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 №13

Выполнил: Веселков М.Ю.

Группа: УИС-411

Преподаватель: доц. Кафедры ЦТУТП

Козьяков П. О.

­

Москва 2024 г.

**СОДЕРЖАНИЕ**

[**ПОСТАНОВКА ЗАДАЧИ** 3](#_Toc184059904)

[**ХОД ВЫПОЛНЕНИЯ РАБОТЫ** 4](#_Toc184059905)

[**РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ** 5](#_Toc184059906)

[**КОД ПРОГРАММЫ** 6](#_Toc184059907)

[**ВЫВОД** 10](#_Toc184059908)

# **ПОСТАНОВКА ЗАДАЧИ**

Создать компонент CDI, выполняющий сохранение данных в различных типах файлов:  XML, JSON, YAML. Выбор способа сохранения данных должен быть реализован через указание соответствующей аннотации или посредством указания класса в XML файле. Посредством перехватчиков провести логирование записи данных.

# **ХОД ВЫПОЛНЕНИЯ РАБОТЫ**

В начале выполнения программы инициализируется CDI-контейнер с использованием библиотеки Weld, который управляет зависимостями и контекстами приложения. Контейнер автоматически настраивается с помощью вызова метода initialize(), что позволяет работать с компонентами, такими как классы с аннотациями @Inject и пользовательскими квалификаторами. Затем из контейнера извлекается экземпляр класса DataSaverComponent, предназначенного для управления процессом сохранения данных в различных форматах. Выбор формата определяется с использованием квалификаторов, таких как @XML, @JSON или @YAML, что позволяет гибко переключаться между форматами сохранения без изменения логики кода.

Далее создается объект данных, который необходимо сохранить. Этот объект может представлять любую структуру данных, например, пользовательские классы, содержащие необходимые поля. Путь к файлу для сохранения указывается с учетом желаемого расширения, которое зависит от выбранного метода сохранения данных. При вызове метода saveData класса DataSaverComponent автоматически выбирается соответствующая реализация интерфейса IDataSaver, такая как XmlDataSaver, JsonDataSaver или YamlDataSaver.

Компонент также автоматически изменяет расширение файла в пути, чтобы оно соответствовало выбранному формату, исключая необходимость ручного редактирования. Процесс сохранения данных включает сериализацию объекта в выбранный формат, обработку возможных исключений и вывод сообщений об успешности операции. После завершения всех операций контейнер Weld корректно завершает свою работу, освобождая ресурсы и завершая выполнение программы. Этот подход позволяет легко добавлять новые форматы данных или модифицировать существующую логику без значительных изменений в приложении.

# **РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ**

Таблица 1 – Результаты работы кода

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | T1 (нс) | T2 (нс) | T3 (нс) | T4 (нс) |
| JSON | 240255900 | 255325900 | 249571100 | 226565600 |
| XML | 51651700 | 44762700 | 44001400 | 49485400 |
| YAML | 58793600 | 63224400 | 66795300 | 66790900 |

Среднее время работы программы при записи с расширением .json – 242929625 нс, с расширением .xml – 47475300 нс, с расширением .yaml – 63901050 нс.

