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: Шедания В. М.

Группа: УИС-411

Преподаватель: доц. Кафедры ЦТУТП

Козьяков П. О.

­

Москва 2024 г.

**СОДЕРЖАНИЕ**

[**ПОСТАНОВКА ЗАДАЧИ** 3](#_Toc179146076)

[ХОД ВЫПОЛНЕНИЯ РАБОТЫ 4](#_Toc179146077)

[РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ 5](#_Toc179146078)

[КОД ПРОГРАММЫ 6](#_Toc179146079)

# **ПОСТАНОВКА ЗАДАЧИ**

Создать компонент CDI, выполняющий нахождение тангенса угла различными способами (по синусу и косинусу, только по косинусу). Выбор способа нахождение тангенса угла должен быть реализован через указание соответствующей аннотации или посредством указания класса в XML файле. Посредством перехватчиков провести сравнение по времени эффективности различных методов.

# **ХОД ВЫПОЛНЕНИЯ РАБОТЫ**

В начале выполнения кода из лабораторной работы инициализируется CDI-контейнер. Создается экземпляр класса Weld, который управляет контекстами и зависимостями в приложении. После этого вызывается метод инициализации, что создает и настраивает контейнер для дальнейшей работы.

Следующим шагом осуществляется извлечение экземпляра класса TangentCalculator из CDI-контейнера. Этот класс предназначен для вычисления тангенса на основе переданных ему параметров. Он использует механизм внедрения зависимостей, что позволяет легко управлять объектами и их зависимостями в рамках приложения.

Затем создаются параметры для вычисления тангенса. В зависимости от того, какой тип параметров требуется, могут быть использованы различные классы, такие как CosineOnlyParameters или SineCosineParameters. В этом конкретном случае используются параметры для синуса и косинуса, что позволяет передать два значения — одно для синуса и одно для косинуса.

После создания параметров вызывается метод calculateTangent экземпляра TangentCalculator. Этот метод осуществляет расчет тангенса на основе переданных параметров, извлекая значения синуса и косинуса. Внутри метода происходит логирование, что позволяет отслеживать, какой именно метод был вызван для расчета.

В методах для вычисления тангенса сначала преобразуются значения синуса и косинуса в тип double, что позволяет производить математические операции. Если используются параметры только для косинуса, вычисление синуса осуществляется с помощью формулы sin = √(1 - cos²). Затем тангенс рассчитывается как отношение синуса к косинусу.

Наконец, после выполнения всех вычислений и получения результата, он выводится на экран. После завершения всех операций происходит завершение работы контейнера Weld, что освобождает ресурсы, занятые приложением.

# **РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ**

Таблица 1 – Результаты работы кода

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | T1 (нс) | T2 (нс) | T3 (нс) | T4 (нс) |
| По косинусу | 357500 | 330900 | 340600 | 343700 |
| Синус и косинус | 151200 | 194200 | 155700 | 155700 |

Среднее время работы программы расчёта тангенса по косинусу 0.343175 мс, по синусу и косинусу – 0.1642 мс.

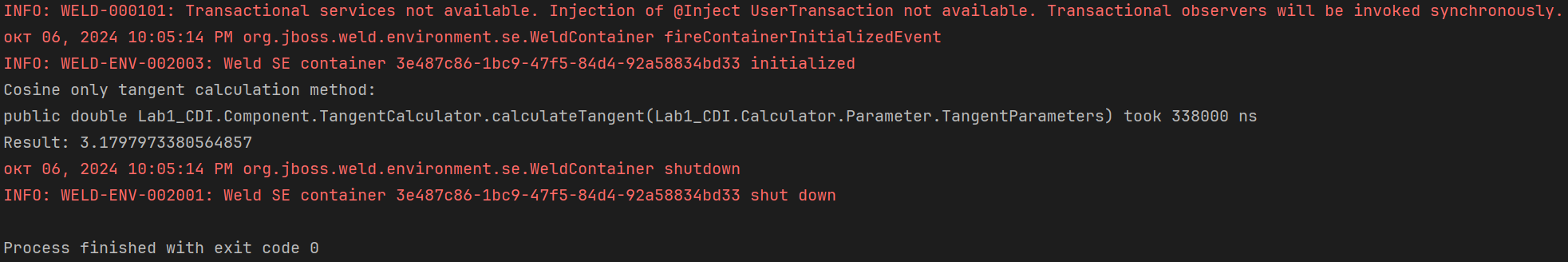


Рисунок 1 – Результат расчётов тангенса по косинусу

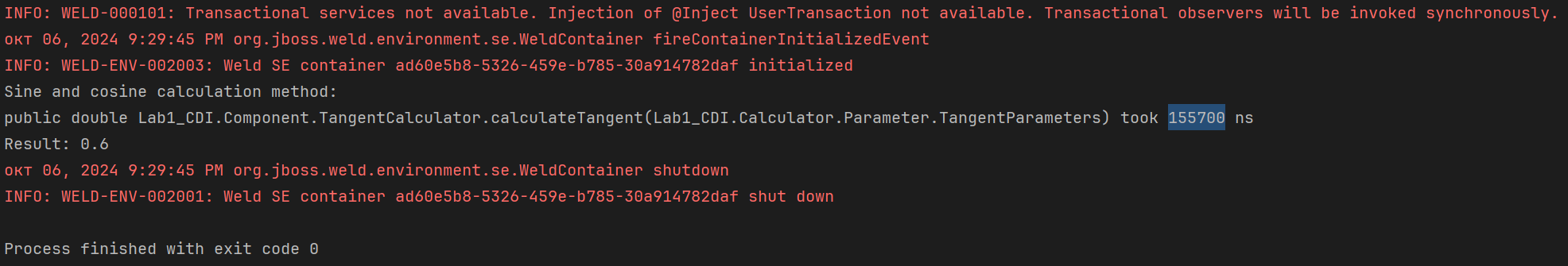


Рисунок 2 – Результат расчётов тангенса по синусу и косинусу

# **КОД ПРОГРАММЫ**

package Lab1\_CDI.Calculator.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 COSINE\_ONLY {  
}

package Lab1\_CDI.Calculator.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 SINE\_COSINE {  
}

package Lab1\_CDI.Calculator.Parameter;  
  
import lombok.Getter;  
  
import java.util.Collections;  
import java.util.List;  
  
