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 №14

Выполнил: Сафонов Г.

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 11](#_Toc180328191)

# **ПОСТАНОВКА ЗАДАЧИ**

Создать компонент CDI, выполняющий преобразование чисел из десятичной системы счисления в различные:  шестнадцатеричную, двоичную, восьмеричную. Выбор способа сохранения 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 первую очередь был создан интерфейс INumberConverter, который описывает метод convert(int decimal). Этот метод предназначен для реализации преобразования переданного целого числа в строку, соответствующую определенной системе счисления. Интерфейс обеспечивает унифицированный подход для различных реализаций конвертеров.

Далее, для каждой системы счисления (двоичной, восьмеричной и шестнадцатеричной) были созданы отдельные классы: BinaryConverter, OctalConverter и HexadecimalConverter. Каждый из этих классов реализует интерфейс INumberConverter и использует методы Integer.toBinaryString(), Integer.toOctalString() и Integer.toHexString() соответственно, чтобы выполнить преобразование в требуемую систему счисления. Для каждого конвертера была добавлена соответствующая аннотация (@BINARY, @OCTAL, @HEXADECIMAL), что позволяет CDI отличать реализации друг от друга и выполнять инъекцию необходимого типа конвертера по аннотации.

После создания классов конвертеров была разработана система аннотаций (@BINARY, @OCTAL, @HEXADECIMAL). Каждая из этих аннотаций помечена как @Qualifier, что позволяет использовать их для различения зависимостей при инъекции. Аннотации также определены с областью действия RUNTIME, что позволяет CDI-спецификации использовать их в процессе работы приложения.

Для реализации механизма профилирования методов был создан класс ProfileInterceptor. В этом классе используется метод @AroundInvoke, который позволяет замерить время выполнения метода convert и вывести его в консоль. Этот перехватчик добавлен к основному компоненту NumberConverter с использованием аннотации @Interceptors, что автоматически активирует его при вызове метода convert.

Основная логика работы сосредоточена в классе NumberConverter, который содержит точку инъекции @Inject с аннотацией, указывающей, какой конвертер использовать. Аннотацию можно менять, чтобы протестировать разные конвертеры. Метод convert вызывает метод convert конкретного конвертера, который был внедрен, и возвращает строку с результатом.

Тестирование функциональности программы проводилось в основном классе Laboratory1. Для этого с помощью CDI и Weld контейнера создается объект NumberConverter, после чего происходит конвертация нескольких целых чисел в указанную систему счисления. Результаты выводятся в консоль, а перехватчик отображает время выполнения для каждого вызова метода convert.

В завершение, после выполнения всех операций и тестирования, контейнер Weld завершается с помощью метода weld.shutdown(). Это освобождает ресурсы, задействованные контейнером CDI.

# **РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ**

Таблица 1 – Результаты работы кода

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | T1 (нс) | T2 (нс) | T3 (нс) | T4 (нс) |
| Двоичная | 45500 | 46400 | 46900 | 45100 |
| Восьмеричная | 44300 | 44300 | 45900 | 47300 |
| Шестнадцатеричная | 63700 | 43900 | 45800 | 47900 |

Среднее время работы программы для перевода числа из десятеричной системы счисления в двоичную 0.04598 мс, в восьмеричную 0.04545 мс, в шестнадцатеричную 0.05033 мс.

