialize();  
  
 try {  
 CollectionManager collectionManager = container.select(CollectionManager.class).get();  
  
 collectionManager.addItem(7);  
 collectionManager.addItem(8);  
 collectionManager.addItem(9);  
 collectionManager.addItem(10);  
 collectionManager.addItem(9);  
 collectionManager.addItem(8);  
 collectionManager.addItem(7);  
  
 collectionManager.removeItem(7); // Duplicate removal  
  
 } finally {  
 weld.shutdown();  
 }  
 }  
  
}

# **ВЫВОД**

В ходе лабораторной работы №3 была разработана система для управления коллекцией чисел, которая реагирует на добавление и удаление элементов с помощью событий. Для этого были использованы CDI-аннотации и события, что позволило компонентам системы взаимодействовать друг с другом.

Система включала три ключевых компонента: CollectionManager — класс, управляющий списком чисел и генерирующий события при добавлении или удалении элементов. SymmetryChecker — компонент, проверяющий симметричность списка при каждом изменении. UniqueValueCounter — класс для подсчета уникальных значений в списке.

Была реализована обработка событий, таких как добавление и удаление элементов, и продемонстрировано взаимодействие компонентов через публикацию и обработку событий. Лабораторная работа позволила понять, как реализовать взаимодействие между объектами с помощью событий и CDI, что способствует гибкости и модульности системы.