@Getter  
public class CosineOnlyParameters<T extends Number> implements TangentParameters<T> {  
  
 private final T cosine;  
  
 public CosineOnlyParameters(T cosine) {  
 if (cosine.doubleValue() < -1 || cosine.doubleValue() > 1) {  
 throw new IllegalArgumentException("Cosine must be between -1 and 1.");  
 } else if (cosine.doubleValue() == 0) {  
 throw new ArithmeticException("Tangent is not determined when cosine is equal to zero");  
 }  
  
 this.cosine = cosine;  
 }  
  
 @Override  
 public List<T> getValues() {  
 return Collections.singletonList(cosine);  
 }  
}

package Lab1\_CDI.Calculator.Parameter;  
  
import lombok.Getter;  
  
import java.util.Arrays;  
import java.util.List;  
  
@Getter  
public class SineCosineParameters<T extends Number> implements TangentParameters<T> {  
  
 private final T sine;  
 private final T cosine;  
  
 public SineCosineParameters(T sine, T cosine) {  
 if (Math.abs(sine.doubleValue()) > 1 || Math.abs(cosine.doubleValue()) > 1) {  
 throw new IllegalArgumentException("Sine and cosine must be between -1 and 1.");  
 } else if (cosine.doubleValue() == 0) {  
 throw new ArithmeticException("Tangent is not determined when cosine is equal to zero");  
 }  
  
 this.sine = sine;  
 this.cosine = cosine;  
 }  
  
 @Override  
 public List<T> getValues() {  
 return Arrays.asList(sine, cosine);  
 }  
}

package Lab1\_CDI.Calculator.Parameter;  
  
import java.util.List;  
  
public interface TangentParameters<T> {  
 List<T> getValues();  
}

package Lab1\_CDI.Calculator;  
  
  
import Lab1\_CDI.Calculator.Annotation.COSINE\_ONLY;  
import Lab1\_CDI.Calculator.Parameter.TangentParameters;  
import lombok.NoArgsConstructor;  
  
import java.util.List;  
  
@COSINE\_ONLY  
@NoArgsConstructor  
public class CosineOnlyCalculator implements ITangentCalculator {  
  
 @Override  
 public <T extends Number> double calculate(TangentParameters<T> parameters) {  
  
 List<T> values = parameters.getValues();  
  
 return calculateTangent(values.get(0));  
 }  
  
 public <T extends Number> double calculateTangent(T cosine) {  
  
 // Logging  
 System.out.println("Cosine only tangent calculation method: ");  
  
 // Convert cosine to double for mathematical operations  
 double cosValue = cosine.doubleValue();  
  
 // sin = sqrt(1 - cos^2)  
 double sinValue = Math.sqrt(1 - cosValue \* cosValue);  
  
 // Calculate tangent  
 return sinValue / cosValue;  
 }  
  
}

package Lab1\_CDI.Calculator;  
  
import Lab1\_CDI.Calculator.Parameter.TangentParameters;  
  
public interface ITangentCalculator {  
 <T extends Number> double calculate(TangentParameters<T> parameters);  
}

package Lab1\_CDI.Calculator;  
  
import Lab1\_CDI.Calculator.Annotation.SINE\_COSINE;  
import Lab1\_CDI.Calculator.Parameter.TangentParameters;  
import lombok.NoArgsConstructor;  
  
import java.util.List;  
  
@SINE\_COSINE  
@NoArgsConstructor  
public class SineCosineCalculator implements ITangentCalculator {  
  
 @Override  
 public <T extends Number> double calculate(TangentParameters<T> parameters) {  
  
 List<T> values = parameters.getValues();  
  
 return calculateTangent(values.get(0), values.get(1));  
 }  
  
 public <T extends Number> double calculateTangent(T sine, T cosine) {  
  
 // Logging  
 System.out.println("Sine and cosine calculation method: ");  
  
 // Convert sine cosine to double for mathematical operations  
 double sinValue = sine.doubleValue();  
 double cosValue = cosine.doubleValue();  
  
 // Calculate tangent  
 return sinValue / cosValue;  
 }  
}

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.Calculator.Parameter.CosineOnlyParameters;  
import Lab1\_CDI.Calculator.Parameter.SineCosineParameters;  
import Lab1\_CDI.Calculator.Parameter.TangentParameters;  
import Lab1\_CDI.Component.TangentCalculator;  
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 {  
 TangentCalculator tangentCalculator = container.select(TangentCalculator.class).get();  
  
 TangentParameters<Double> parameters = new CosineOnlyParameters<Double>(0.3);  
  
// TangentParameters<Double> parameters = new SineCosineParameters<>(0.3, 0.5);  
  
 double tangent = tangentCalculator.calculateTangent(parameters);  
 System.out.println("Result: " + tangent);  
  
 } finally {  
 weld.shutdown();  
 }  
 }  
}