Лабораторная работа №3
«Стек трейтов»

Скоробогатов С.Ю.

2 апреля 2016 г.

1 Цель работы

Целью данной работы является изучение реализации на языке Scala объектно-ориентированного образца проектирования «Decorator» через стек трейтов.

2 Исходные данные

В качестве исходных данных для данной лабораторной работы выступает исходный текст программы, приведённый на листинге 1.

Программа представляет собой заготовку лексического анализатора, спроектированного на основе образца «Decorator».

Класс `Pos` представляет позицию в тексте программы. Его поля `prog`, `offs`, `line` и `col` содержат текст программы, индекс текущей литеры в программе, номер строчки и номер колонки, соответственно. Метод `ch` возвращает код текущей литеры или -1, если достигнут конец текста программы. Метод `inc` возвращает новую позицию, получаемую путём сдвига по тексту программы вправо на одну литеру. Для создания объекта, представляющего позицию в самом начале текста программы, можно использовать конструктор, принимающий в качестве параметра текст программы:

```
val origin = new Pos("this is program")
```

Объект-синглетон `DomainTags` содержит набор констант типа `Tag`, маркирующих различные лексические домены. В рассматриваемой заготовке лексического анализатора присутствут только две константы: `WHITESPACE` (участок с пробельными символами) и `END_OF_PROGRAM` (тег псевдолексемы «конец программы»).

Класс `Scanner` является базовым классом для лексических анализаторов. Подразумевается, что его метод `scan`, предназначенный для распознавания лексемы, начинающейся с позиции `start`, будет переопределён в наследниках. Метод `scan` должен возвращать тег лексемы и позицию в тексте программы, непосредственно следующую за лексемой. Реализация метода `scan`, представленная в классе `Scanner` по умолчанию, выдаёт сообщение о синтаксической ошибке в позиции `start`.

Класс `Token` представляет собой итератор по токенам, описывающим последовательность лексем, прочитанную из текста программы. Его поля `tag`, `start` и `follow` содержат тег, позицию начала лексемы и позицию непосредственно после конца лексемы, соответственно. Метод

`image` возвращает строковое представление лексемы. Для получения токена, описывающего следующую лексему, можно использовать метод `next`. Создание объекта `Token` осуществляется путём вызова конструктора, принимающего два параметра: позицию начала лексемы и объект класса `Scanner`, который будет использоваться для распознавания лексем:

```
val firstToken = new Token(new Pos("this is program"), new Scanner)
```

Трейт `Whitespaces` – это декоратор, распознающий непустую последовательность пробельных символов. Он является наследником класса `Scanner` и переопределяет метод `scan` таким образом, чтобы в случае, если в текущей позиции текста программы не находится пробельный символ, вызывалась бы версия метода `scan` следующего класса в линеаризации класса, к которому трейт `Whitespaces` подмешан.

3 Задание

В лабораторной работе предлагается дополнить заготовку лексического анализатора трейтами, осуществляющими распознавание лексем, описанных в таблицах 1, 2 и 3. Кроме того, нужно реализовать в лексическом анализаторе восстановление при ошибках.

Для проверки работоспособности разработанного лексического анализатора нужно запустить программу вида

```
var t = new Token(
  new Pos("program source goes here!"),
  new Scanner
    with Numbers
    with Idents
    with Comments
    with Whitespaces
)
while (t.tag != END_OF_PROGRAM) {
  println(t.tag.toString + " " + t.start + "-" + t.follow + ": " + t.image)
  t = t.next
}
```

Здесь `Numbers`, `Idents`, `Comments` и `Whitespaces` – трейты-декораторы, подмешанные к классу `Scanner` для распознавания числовых литералов, идентификаторов, комментариев и пробелов, соответственно. Конкретный ассортимент декораторов определяется вариантом задания.

Алгоритм 1 Заготовка лексического анализатора

```
class Pos private(val prog: String, val offs: Int, val line: Int, val col: Int) {
  def this(prog: String) = this(prog, 0, 1, 1)
  def ch = if (offs == prog.length) -1 else prog(offs)
  def inc = ch match {
    case '\n' => new Pos(prog, offs+1, line+1, 1)
    case -1   => this
    case _    => new Pos(prog, offs+1, line, col+1)
  }
  override def toString = "(" + line + ", " + col + ")"
}

object DomainTags extends Enumeration {
  type Tag = Value
  val WHITESPACE, IDENT, NUMBER, END_OF_PROGRAM = Value
}
import DomainTags._

class Scanner {
  def scan(start: Pos): (Tag, Pos) =
    sys.error("syntax error at " + start)
}

class Token(val start: Pos, scanner: Scanner) {
  val (tag, follow) = start.ch match {
    case -1 => (END_OF_PROGRAM, start)
    case _  => scanner.scan(start)
  }

  def image = start.prog.substring(start.offs, follow.offs)
  def next = new Token(follow, scanner)
}

trait Whitespaces extends Scanner {
  private def missWhitespace(pos: Pos): Pos = pos.ch match {
    case ' '  => missWhitespace(pos.inc)
    case '\t' => missWhitespace(pos.inc)
    case '\n' => missWhitespace(pos.inc)
    case _    => pos
  }

  override def scan(start: Pos) = {
    val follow = missWhitespace(start)
    if (start != follow) (WHITESPACE, follow)
    else super.scan(start)
  }
}
```

