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 №12

Выполнил: Бугаева А. Л.

Группа: УИС-411

Преподаватель: доц. Кафедры ЦТУТП

Козьяков П. О.

­

Москва 2024 г.

**СОДЕРЖАНИЕ**

[ПОСТАНОВКА ЗАДАЧИ 3](#_Toc180328187)

[ХОД ВЫПОЛНЕНИЯ РАБОТЫ 4](#_Toc180328188)

[РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ 6](#_Toc180328189)

[КОД ПРОГРАММЫ 7](#_Toc180328190)

[ВЫВОД 12](#_Toc180328191)

# **ПОСТАНОВКА ЗАДАЧИ**

Создать компонент CDI, выполняющий загрузку данных из различных типов файлов:  XML, JSON, YAML. Выбор способа загрузки данных должен быть реализован через указание соответствующей аннотации или посредством указания класса в XML файле. Посредством перехватчиков провести логирование чтения данных.

# **ХОД ВЫПОЛНЕНИЯ РАБОТЫ**

В ходе выполнения лабораторной работы был разработан проект, использующий технологии CDI (Contexts and Dependency Injection) для чтения данных из файлов в различных форматах (JSON, XML и YAML). Код реализует инъекцию зависимостей с использованием аннотаций и предоставляет интерфейс для работы с различными форматами данных.

Работа началась с создания интерфейса IFileReader, который определяет метод readContent(String filePath). Этот интерфейс служит общим контрактом для всех классов, реализующих чтение данных из файлов. Каждая реализация этого интерфейса (например, JSONFileReader, XMLFileReader и YAMLFileReader) отвечает за чтение данных из конкретного формата файла.

Далее были созданы аннотации-квалификаторы @JSON, @XML и @YAML, которые позволяют различать реализации IFileReader при инъекции зависимостей. Эти аннотации помогают CDI выбрать нужный класс для использования в зависимости от требований к формату данных.

Класс FileReader использует аннотацию @Inject для инъекции зависимости IFileReader, что позволяет легко изменять формат данных, который будет считываться, просто изменяя используемую аннотацию. В коде была временно закомментирована инъекция для форматов JSON и XML, оставив только YAML.

Каждый из классов, реализующих IFileReader, содержит метод readContent, который читает файл и возвращает содержимое в виде строки. В процессе чтения используется библиотека Jackson, которая позволяет эффективно работать с различными форматами данных. Если во время чтения файла возникает ошибка, она обрабатывается, и пользователю возвращается сообщение об ошибке.

В классе ProfileInterceptor реализован механизм перехвата, который замеряет время выполнения метода чтения файла. Это позволяет оценить производительность работы с разными форматами данных. Перехватчик регистрирует время выполнения в миллисекундах после завершения метода.

Для запуска приложения в классе Laboratory1 инициализируется CDI-контейнер Weld. Затем осуществляется инъекция класса FileReader, после чего вызывается метод readContent с указанием пути к YAML-файлу. Результат выполнения выводится на консоль.

Таким образом, была успешно реализована система, позволяющая считывать данные из файлов в разных форматах с использованием CDI, аннотаций и механизма перехвата. Проект демонстрирует принципы инкапсуляции и модульности, обеспечивая простоту и гибкость в работе с данными.

# **РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ**

Таблица 1 – Результаты работы кода

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | T1 (мс) | T2 (мс) | T3 (мс) | T4 (нс) |
| JSON | 307 | 312 | 308 | 305 |
| XML | 338 | 348 | 340 | 338 |
| YAML | 369 | 359 | 347 | 345 |

Для каждого формата данных были вычислены средние времена обработки. Среднее время для JSON составляет 308.5 мс, для XML — 342.67 мс, а для YAML — 358.25 мс.

