ФЕДЕРАЛЬНОЕ АГЕНСТВО ПО ОБРАЗОВАНИЮ

ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧЕРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМ. Р.Е. АЛЕКСЕЕВА

ИНСТИТУТ РАДИОЭЛЕКТРОНИКИ И ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Кафедра «Вычислительные системы и технологии»

Функциональное программирование

Отчет

по лабораторным работам

Выполнил студент группы 22-ПО

Соменков Ф.В.

Провел ст. преподаватель кафедры ВСТ

Тарасов А.В.

г. Нижний Новгород

2025

В данной работе реализован калькулятор, демонстрирующий различные подходы к вычислениям в функциональном стиле на Scala. Рассмотрим ключевые аспекты реализации.

**1. Tagless Final подход**

Основу архитектуры составляет трейт Calculator, параметризованный типом высшего порядка F[\_]:

trait Calculator[F[\_]] {

def add(a: Int, b: Int): F[Int]

def subtract(a: Int, b: Int): F[Int]

def multiply(a: Int, b: Int): F[Int]

def divide(a: Int, b: Int): F[Either[String, Int]] *// Ошибки*

}

Это позволяет абстрагироваться от конкретной реализации вычислений. Например, операция divide возвращает результат в контексте F, содержащий либо ошибку (Left), либо успешный результат (Right).

**2. Реализация интерпретаторов**

Немедленные вычисления

object ImmediateInterpreter extends Calculator[Id] {

override def add(a: Int, b: Int): Int = a + b

override def subtract(a: Int, b: Int): Int = a - b

override def multiply(a: Int, b: Int): Int = a \* b

override def divide(a: Int, b: Int): Either[String, Int] =

if (b == 0) Left("Division by zero") else Right(a / b)

}

type Id[A] = A

Здесь вычисления выполняются сразу при вызове операций. Тип Id[A] просто возвращает значение типа A без дополнительного контекста.

Ленивые вычисления

object LazyInterpreter extends Calculator[Lazy] {

override def add(a: Int, b: Int): () => Int = () => a + b

override def subtract(a: Int, b: Int): () => Int = () => a - b

override def multiply(a: Int, b: Int): () => Int = () => a \* b

override def divide(a: Int, b: Int): () => Either[String, Int] =

() => if (b == 0) Left("Division by zero") else Right(a / b)

}

type Lazy[A] = () => A

В этом случае каждая операция возвращает функцию, которая будет вычислена только при явном вызове.

**3. Примеры программ**

Программа с немедленными вычислениями

def programId(calc: Calculator[Id]): Either[String, Int] = {

val sum = calc.add(10, 5) *// 15*

val product = calc.multiply(sum, 2) *// 30*

calc.divide(product, 3) *// Right(10)*

}

Здесь все вычисления происходят сразу в момент вызова методов.

Программа с ленивыми вычислениями

def programLazy(calc: Calculator[Lazy]): () => Either[String, Int] = { () =>

{

val sum = calc.add(10, 5)() *// вычисляется только здесь*

val product = calc.multiply(sum, 2)() *// и здесь*

calc.divide(product, 3)() *// и здесь*

}

}

Вычисления происходят только при вызове конечной функции.

**4. Работа с ленивыми списками**

Демонстрация работы с бесконечной последовательностью:

val infiniteNumbers: LazyList[Int] = LazyList.from(1)

val processedNumbers = infiniteNumbers

.map { x =>

println(s"Обрабатываем $x")

x \* 2

}

.filter { x =>

println(s"Проверяем $x")

x % 3 == 0

}

*// Будет вычислено ровно столько элементов, сколько запрошено*

println(processedNumbers.take(3).toList)

Вывод в консоль покажет, что операции применяются лениво, только к необходимым элементам.

**5. Комбинированный пример**

Пример использования калькулятора с ленивыми списками:

val calculations = LazyList.from(1).take(5).map { n =>

val calc = LazyInterpreter

val sum = calc.add(n, n)() *// 2n*

val product = calc.multiply(sum, n)() *// 2n²*

product

}

println("Результаты: " + calculations.toList)

*// Выведет: Результаты: List(2, 8, 18, 32, 50)*

Анализ результатов

При выполнении программы можно наблюдать:

1. Для немедленных вычислений:

println(programId(ImmediateInterpreter)) *// Right(10)*

Все операции выполняются сразу.

1. Для ленивых вычислений:

println(programLazy(LazyInterpreter)()) *// Right(10)*

Вычисления происходят только при вызове ().

1. Для ленивых списков видно, что обработка элементов происходит только при реальной необходимости, что подтверждается выводами в консоль.

Весь код:  
import scala.language.higherKinds

import scala.util.Either

import scala.collection.immutable.LazyList

trait Calculator[F[\_]] {

  def add(a: Int, b: Int): F[Int]

  def subtract(a: Int, b: Int): F[Int]

  def multiply(a: Int, b: Int): F[Int]

  def divide(a: Int, b: Int): F[Either[String, Int]] // Ошибки

}

// Немедленные вычисления

object ImmediateInterpreter extends Calculator[Id] {

  override def add(a: Int, b: Int): Int = a + b

  override def subtract(a: Int, b: Int): Int = a - b

  override def multiply(a: Int, b: Int): Int = a \* b

  override def divide(a: Int, b: Int): Either[String, Int] =

    if (b == 0) Left("Division by zero") else Right(a / b)

}

// Ленивые вычисления

object LazyInterpreter extends Calculator[Lazy] {

  override def add(a: Int, b: Int): () => Int = () => a + b

  override def subtract(a: Int, b: Int): () => Int = () => a - b

  override def multiply(a: Int, b: Int): () => Int = () => a \* b

  override def divide(a: Int, b: Int): () => Either[String, Int] =

    () => if (b == 0) Left("Division by zero") else Right(a / b)

}

type Id[A] = A // Немедленные вычисления

type Lazy[A] = () => A // Ленивые вычисления

object Calculator extends App {

  // Для немедленных вычислений

  def programId(calc: Calculator[Id]): Either[String, Int] = {

    val sum = calc.add(10, 5)

    val product = calc.multiply(sum, 2)

    calc.divide(product, 3)

  }

  // Для ленивых вычислений

  def programLazy(calc: Calculator[Lazy]): () => Either[String, Int] = { () =>

    {

      val sum = calc.add(10, 5)()

      val product = calc.multiply(sum, 2)()

      calc.divide(product, 3)()

    }

  }

  println("Немедленные вычисления")

  val immediateResult: Either[String, Int] = programId(ImmediateInterpreter)

  println(immediateResult)

  println("\nЛенивые вычисления")

  val lazyResult: () => Either[String, Int] = programLazy(LazyInterpreter)

  println(lazyResult())

  println("\nДемонстрация ленивых списков:")

  val infiniteNumbers: LazyList[Int] = LazyList.from(1)

  val processedNumbers = infiniteNumbers

    .map { x =>

      println(s"Умножение к $x")

      x \* 2

    }

    .filter { x =>

      println(s"Фильтрую $x")

      x % 3 == 0

    }

  println("\nПервые 5 обработанных элементов:")

  val firstFive = processedNumbers.take(5)

  println(firstFive.toList)

  println("\nСледующие 3:")

  val nextThree = processedNumbers.drop(5).take(3)

  println(nextThree.toList)

  println("\nС калькулятором в списке:")

  val calculations = LazyList.from(1).take(5).map { n =>

    val calc = LazyInterpreter

    val sum = calc.add(n, n)()

    val product = calc.multiply(sum, n)()

    product

  }

  println("Результаты вычислений: " + calculations.toList)

}