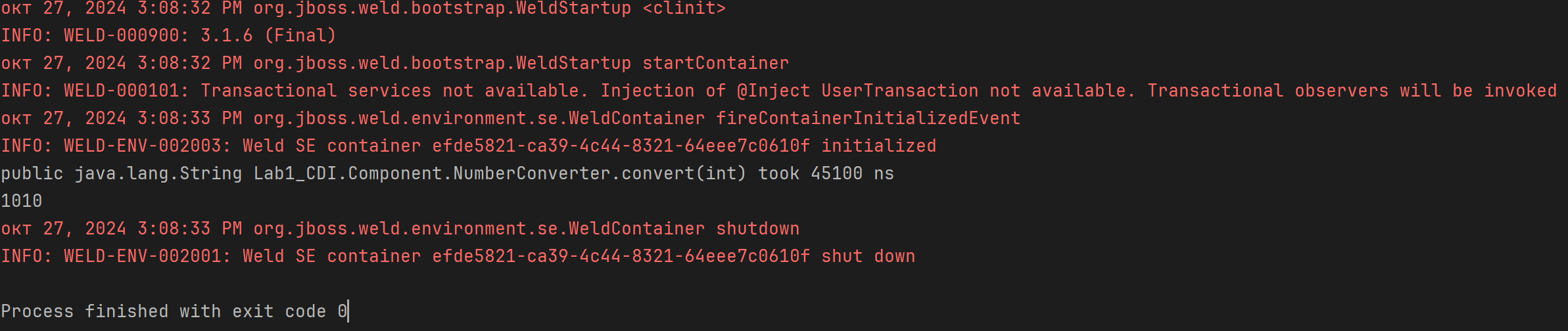


Рисунок 1 – Результат перевода в двоичную

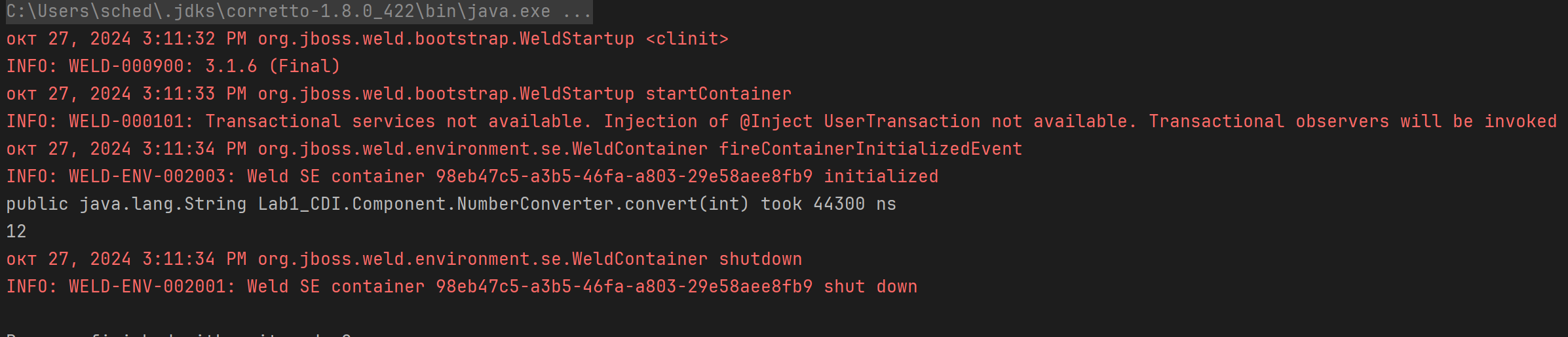


Рисунок 2 – Результат перевода в восьмеричную

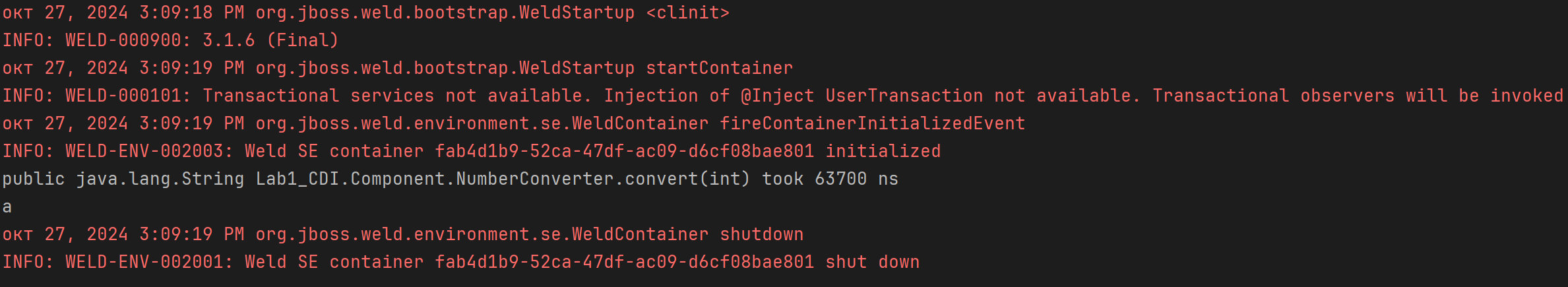


Рисунок 3 – Результат перевода в шестнадцатеричную

# **КОД ПРОГРАММЫ**

package Lab1\_CDI.Component;  
  
import Lab1\_CDI.Converter.Abstraction.INumberConverter;  
import Lab1\_CDI.Converter.Annotation.BINARY;  
import Lab1\_CDI.Converter.Annotation.HEXADECIMAL;  
import Lab1\_CDI.Converter.Annotation.OCTAL;  
import Lab1\_CDI.Interceptor.ProfileInterceptor;  
import lombok.NoArgsConstructor;  
  
import javax.inject.Inject;  
import javax.interceptor.Interceptors;  
  
