МИНИСТЕРСТВО ТРАНСПОРТА РФ

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

Кафедра ЦТУТП

**Отчёт**

По лабораторной работе №1  
по дисциплине «Корпоративные информационные системы»

Тема: «Реализация CDI-компонента»

Вариант №15

Выполнил: Дронов А.А.

Группа: УИС-411

Преподаватель: доц. Кафедры ЦТУТП

Козьяков П. О.

­

Москва 2024 г.

**СОДЕРЖАНИЕ**

[ПОСТАНОВКА ЗАДАЧИ 3](#_Toc180325056)

[ХОД ВЫПОЛНЕНИЯ РАБОТЫ 4](#_Toc180325057)

[РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ 4](#_Toc180325058)

[КОД ПРОГРАММЫ 7](#_Toc180325059)

[ВЫВОД 11](#_Toc180325060)

# **ПОСТАНОВКА ЗАДАЧИ**

Создать компонент CDI, выполняющий хранение данных в различных видах коллекций: связанный список, динамический массив, набор (set). Выбор способа хранения данных должен быть реализован через указание соответствующей аннотации или посредством указания класса в XML файле. Посредством перехватчиков провести логирование преобразования чисел.

# **ХОД ВЫПОЛНЕНИЯ РАБОТЫ**

В ходе выполнения лабораторной работы была разработана система управления коллекциями с использованием технологии CDI (Contexts and Dependency Injection) и механизма перехватчиков (Interceptors). Основная цель заключалась в реализации компонента, способного динамически выбирать тип хранилища на основе аннотаций, а также предоставлении подробного логирования действий с коллекциями.

Вначале были созданы пользовательские аннотации @ArrayStorage, @LinkedListStorage и @SetStorage для указания желаемого типа коллекции. Каждая аннотация была оформлена с использованием мета-аннотации @Qualifier, чтобы её можно было применять при внедрении CDI-компонентов. Эти аннотации стали основой для выбора типа хранилища.

После этого был разработан интерфейс StorageManager, представляющий основные операции для работы с коллекциями, такие как добавление элементов и получение всех хранимых объектов. Для каждой коллекции (динамического массива, связанного списка и множества) была создана своя реализация интерфейса. Эти классы содержали стандартные операции с соответствующими коллекциями Java (ArrayList, LinkedList и HashSet).

Для динамического выбора хранилища был создан компонент StorageComponent, который с помощью CDI-инъекции внедрял реализацию StorageManager, соответствующую указанной аннотации. Это позволяло гибко переключаться между различными типами коллекций без изменения основного кода. Кроме того, метод toString был переопределён в StorageComponent для отображения типа используемого хранилища и содержимого коллекции. Определение типа происходило через проверку аннотации, применённой к классу реализации.

Механизм перехватчиков был добавлен для логирования всех операций с коллекциями. Перехватчик StorageInterceptor регистрировал начало и завершение каждого метода, фиксируя параметры и результаты вызовов. Для улучшения читаемости логов использовалась динамическая проверка имени класса, чтобы исключить отображение CDI-прокси.

На этапе тестирования компонент StorageComponent был протестирован с каждым из трёх типов хранилищ. В консоль выводилась информация о выполнении операций, включая добавление элементов и получение всех хранимых данных. Логирование также демонстрировало, сколько времени занимало выполнение каждой операции, что позволяло анализировать производительность различных коллекций.

