Лабораторная работа № 3 «Обобщённые классы в Scala»

Скоробогатов С.Ю.

25 апреля 2021 г.

1 Цель работы

Целью данной работы является приобретение навыков разработки обобщённых классов на языке Scala с использованием неявных преобразований типов.

2 Задание

В ходе выполнения лабораторной работы требуется разработать обобщённый класс. Особенностью задания является то, что смысл некоторых операций над объектами разрабатываемого класса зависит от типа, которым этот класс параметризован. Более того, даже наличие некоторых операций может зависеть от типового параметра. Эта функциональность реализуется с помощью неявных объектов и неявных параметров методов.

2.1 Пример решения

Рассмотрим решение следующего варианта:

> Класс `EquationSystem[T]`, представляющий систему из двух линейных уравнений от двух переменных с коэффициентами типа `T` и операцией `solve`, возвращающей решение уравнения в виде `Option[(S, S)]`, где тип `S` зависит от типа `T`. Эта зависимость задаётся следующими правилами: если `T` — тип с плавающей точкой, то `S` совпадает с `T`; если `T` — целочисленный тип, то `S` — дробь, числитель и знаменатель которой имеют тип `T`; в остальных случаях операция `solve` недоступна. Для представления дробей следует создать отдельный класс.

Первый вариант решения (скачать)

Данное решение поддерживает только типы `Int` и `Float`:

```
class EquationSystem[T](a: ((T, T), (T, T)), b: (T, T)) {
  val ((a11, a12), (a21, a22)) = a
  val (b1, b2) = b

  def solve[S]()(implicit ops: EquationSystemOps[T, S]): Option[(S, S)] = {
    val d = ops.sub(ops.mul(a11, a22), ops.mul(a21, a12))
    if (ops.is_zero(d)) {
      None
    } else {
      val d1 = ops.sub(ops.mul(b1, a22), ops.mul(a12, b2))
      val d2 = ops.sub(ops.mul(a11, b2), ops.mul(b1, a21))
      Some((ops.div(d1, d), ops.div(d2, d)))
    }
  }
}

abstract class EquationSystemOps[T, S] {
  def mul(a: T, b: T): T
  def sub(a: T, b: T): T
  def div(a: T, b: T): S
  def is_zero(a: T): Boolean
}

class Ratio(val num: Int, val denom: Int) {
  override def toString: String =
    num.toString + "/" + denom.toString
}

object EquationSystemOps {
  implicit object float extends EquationSystemOps[Float, Float] {
    override def mul(a: Float, b: Float): Float = a * b
    override def sub(a: Float, b: Float): Float = a - b
    override def div(a: Float, b: Float): Float = a / b
    override def is_zero(a: Float): Boolean = a == 0
  }

  implicit object int extends EquationSystemOps[Int, Ratio] {
    override def mul(a: Int, b: Int): Int = a * b
    override def sub(a: Int, b: Int): Int = a - b
    override def div(a: Int, b: Int): Ratio = new Ratio(a, b)
    override def is_zero(a: Int): Boolean = a == 0
  }
}


object Main extends App {
  val s1 = new EquationSystem(((2, 1), (3, 2)), (5, 3))
  val s2 = new EquationSystem(((3, 4), (2, 5)), (18, 19))
  val s3 = new EquationSystem[Float](((2, 1), (3, 2)), (5, 3))
  val s4 = new EquationSystem(((2, 1), (4, 2)), (5, 3))
  println(s1.solve())
  println(s2.solve())
  println(s3.solve())
  println(s4.solve())

  val s5 = new EquationSystem((("a", "b"), ("c", "d")), ("e", "f"))
  /* Следующая строчка не компилируется */
  // println(s5.solve())
}
```

У него есть два недостатка:

- многословность — приходится явно описывать операции `mul`, `sub`, `is_zero` для встроенных типов, хотя в стандартной библиотеке есть готовые аналоги;
- он ограничен только типами `Int` и `Float`, хотя в задании требуется поддерживать любые целочисленные и любые вещественные типы.