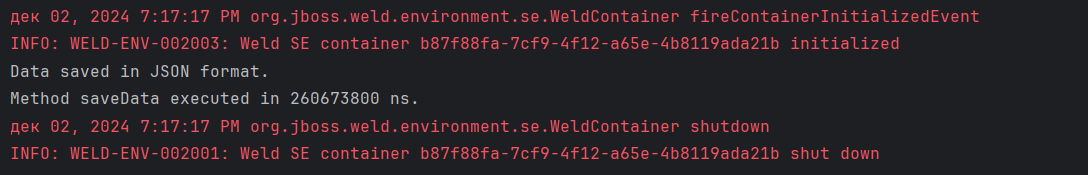


Рисунок 1 – Сохранение с расширением .json

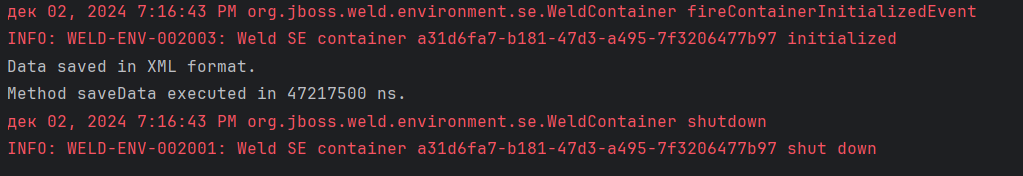


Рисунок 2 – Сохранение с расширением .xml

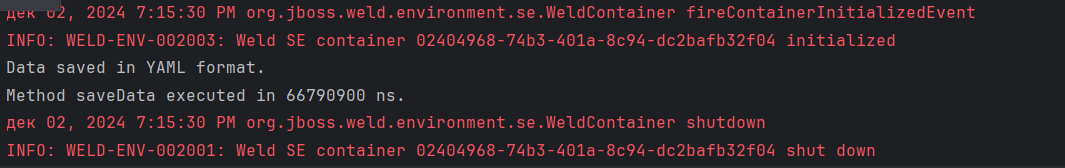


Рисунок 3 – Сохранение с расширением .yaml

# **КОД ПРОГРАММЫ**

package Lab1\_CDI.Component;  
  
import Lab1\_CDI.DataSaver.Annotation.JSON;  
import Lab1\_CDI.DataSaver.Annotation.YAML;  
import Lab1\_CDI.DataSaver.IDataSaver;  
import Lab1\_CDI.DataSaver.Annotation.XML;  
import Lab1\_CDI.Interceptor.LoggingInterceptor;  
  
import javax.inject.Inject;  
import javax.interceptor.Interceptors;  
  
@Interceptors(LoggingInterceptor.class)  
public class DataSaverComponent {  
  
 @Inject  
*// @JSON* @XML  
*// @YAML* private IDataSaver dataSaver;  
  
 public void saveData(Object data, String filePath) throws Exception {  
 dataSaver.save(data, filePath);  
 }  
}

package Lab1\_CDI.DataSaver.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.*TYPE*, ElementType.*FIELD*, ElementType.*METHOD*, ElementType.*PARAMETER*})  
public @interface JSON {  
}

package Lab1\_CDI.DataSaver.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.*TYPE*, ElementType.*FIELD*, ElementType.*METHOD*, ElementType.*PARAMETER*})  
public @interface XML {  
}

package Lab1\_CDI.DataSaver.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.*TYPE*, ElementType.*FIELD*, ElementType.*METHOD*, ElementType.*PARAMETER*})  
public @interface YAML {  
}

package Lab1\_CDI.DataSaver;  
  
public interface IDataSaver {  
 void save(Object data, String filePath) throws Exception;  
}

package Lab1\_CDI.DataSaver;  
  
import Lab1\_CDI.DataSaver.Annotation.JSON;  
import com.fasterxml.jackson.databind.ObjectMapper;  
import javax.enterprise.context.Dependent;  
import javax.inject.Named;  
import java.io.File;  
import java.io.IOException;  
  
@Named("JSON")  
@Dependent  
@JSON  
public class JsonDataSaver implements IDataSaver {  
  
 @Override  
 public void save(Object data, String filePath) throws Exception {  
 ObjectMapper objectMapper = new ObjectMapper();  
 try {  
 objectMapper.writeValue(new File(filePath), data);  
 System.*out*.println("Data saved in JSON format.");  
 } catch (IOException e) {  
 e.printStackTrace();  
 throw new Exception("Error saving JSON");  
 }  
 }  
}

package Lab1\_CDI.DataSaver;  
  
import javax.enterprise.context.Dependent;  
import javax.inject.Named;  
import java.io.File;  
import javax.xml.bind.JAXBContext;  
import javax.xml.bind.JAXBException;  
import javax.xml.bind.Marshaller;  
import Lab1\_CDI.DataSaver.Annotation.XML;  
  
@Named("XML")  
@Dependent  
@XML  
public class XmlDataSaver implements IDataSaver {  
  