# **РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ**

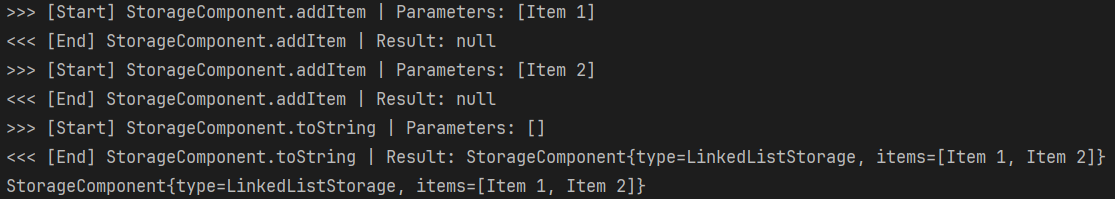


Рисунок 1 – Результат работы программы с хранилищем в виде связанного списка

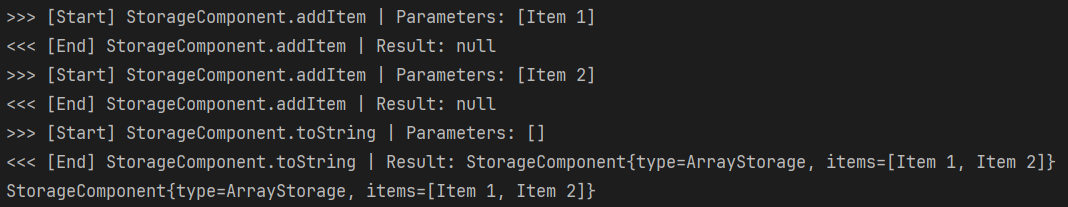


Рисунок 2 – Результат работы программы с хранилищем в виде динамического массива

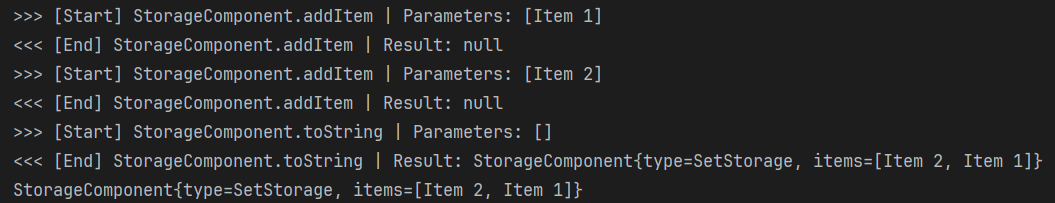


Рисунок 3 – Результат работы программы с хранилищем в виде набора

# **КОД ПРОГРАММЫ**

package Lab1\_CDI.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})  
public @interface ArrayStorage {  
}

package Lab1\_CDI.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})  
public @interface LinkedListStorage {  
}

package Lab1\_CDI.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})  
public @interface SetStorage {  
}

package Lab1\_CDI.Component;  
  
import Lab1\_CDI.Annotation.ArrayStorage;  
import Lab1\_CDI.Annotation.LinkedListStorage;  
import Lab1\_CDI.Annotation.SetStorage;  
import Lab1\_CDI.Interceptor.StorageInterceptor;  
import Lab1\_CDI.Storage.StorageManager;  
import lombok.NoArgsConstructor;  
  
import javax.inject.Inject;  
import javax.interceptor.Interceptors;  
import java.util.Collection;  
  
@Interceptors(StorageInterceptor.class) // Interceptor  
@NoArgsConstructor  
public class StorageComponent<T> {  
  
 @Inject  
// @ArrayStorage  
// @LinkedListStorage  
 @SetStorage  
 private StorageManager<T> storageManager;  
  
 public void addItem(T item) {  
 storageManager.addItem(item);  
 }  
  
 public Collection<T> getItems() {  
 return storageManager.getItems();  
 }  
  
 @Override  
 public String toString() {  
 String storageType;  
  
 if (storageManager.getClass().isAnnotationPresent(ArrayStorage.class)) {  
 storageType = "ArrayStorage";  
 } else if (storageManager.getClass().isAnnotationPresent(LinkedListStorage.class)) {  
 storageType = "LinkedListStorage";  
 } else if (storageManager.getClass().isAnnotationPresent(SetStorage.class)) {  
 storageType = "SetStorage";  
 } else {  
 storageType = "UnknownStorage";  
 }  
  