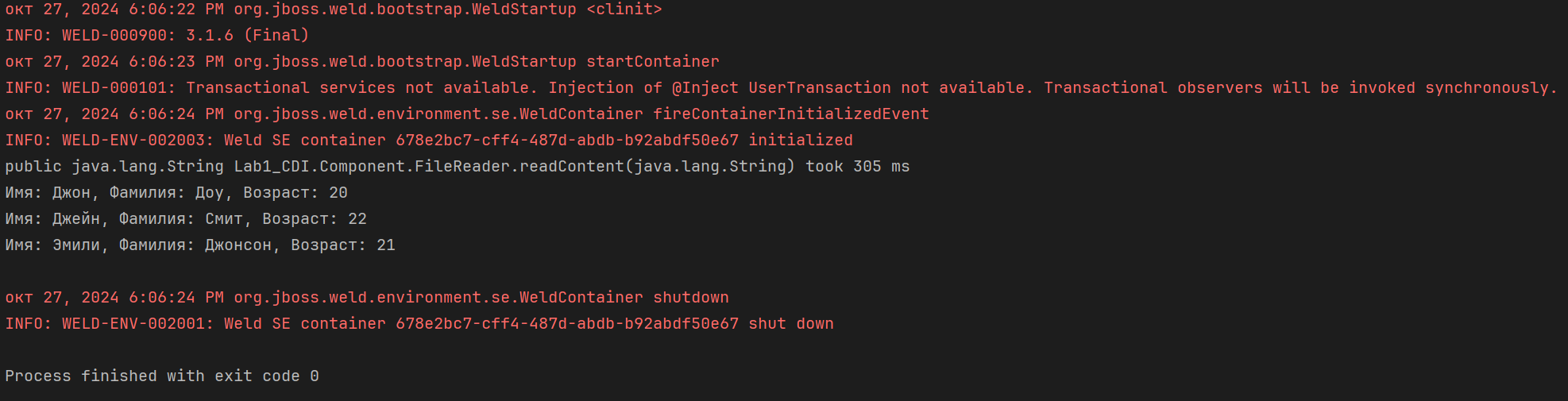


Рисунок 1 – Результат работы программы с JSON-файлом

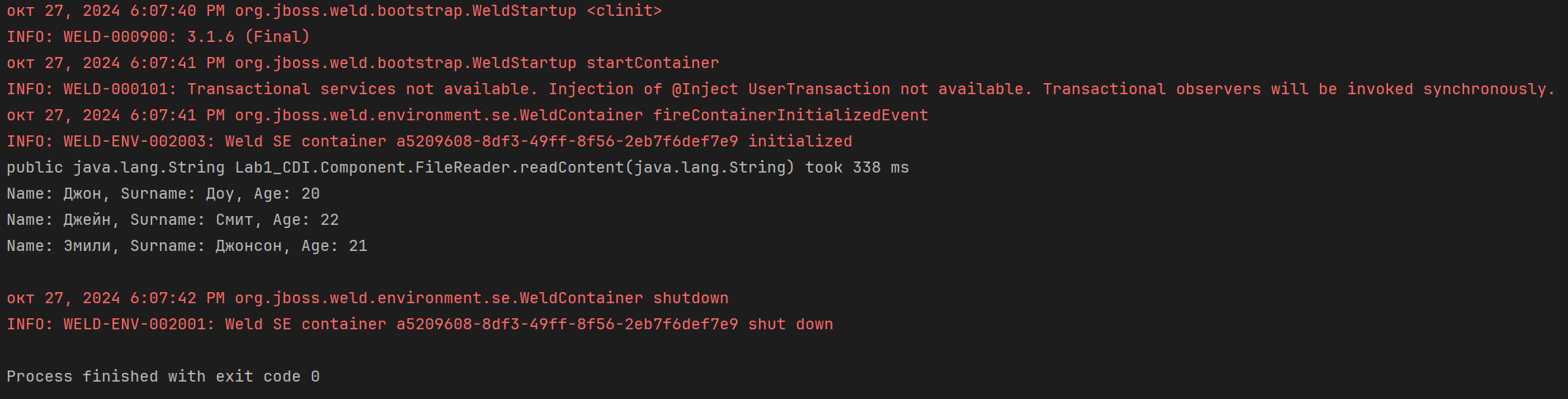


Рисунок 2 – Результат работы программы с XML-файлом

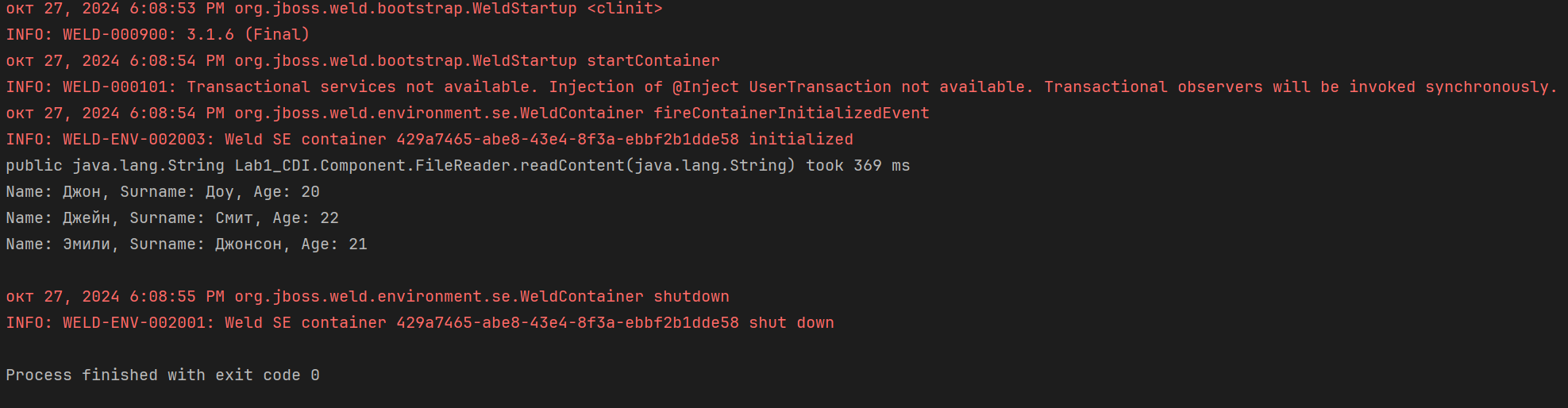


Рисунок 3 – Результат работы программы с YAML-файлом

# **КОД ПРОГРАММЫ**

package Lab1\_CDI.Component;  
  
import Lab1\_CDI.FileReader.Abstraction.IFileReader;  
import Lab1\_CDI.FileReader.Annotation.JSON;  
import Lab1\_CDI.FileReader.Annotation.XML;  
import Lab1\_CDI.FileReader.Annotation.YAML;  
import Lab1\_CDI.Interceptor.ProfileInterceptor;  
  
import javax.inject.Inject;  
import javax.interceptor.Interceptors;  
  
@Interceptors(ProfileInterceptor.class)  
public class FileReader {  
  
 @Inject  
// @JSON  
// @XML  
 @YAML  
 private IFileReader fileReader;  
  
 public String readContent(String filepath) {  
 return fileReader.readContent(filepath);  
 }  
}

package Lab1\_CDI.FileReader.Abstraction;  
  
public interface IFileReader {  
 String readContent(String filePath);  
}

package Lab1\_CDI.FileReader.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 JSON {  
}

package Lab1\_CDI.FileReader.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 XML {  
}

package Lab1\_CDI.FileReader.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 YAML {  
}

package Lab1\_CDI.FileReader;  
  
import Lab1\_CDI.FileReader.Abstraction.IFileReader;  
import Lab1\_CDI.FileReader.Annotation.JSON;  
import Lab1\_CDI.Model.Student;  
import com.fasterxml.jackson.core.type.TypeReference;  
import com.fasterxml.jackson.databind.ObjectMapper;  
import lombok.NoArgsConstructor;  
  
import java.io.File;  
import java.io.IOException;  
import java.util.List;  
  
@JSON  
@NoArgsConstructor  
public class JSONFileReader implements IFileReader {  
 @Override  
 public String readContent(String filePath) {  
 StringBuilder output = new StringBuilder(); // Создаём StringBuilder для формирования строки  