 @Override  
 public void save(Object data, String filePath) throws Exception {  
 try {  
 JAXBContext context = JAXBContext.*newInstance*(data.getClass());  
 Marshaller marshaller = context.createMarshaller();  
 marshaller.setProperty(Marshaller.*JAXB\_FORMATTED\_OUTPUT*, true);  
 marshaller.marshal(data, new File(filePath));  
 System.*out*.println("Data saved in XML format.");  
 } catch (JAXBException e) {  
 e.printStackTrace();  
 throw new Exception("Error saving XML");  
 }  
 }  
}

package Lab1\_CDI.DataSaver;  
  
import Lab1\_CDI.DataSaver.Annotation.YAML;  
import org.yaml.snakeyaml.DumperOptions;  
import org.yaml.snakeyaml.Yaml;  
import javax.enterprise.context.Dependent;  
import javax.inject.Named;  
import java.io.FileWriter;  
import java.io.IOException;  
  
  
@Named("YAML")  
@Dependent  
@YAML  
public class YamlDataSaver implements IDataSaver {  
  
 @Override  
 public void save(Object data, String filePath) throws Exception {  
 Yaml yaml = new Yaml(new DumperOptions());  
 try (FileWriter writer = new FileWriter(filePath)) {  
 yaml.dump(data, writer);  
 System.*out*.println("Data saved in YAML format.");  
 } catch (IOException e) {  
 e.printStackTrace();  
 throw new Exception("Error saving YAML");  
 }  
 }  
}

package Lab1\_CDI.Interceptor;  
  
import javax.interceptor.AroundInvoke;  
import javax.interceptor.InvocationContext;  
  
public class LoggingInterceptor {  
  
 @AroundInvoke  
 public Object log(InvocationContext context) throws Exception {  
 long start = System.*nanoTime*();  
 Object result = context.proceed();  
 long end = System.*nanoTime*();  
 System.*out*.println("Method " + context.getMethod().getName() + " executed in " + (end - start) + " ns.");  
 return result;  
 }  
}

package Lab1\_CDI;  
  
import javax.xml.bind.annotation.XmlElement;  
import javax.xml.bind.annotation.XmlRootElement;  
import com.fasterxml.jackson.annotation.JsonProperty;  
  
  
@XmlRootElement  
public class YourDataObject {  
  
  
 private String name;  
 private int age;  
  
  
 public YourDataObject() {}  
  
  
 public YourDataObject(String name, int age) {  
 this.name = name;  
 this.age = age;  
 }  
  
  
 @XmlElement  
 @JsonProperty("name")  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 @XmlElement  
 @JsonProperty("age")  
 public int getAge() {  
 return age;  
 }  
  
 public void setAge(int age) {  
 this.age = age;  
 }  
  
  
 @Override  
 public String toString() {  
 return "YourDataObject{name='" + name + "', age=" + age + '}';  
 }  
}

package Lab1\_CDI;  
  
import Lab1\_CDI.Component.DataSaverComponent;  
import org.jboss.weld.environment.se.Weld;  
import org.jboss.weld.environment.se.WeldContainer;  
  
public class Laboratory1 {  
  
 public static void main(String[] args) {  
 Weld weld = new Weld();  
 WeldContainer container = weld.initialize();  
  
 try {  
 DataSaverComponent dataSaverComponent = container.select(DataSaverComponent.class).get();  
 Object data = new YourDataObject("Веселков", 21);  
 String filePath = "C:\\Users\\Matthew\\Documents\\1\\output.xml";  
 dataSaverComponent.saveData(data, filePath);  
 } catch (Exception e)  
 {  
 e.printStackTrace();  
 } finally {  
 weld.shutdown();  
 }  
 }  
}

# **ВЫВОД**

В ходе выполнения лабораторной работы был создан CDI-компонент для сохранения данных в различные форматы файлов: JSON, XML и YAML. Использовался механизм внедрения зависимостей для управления объектами и их взаимодействиями, а также аннотации для выбора формата записи. Время выполнения операции сохранения данных в каждом формате было измерено и сравнено.

По результатам работы программы, сохранение данных в формате JSON заняло в среднем 242,93 мс, в формате XML — 47,48 мс, а в формате YAML — 63,90 мс. Эти результаты показывают, что формат XML является наиболее эффективным по скорости, в то время как формат JSON требует наибольшего времени. Формат YAML занимает промежуточное место по производительности.

Лабораторная работа позволила закрепить навыки работы с CDI-компонентами, аннотациями и перехватчиками, а также продемонстрировала, как выбор подходящего формата данных может существенно повлиять на производительность приложения.