@Interceptors(ProfileInterceptor.class) // Перехватчик  
@NoArgsConstructor  
public class NumberConverter {  
  
  
 // Точка внедрения зависимости  
 // Dependency Injection  
 @Inject  
// @BINARY  
 @OCTAL  
// @HEXADECIMAL  
 private INumberConverter numberConverter;  
  
 public String convert(int decimal) {  
 return numberConverter.convert(decimal);  
 }  
  
}

package Lab1\_CDI.Converter.Abstraction;  
  
public interface INumberConverter {  
 String convert(int decimal);  
}

package Lab1\_CDI.Converter.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 BINARY {  
}

package Lab1\_CDI.Converter.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 HEXADECIMAL {  
}

/package Lab1\_CDI.Converter.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 OCTAL {  
}

package Lab1\_CDI.Converter;  
  
import Lab1\_CDI.Converter.Abstraction.INumberConverter;  
import Lab1\_CDI.Converter.Annotation.BINARY;  
import lombok.NoArgsConstructor;  
  
@BINARY  
@NoArgsConstructor  
public class BinaryConverter implements INumberConverter {  
  
 @Override  
 public String convert(int decimal) {  
 return Integer.toBinaryString(decimal);  
 }  
}

package Lab1\_CDI.Converter;  
  
import Lab1\_CDI.Converter.Abstraction.INumberConverter;  
import Lab1\_CDI.Converter.Annotation.HEXADECIMAL;  
import lombok.NoArgsConstructor;  
  
@HEXADECIMAL  
@NoArgsConstructor  
public class HexadecimalConverter implements INumberConverter {  
  
 @Override  
 public String convert(int decimal) {  
 return Integer.toHexString(decimal);  
 }  
}

package Lab1\_CDI.Converter;  
  
import Lab1\_CDI.Converter.Abstraction.INumberConverter;  
import Lab1\_CDI.Converter.Annotation.OCTAL;  
import lombok.NoArgsConstructor;  
  
@OCTAL  
@NoArgsConstructor  
public class OctalConverter implements INumberConverter {  
  
 @Override  
 public String convert(int decimal) {  
 return Integer.toOctalString(decimal);  
 }  
}

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.nanoTime();  
 Object result;  
  
 try {  
 result = ic.proceed();  
 } finally {  
 long diffTime = System.nanoTime() - initTime;  
 System.out.println(ic.getMethod() + " took " + diffTime + " ns");  
 }  
  
 return result;  
 }  
}

package Lab1\_CDI;  
  
import Lab1\_CDI.Component.NumberConverter;  
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 {  
  
 NumberConverter numberConverter = container.select(NumberConverter.class).get();  
  
 int decimalToConvert = 10;  
  
 System.out.println(numberConverter.convert(decimalToConvert));  
  
 } finally {  
 weld.shutdown();  
 }  
 }  
}

# **ВЫВОД**

В ходе выполнения лабораторной работы была разработана модульная система для конвертации целых чисел в различные системы счисления (двоичную, восьмеричную и шестнадцатеричную) с применением технологий CDI и перехватчиков для профилирования. Мы создали интерфейс INumberConverter, который позволил задать единый подход к преобразованию чисел, а также разработали три реализации конвертеров с использованием аннотаций @BINARY, @OCTAL и @HEXADECIMAL для их различения и внедрения через CDI.

Использование CDI позволило гибко управлять зависимостями, легко заменяя типы конвертеров без изменения основной логики компонента NumberConverter. Реализованный перехватчик ProfileInterceptor добавил возможность мониторинга производительности, что особенно важно для оценки времени выполнения и оптимизации приложений.

Таким образом, лабораторная работа показала преимущества использования CDI в построении модульных, легко расширяемых и тестируемых приложений. Аннотации и перехватчики позволили сократить объем кода, повысить его читаемость и упростить внедрение новых функциональных модулей, что особенно полезно для поддержки и масштабирования приложений.