 try {  
 // Чтение студентов из файла  
 List<Student> students = readJSONFile(filePath);  
 // Формируем строку для вывода  
 for (Student student : students) {  
 output.append("Имя: ").append(student.getName())  
 .append(", Фамилия: ").append(student.getSurname())  
 .append(", Возраст: ").append(student.getAge())  
 .append("\n"); // Переход на новую строку  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 return "Ошибка при чтении файла: " + e.getMessage(); // Возвращаем сообщение об ошибке  
 }  
  
 return output.toString(); // Возвращаем сформированную строку  
 }  
  
 public List<Student> readJSONFile(String filePath) throws IOException {  
 ObjectMapper objectMapper = new ObjectMapper();  
 return objectMapper.readValue(new File(filePath), new TypeReference<List<Student>>() {});  
 }  
}

package Lab1\_CDI.FileReader;  
  
import Lab1\_CDI.FileReader.Abstraction.IFileReader;  
import Lab1\_CDI.FileReader.Annotation.XML;  
import Lab1\_CDI.Model.Student;  
import com.fasterxml.jackson.core.type.TypeReference;  
import com.fasterxml.jackson.dataformat.xml.XmlMapper;  
import lombok.NoArgsConstructor;  
  
import java.io.File;  
import java.io.IOException;  
import java.util.List;  
  
@XML  
@NoArgsConstructor  
public class XMLFileReader implements IFileReader {  
 @Override  
 public String readContent(String filePath) {  
 StringBuilder output = new StringBuilder(); // Создаём StringBuilder для формирования строки  
 try {  
 // Чтение студентов из файла  
 List<Student> students = readXMLFile(filePath);  
 // Формируем строку для вывода  
 for (Student student : students) {  
 output.append("Name: ").append(student.getName())  
 .append(", Surname: ").append(student.getSurname())  
 .append(", Age: ").append(student.getAge())  
 .append("\n"); // Переход на новую строку  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 return "Ошибка при чтении файла: " + e.getMessage(); // Возвращаем сообщение об ошибке  
 }  
  
