Лабораторная работа № 3 «Обобщённые классы в Scala»

22 марта 2023 г.

Яровикова Анастасия, ИУ9-61Б

Цель работы

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

Индивидуальный вариант

Класс Formula[T], представляющий формулу, состоящую из имён переменных, констант типа T и бинарных операций. Ассортимент операций зависит от типа T: если определён Numeric[T], то должны присутствовать четыре базовых арифметических операций; если T — это string, то должна присутствовать только операция сложения, обозначающая конкатенацию.

Для создания объектов класса Formula[T] доступны два конструктора: первый принимает имя переменной и создаёт формулу, состоящую из единственной переменной; второй конструктор принимает значение типа T и создаёт формулу, состоящую из единственной константы. Все остальные формулы возвращаются бинарными операциями.

В классе Formula[T] должен быть определён метод, принимающий отображение имён переменных в их значения и считающий значение формулы.

Реализация

Для выполнения поставленной задачи реализованы два класса: основной класс `Formula[T]` и вспомогательный класс `Variable`. Класс формулы реализован как обертка над функцией, принимающей отображение имен переменных в их значение и возвращающей значение формулы. Этот прием взят из лабораторной работы №2. Для реализации различного ассортимента операций в зависимости от типа T реализованы методы операций с неявными параметрами типа `Numeric[T]`, `FormulaOps[T]`, `DivOps[T]`. Первый принадлежит стандартной

библиотеке Scala, а последние два были реализованы вручную с помощью одноименных трейтов и объектов-компаньонов.

Код реализации:

```
class Variable(val name: String)

class Formula[T] private (s: Map[String, T] => T) {
  def this(varName: Variable) = this(varMap => varMap(varName.name))
  def this(value: T) = this(varMap => value)

  def eval(vars: Map[String, T]): T = s(vars)

  def +(other: Formula[T])(implicit op: FormulaOps[T]): Formula[T] = {
    new Formula(varMap => op.add(this.eval(varMap), other.eval(varMap)))
  }

  def -(other: Formula[T])(implicit num: Numeric[T]): Formula[T] = {
    new Formula(varMap => num.minus(this.eval(varMap), other.eval(varMap)))
  }

  def *(other: Formula[T])(implicit num: Numeric[T]): Formula[T] = {
    new Formula(varMap => num.times(this.eval(varMap), other.eval(varMap)))
  }

  def /(other: Formula[T])(implicit op: DivOps[T]): Formula[T] = {
    new Formula(varMap => op.div(this.eval(varMap), other.eval(varMap)))
  }
}

trait DivOps[T] {
  def div(a: T, b: T): T
}

object DivOps {
  implicit def float_ops[T](implicit frac: Fractional[T]): DivOps[T] =
    new DivOps[T] {
      def div(a: T, b: T): T = frac.div(a, b)
    }

  implicit def int_ops[T](implicit integral: Integral[T]): DivOps[T] =
    new DivOps[T] {
      def div(a: T, b: T): T = integral.quot(a, b)
    }
}
```

```scala
trait FormulaOps[T] {
  def add(a: T, b: T): T
}

object FormulaOps {
  implicit def num_ops[T](implicit num: Numeric[T]): FormulaOps[T] =
    new FormulaOps[T] {
      def add(a: T, b: T): T = num.plus(a, b)
    }

  implicit final val str_ops: FormulaOps[String] =
    new FormulaOps[String] {
      def add(a: String, b: String): String = a + b
    }
}

object Main extends App {
  val f1 = new Formula("abc")
  val d = new Variable("d")
  val f2 = new Formula[String](d)
  val f12 = f1 + f2
  println(f12.eval(Map.apply("abc" -> "abc", "d" -> "defg")))

  val a = new Variable("a")
  val f3 = new Formula[Long](a)
  val b = new Variable("b")
  val f4 = new Formula[Long](b)
  val f34 = f3 / f4
  println(f34.eval(Map.apply("a" -> 2_000L, "b" -> 2_00L)))

  val f5 = new Formula[Double](a)
  val f6 = new Formula[Double](b)
  val f56 = f5 / f6
  println(f56.eval(Map.apply("a" -> 256.12, "b" -> 16.06)))

  val c = new Variable("c")
  val f7 = new Formula[String](c)
  val f8 = new Formula[String](d)
  val f78 = f7 + f8
  println(f78.eval(Map.apply("c" -> "abra", "d" -> "cadabra")))

  val f91 = new Formula[Byte](a)
  val f92 = new Formula[Byte](b)
  val f93 = new Formula[Byte](c)
  val f9 = f91 * (f92 - f93)
  println(f9.eval(Map.apply("a" -> 7, "b" -> 6, "c" -> 2)))
```

```
  val fa = new Formula[Int](a)
  val fb = new Formula[Int](b)
  val fc = new Formula[Int](c)
  val fd = new Formula(4)
  val ff = fa * fa - fd * fb + fc
  println(ff.eval(Map.apply("a" -> 7, "b" -> 6, "c" -> 2)))
}
```

Тестирование

Для тестирования необходимо в объекте Main создать необходимые переменные класса `Variable` и класса `Formule`, далее произвести операции для создания желаемой формулы и вызвать метод `eval`, передав в него в качестве параметров значения переменных. Ниже представлены результаты тестирования формул, описанных в коде реализации:

```
// "abc" + d (d = "defg")
abcdefg
// a/b (a = 2_000L, b = 2_00L)
10
// a/b (a = 256.12, b = 16.06)
15.947696139476962
// c+d (c ="abra", d ="cadabra")
abracadabra
// a*(b-c) (a = 7, b = 6, c = 2)
28
// a*a - 4*b + c (a = 7, b = 6, c = 2)
27
```

Вывод

В ходе данной лабораторной работы были приобретены навыки разработки обощенных классов на языке Scala с использованием неявных преобразований типов. Также были изучены выразительные возможности языка Scala, позволяющие написать немногословный, но при этом поддерживающий строки и любые целочисленные и вещественные типы языка Scala, код.