Второй вариант решения (скачать)

Данное решение поддерживает все числовые типы стандартной библиотеки:

```
class EquationSystem[T](a: ((T, T), (T, T)), b: (T, T)) (implicit num: Numeric[T] = null) {
  val ((a11, a12), (a21, a22)) = a
  val (b1, b2) = b
  def solve[S]()(implicit ops: EquationSystemOps[T, S]): Option[(S, S)] = {
    val d = num.minus(num.times(a11, a22), num.times(a21, a12))
    if (num.equiv(d, num.zero)) {
      None
    } else {
      val d1 = num.minus(num.times(b1, a22), num.times(a12, b2))
      val d2 = num.minus(num.times(a11, b2), num.times(b1, a21))
      Some((ops.div(d1, d), ops.div(d2, d)))
    }
  }
}
trait EquationSystemOps[T, S] {
  def div(a: T, b: T): S
}
class Ratio[T](val num: T, val denom: T) {
  override def toString: String =
    num.toString + "/" + denom.toString
}
object EquationSystemOps {
  implicit def float_ops[T](implicit frac: Fractional[T]): EquationSystemOps[T, T] =
    new EquationSystemOps[T, T] {
      def div(a: T, b: T): T = frac.div(a, b)
    }
  implicit def int_ops[T](implicit integral: Integral[T]): EquationSystemOps[T, Ratio[T]] =
    new EquationSystemOps[T, Ratio[T]] {
      def div(a: T, b: T): Ratio[T] = new Ratio(a, b)
    }
}
object Main extends App {
  val s1 = new EquationSystem(((2, 1), (3, 2)), (5, 3))
  val s2 = new EquationSystem(((3, 4), (2, 5)), (18, 19))
  val s3 = new EquationSystem(((2.0, 1.0), (3.0, 2.0)), (5.0, 3.0))
  val s4 = new EquationSystem(((2, 1), (4, 2)), (5, 3))
  println(s1.solve())
  println(s2.solve())
  println(s3.solve())
  println(s4.solve())
  val s5 = new EquationSystem((("a", "b"), ("c", "d")), ("e", "f"))
  /* Следующая строчка не компилируется */
  // println(s5.solve())
}
```

Некоторые особенности:

- В задании говорится: «в остальных случаях операция `solve` недоступна». Поэтому мы используем запись `implicit num: Numeric[T] = null` для того, чтобы показать необязательность реализации `Numeric[T]` для параметра системы линейных уравнений.
- Неявные объекты могут быть не только объектами-*синглтонами*, но и функциями, возвращающими неявный объект. Неявные функции нужны, если неявный объект должен иметь параметры типа.
- Неявные функции возвращают безымянные классы, реализующие трейт `EquationSystemOps[T, S]`.

2.2 Индивидуальный вариант

Класс `ArithProgression[T]`, представляющий арифметическую прогрессию с элементами типа `T` и операцией вычисления *i*-го члена и суммы первых *n* членов. Арифметическая прогрессия должны быть задана начальным членом и разностью. В качестве типа `T` может выступать числовой тип или строка. В случае строки сложение означает конкатенацию.

3 Отчёт по лабораторной работе

Отчёт выполняется в разметке Markdown по следующему шаблону:

```
% Лабораторная работа № 3 «Обобщённые классы в Scala»
% 22 марта 2023 г.
% Вася Пупкин, ИУ9-63Б
# Цель работы
‹переписываете цель работы из задания›
# Индивидуальный вариант
‹переписываете индивидуальный вариант из задания›
# Реализация и тестирование
```scala
xxxxx
```

Вывод
‹пишете, чему научились›
```

В отчёте приведён лишь необходимый минимум. Можно писать больше и интереснее — интересные и вдумчивые отчёты поощряются дополнительным баллом.

Шаблон отчёта

Ваш отчёт будет конвертирован в PDF при помощи pandoc следующей командой:

```
pandoc \
  --pdf-engine=xelatex \
  -V 'mainfont:Liberation Serif' \
  -V 'monofont:Liberation Mono' \
  "$SOURCE" -o "$PDF"
```

Язык реализации: Markdown

Код решения

Из файла

Отправить