Таблица 1: Краткое описание лексики вариантов языков

1	Идентификаторы: последовательности латинских букв, начинающиеся с гласной буквы. Числовые литералы: последовательности десятичных цифр, перед которыми может стоять знак «минус». Операции: «−−», «<», «<=».
2	Комментарии: начинаются с «/*», заканчиваются на «*/» и могут пересекать границы строк текста. Идентификаторы: последовательности латинских букв и десятичных цифр, в которых буквы и цифры чередуются. Ключевые слова: «for», «if», «m1».
3	Строковые литералы: ограничены апострофами, для включения апострофа в литерал он удваивается, не пересекают границы строк текста. Числовые литералы: последовательности десятичных цифр, которые могут включать точку и предваряться знаком «минус». Идентификаторы: последовательности буквенных символов Unicode, точек и цифр, начинающиеся с буквы.
4	Идентификаторы: либо последовательности латинских букв, либо непустые последовательности десятичных цифр, ограниченные круглыми скобками. Числовые литералы: либо последовательности десятичных цифр, не начинающиеся с нуля, либо «0». Операции: «()», «:», «:=».
5	Комментарии: начинаются с «//» и продолжаются до окончания строки текста. Идентификаторы: любой текст, не содержащий «/» и ограниченный символами «/». Ключевые слова: «/while/», «/do/», «/end/».
6	Строковые литералы: ограничены двойными кавычками, могут содержать Escape-последовательности «\"», «\n» и «\t», не пересекают границы строк текста. Числовые литералы: последовательности десятичных цифр, разбитые точками на группы по три цифры («100», «1.000», «1.000.000»). Идентификаторы: последовательности буквенных символов Unicode, знаков подчёркивания и цифр, начинающиеся с буквы или подчёркивания.
7	Идентификаторы: последовательности латинских букв и десятичных цифр, оканчивающиеся на цифру. Числовые литералы: последовательности десятичных цифр, органиченные знаками «<» и «>». Операции: «<=», «=», «==».

Таблица 2: Краткое описание лексики вариантов языков (продолжение)

8	Комментарии: целиком строка текста, начинающаяся с «*». Идентификаторы: либо последовательности латинских букв нечётной длины, либо последовательности символов «*». Ключевые слова: «with», «end», «***».
9	Строковые литералы: органичены двойными кавычками, для включения двойной кавычки она удваивается, для продолжения литерала на следующей строчке текста в конце текущей строчки ставится знак «\». Числовые литералы: либо последовательности десятичных цифр, либо последовательности шестнадцатеричных цифр, начинающиеся с «$». Идентификаторы: последовательности буквенных символов Unicode, цифр и знаков «$», начинающиеся с буквы.
10	Идентификаторы: последовательности заглавных латинских букв, за которыми могут располагаться последовательности знаков «+», «−» и «*». Числовые литералы: знак «*» или последовательности, состоящие целиком либо из знаков «+», либо из знаков «−». Ключевые слова: «ON», «OFF», «***».
11	Комментарии: начинаются с «(*» или «{», заканчиваются на «*)» или «}» и могут пересекать границы строк текста. Идентификаторы: последовательности латинских букв, представляющие собой конкатенации двух одинаковых слов («zz», «abab»). Ключевые слова: « ifif », «do», «dodo».
12	Строковые литералы: ограничены обратными кавычками, могут занимать несколько строчек текста, для включения обратной кавычки она удваивается. Числовые литералы: десятичные литералы представляют собой последовательности десятичных цифр, двоичные – последовательности нулей и единиц, оканчивающиеся буквой «b». Идентификаторы: последовательности десятичных цифр и знаков «?», «*» и «|», не начинающиеся с цифры.
13	Идентификаторы: последовательности буквенных символов Unicode и десятичных цифр, начинающиеся и заканчивающиеся на одну и ту же букву. Числовые литералы: последовательности шестнадцатеричных цифр (чтобы литерал не был похож на идентификатор, его можно предварять нулём). Ключевые слова: «qeq», «xx», «xxx».
14	Символьные литералы: ограничены апострофами, могут содержать Escape-последовательности «\'», «\n» и «\xxxx» (в последней Escape-последовательности буквы «x» обозначают шестнадцатеричные цифры). Идентификаторы: последовательности символов Unicode длиной от 2 до 10 символов, начинающиеся и заканчивающиеся буквой. Ключевые слова: «z», «for», «forward».

Таблица 3: Краткое описание лексики вариантов языков (продолжение)

15	Идентификаторы: начинаются с латинской буквы, за которой в круглых скобках следует непустая последовательность десятичных цифр (например, «i(1)», «A(50)»). Числовые литералы: последовательности десятичных цифр, в которых группы по три разряда разделены точками (например, «10.000.000», «999»). Комментарии: последовательности символов, не содержащие «−*», заключённые в «*−» и «−*». Ключевые слова: «Y(0)», «IF(1)», «DO(1)».
16	Идентификаторы: начинаются с латинской буквы «s», «t» или «v», за которой через точку следует непустая последовательность букв и цифр (например, «s.1», «t.a1»). Комментарии: строчка программы целиком считается комментарием, если в первой позиции в ней стоит «*». Числовые константы: непустые последовательности десятичных цифр. Строковые константы: произвольные последовательности символов, заключённые в апострофы (если строковая константа должна содержать апостроф, он удваивается). Ключевые слова: «$ENTRY», «$EXTERN».
17	Идентификаторы: последовательности латинских букв и десятичных цифр, начинающиеся с буквы, после которых через знак подчёркивания может следовать последовательность из символов «+», «−», «*», «/» и «!» (например, «alpha1», «unary_!»). Числовые константы: непустые последовательности десятичных цифр или одиночные символы ASCII, заключённые в апострофы. Комментарии: как в языке C++. Ключевые слова: «class», «if», «for».
18	Идентификаторы: произвольные последовательности латинских букв и десятичных цифр, в которых есть хотя бы одна буква. Числовые константы: непустые последовательности десятичных цифр. Строковые литералы: произвольные последовательности символов, начинающиеся со знака «%» и продолжающиеся до конца строчки программы. Ключевые слова: «while», «unless», «for».