 return "StorageComponent{" +  
 "type=" + storageType +  
 ", items=" + getItems() +  
 '}';  
 }  
}

package Lab1\_CDI.Interceptor;  
  
import javax.interceptor.AroundInvoke;  
import javax.interceptor.InvocationContext;  
import java.util.Arrays;  
  
public class StorageInterceptor {  
  
 @AroundInvoke  
 public Object logOperation(InvocationContext ic) throws Exception {  
 // Получаем реальный класс через метод getTarget(), если это прокси  
 String className = ic.getTarget().getClass().getSuperclass().getSimpleName();  
 String methodName = ic.getMethod().getName();  
 String params = Arrays.toString(ic.getParameters());  
  
 System.out.println(">>> [Start] " + className + "." + methodName + " | Parameters: " + params);  
  
 Object result = null;  
 try {  
 result = ic.proceed();  
 } finally {  
 System.out.println("<<< [End] " + className + "." + methodName + " | Result: " + result);  
 }  
  
 return result;  
 }  
}

package Lab1\_CDI.Storage;  
  
import Lab1\_CDI.Annotation.ArrayStorage;  
  
import java.util.ArrayList;  
import java.util.Collection;  
  
@ArrayStorage  
public class ArrayStorageManager<T> implements StorageManager<T> {  
 private final ArrayList<T> storage = new ArrayList<>();  
  
 @Override  
 public void addItem(T item) {  
 storage.add(item);  
 }  
  
 @Override  
 public Collection<T> getItems() {  
 return storage;  
 }  
}

package Lab1\_CDI.Storage;  
  
  
import Lab1\_CDI.Annotation.LinkedListStorage;  
  
import java.util.Collection;  
import java.util.LinkedList;  
  
@LinkedListStorage  
public class LinkedListStorageManager<T> implements StorageManager<T> {  
 private final LinkedList<T> storage = new LinkedList<>();  
  
 @Override  
 public void addItem(T item) {  
 storage.add(item);  
 }  
  
 @Override  
 public Collection<T> getItems() {  
 return storage;  
 }  
}

package Lab1\_CDI.Storage;  
  
  
import Lab1\_CDI.Annotation.SetStorage;  
  
import java.util.Collection;  
import java.util.HashSet;  
  
@SetStorage  
public class SetStorageManager<T> implements StorageManager<T> {  
 private final HashSet<T> storage = new HashSet<>();  
  
 @Override  
 public void addItem(T item) {  
 storage.add(item);  
 }  
  
 @Override  
 public Collection<T> getItems() {  
 return storage;  
 }  
}

package Lab1\_CDI.Storage;  
  
import java.util.Collection;  
  
public interface StorageManager<T> {  
 void addItem(T item);  
 Collection<T> getItems();  
}

package Lab1\_CDI;  
  
import Lab1\_CDI.Component.StorageComponent;  
import lombok.NoArgsConstructor;  
import org.jboss.weld.environment.se.Weld;  
import org.jboss.weld.environment.se.WeldContainer;  
  
@NoArgsConstructor  
public class Laboratory1 {  
 protected static Weld weld;  
 protected static WeldContainer container;  
  
 public static void main(String[] args) {  
  
 try {  
 weld = new Weld();  
 container = weld.initialize();  
  
 StorageComponent<String> storage = container.select(StorageComponent.class).get();  
  
 storage.addItem("Item 1");  
 storage.addItem("Item 2");  
  
 System.out.println(storage);  
  
 } finally {  
 weld.shutdown();  
 }  
 }  
}

# **ВЫВОД**

В ходе выполнения лабораторной работы была создана программа, реализующая компонент управления данными в различных коллекциях с использованием технологий CDI и перехватчиков. Программа продемонстрировала гибкость и возможность выбора типа коллекции (динамический массив, связанный список или множество) на основе аннотаций. Логирование операций предоставляло подробный вывод о работе программы, включая параметры и результаты вызовов методов.