 return output.toString(); // Возвращаем сформированную строку  
 }  
  
 public List<Student> readXMLFile(String filePath) throws IOException {  
 XmlMapper xmlMapper = new XmlMapper();  
 return xmlMapper.readValue(new File(filePath), new TypeReference<List<Student>>() {  
 });  
 }  
}

package Lab1\_CDI.FileReader;  
  
import Lab1\_CDI.FileReader.Abstraction.IFileReader;  
import Lab1\_CDI.FileReader.Annotation.YAML;  
import Lab1\_CDI.Model.Student;  
import com.fasterxml.jackson.core.type.TypeReference;  
import com.fasterxml.jackson.dataformat.yaml.YAMLMapper;  
import lombok.NoArgsConstructor;  
  
import java.io.File;  
import java.io.IOException;  
import java.util.List;  
  
@YAML  
@NoArgsConstructor  
public class YAMLFileReader implements IFileReader {  
 @Override  
 public String readContent(String filePath) {  
 StringBuilder output = new StringBuilder();  
 try {  
 // Чтение студентов из файла  
 List<Student> students = readYAMLFile(filePath);  
 // Формируем строку для вывода  
 for (Student student : students) {  
 output.append("Name: ").append(student.getName())  
 .append(", Surname: ").append(student.getSurname())  
 .append(", Age: ").append(student.getAge())  
 .append("\n"); // Переход на новую строку  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 return "Ошибка при чтении файла: " + e.getMessage(); // Возвращаем сообщение об ошибке  
 }  
  
 return output.toString(); // Возвращаем сформированную строку  
 }  
  
 public List<Student> readYAMLFile(String filePath) throws IOException {  
 YAMLMapper yamlMapper = new YAMLMapper();  
 return yamlMapper.readValue(new File(filePath), new TypeReference<List<Student>>() {});  
 }  
}

package Lab1\_CDI.Interceptor;  
  
import lombok.NoArgsConstructor;  
  
import javax.interceptor.AroundInvoke;  
import javax.interceptor.InvocationContext;  
  
@NoArgsConstructor  
public class ProfileInterceptor {  
  
 @AroundInvoke  
 public Object profile(InvocationContext ic) throws Exception {  
 long initTime = System.currentTimeMillis(); // Начальное время в миллисекундах  
 Object result;  
  
 try {  
 result = ic.proceed(); // Выполнение метода  
 } finally {  
 long diffTime = System.currentTimeMillis() - initTime; // Разница времени в миллисекундах  
 System.out.println(ic.getMethod() + " took " + diffTime + " ms"); // Вывод в миллисекундах  
 }  
  
 return result; // Возврат результата метода  
 }  
}

package Lab1\_CDI.Model;  
  
import lombok.\*;  
  
@Data  
public class Student {  
 private String name;  
 private String surname;  
 private int age;  
}

package Lab1\_CDI;  
  
import Lab1\_CDI.Component.FileReader;  
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) {  
 weld = new Weld();  
 container = weld.initialize();  
  
 try {  
  
// String JSON\_FILE\_PATH = "src/main/resources/files/data.json";  
// String XML\_FILE\_PATH = "src/main/resources/files/data.xml";  
 String YAML\_FILE\_PATH = "src/main/resources/files/data.yaml";  
  
 FileReader fileReader = container.select(FileReader.class).get();  
  
// String output = fileReader.readContent(JSON\_FILE\_PATH);  
// String output = fileReader.readContent(XML\_FILE\_PATH);  
 String output = fileReader.readContent(YAML\_FILE\_PATH);  
  
 System.out.println(output);  
  
 } finally {  
 weld.shutdown();  
 }  
 }  
}

# **ВЫВОД**

В ходе выполнения лабораторной работы была реализована система для чтения данных из файлов форматов JSON, XML и YAML с использованием технологии CDI (Contexts and Dependency Injection). Проект продемонстрировал практическое применение паттернов инъекции зависимостей и механизма перехвата.

В результате работы была достигнута гибкость кода: благодаря аннотациям-квалификаторам (@JSON, @XML, @YAML) удалось легко переключаться между различными реализациями интерфейса IFileReader. Это позволяет изменять формат данных, который будет считываться, без необходимости изменения кода класса FileReader, что соответствует принципу "разделяй и властвуй".

Реализация перехватчика ProfileInterceptor позволила отслеживать время выполнения метода чтения файла, что является полезным инструментом для анализа производительности. Это дает возможность оптимизировать код в будущем, если это потребуется.

В целом, лабораторная работа показала, как эффективно использовать технологии CDI для создания модульных и поддерживаемых приложений. Реализованный подход значительно упрощает процесс работы с данными и позволяет легко адаптировать приложение под новые требования. Полученные навыки будут полезны в дальнейшей разработке программного обеспечения и при проектировании архитектуры приложений.