Знакомство со Scala

Восьмое занятие

API: стандартные Scala классы

- Функции
- Option
- Кортежи
- Коллекции
- PartialFunction
- XML

API: case class-ы

- Создание без new, удобно когда много

```
val e = Expr(Add(Var("a"), Number("5")))
```

- toString, equals, hashCode
- Экстракторы!

```
e match {
  case Expr(Add(Var(_), Number(n))) =>
}
```

API: Именованные аргументы и аргументы по умолчанию

```
def img(
  file: String,
  prefix: String = "/images/",
  alt: String = "o",
  size: Int = 16,
  flt: Option[String] = None): NodeSeq = {…}
```

```
<a href="#">{img("ok.png")}</a>
<a href="#">{img("back.png", alt = "back", size = 22)}</a>
```

API: Управляющие конструкции

```
def withPrintWriter(file: File)(op: PrintWriter => Unit) {
  val writer = new PrintWriter(file)
  try {
    op(writer)
  } finally {
    writer.close()
  }
}

withPrintWriter(new File("date.txt")) {
  writer => writer.println(new java.util.Date)
}
```

API: Traits

```
trait Logging {
  protected val log = LoggerFactory.getLogger(getClass)
}
class Handler {…}
trait AdminRightsRequired {
  this: Handler =>
  // …
}
trait TransactionRequired {
  this: Handler =>
  // …
}

class MyHandler extends Handler
  with Logging
  with AdminRightsRequired
  with TransactionRequired
```

API: Экстракторы

```
object EMail {
  def unapply(str: String): Option[(String, String)] = {
    val parts = str split "@"
    if (parts.length == 2) Some(parts(0), parts(1))
    else None
  }
}

def printEmail(s:String) = s match {
  case EMail(user, domain) => println(user+" AT "+domain)
  case _ => println("not an email address")
}
```

API: Неявные преобразования

- Мы уже знаем, как сделать: `1.day + 5.hours`
- Ещё пример: Foursquare Rogue

```
val query = Venue where (_.mayor eqs 1) and (_.tags
contains "karaoke")
val result = query.fetch()
```

- Ещё пример: Lift

```
"#line *" #> List("a", "b", "c") &
"#link [href]" #> "http://dogscape.com" &
"span [class+]" #> "error"
```

API: Мощная типизация

```
sealed trait NoFuel
sealed trait Fueled
sealed trait NoO2
sealed trait HasO2

final case class Rocket[Fuel, O2] private[RocketModule]()
def createRocket = Rocket[NoFuel, NoO2]()
def addFuel[O2](x : Rocket[NoFuel, O2]) = Rocket[Fueled, O2]()
def addO2[Fuel](x : Rocket[Fuel, NoO2]) = Rocket[Fuel, HasO2]()
def launch(x : Rocket[Fueled, HasO2]) = "blastoff"
implicit def toPiped[V] (value:V) = new {
   def |>[R] (f : V => R) = f(value)
}

def test1 = createRocket |> addFuel |> addO2 |> launch
def test2 = createRocket |> addO2 |> addFuel |> launch
// def test3 = createRocket |> addO2 |> launch
// def test4 = createRocket |> addFuel |> launch
// def test5 = createRocket |> addFuel |> addO2 |> addFuel |> launch
```

http://james-iry.blogspot.com/2010/10/phantom-types-in-haskell-and-scala.html

Задание 8-1

Написать экстракторы для целых чисел:

```
i match {
  case Sqr(x) => // является квадратом x
  case AsDigits(List(1, 2, 3)) => // цифры числа
}
```

Написать экстракторы для списков:

```
list match {
  case Sum(x) => // сумма элементов списка равна x
  case Even(Set()) => // чётные элементы
}
```

Пример комбинации: `AsDigits(1 :: 2 :: Sum(x))`

Example: formatters API

```
// Java
NumberFormat fmt = NumberFormat.getInstance();
fmt.setGroupingUsed(true);
fmt.setMinimumFractionDigits(2);
System.out.println(“$”+fmt.format(12345678.90));

// Scala
import FormatUtils._
println(.012345678 as percents)
println(2.01234567 as currency)

println(parse("56.02%"))
println(parse("$122,224,356.02"))
```

Example: formatters API

```
abstract class Format {
   def format(value: Double): String
}

 class Formatted(num: Double) {
   def as(fmt: Format) = fmt.format(num)
 }

implicit def covert2Formatted(i: Double) = new
Formatted(i)
```

Example: formatters API

```
object Currency extends Format {
   val fmt = {
     val nf = NumberFormat.getInstance(Locale.US)
     nf.setMaximumFractionDigits(2)
     nf.setMinimumFractionDigits(2)
     nf.setGroupingUsed(true)
     nf
   }

   def format(value: Double): String = "$" + fmt.format(value)

   def unapply(str: String): Option[Double] =
     if (str.startsWith("$"))
       try {
       Some(fmt.parse(str.substring(1)).doubleValue)
       } catch {
         case _ => None
       }
     else None
 }
```

Example: formatters API

```
object Percents extends Format {
   val fmt = {
     val nf = NumberFormat.getInstance(Locale.US)
     nf.setMaximumFractionDigits(2); nf.setMinimumFractionDigits(2)
     nf
   }

   def format(value: Double): String =
     if (value > 1) "100.00%"
     else if (value < 0) "0.00%"
     else fmt.format(value * 100) + "%"

   def unapply(str: String): Option[Double] =
     if (str.endsWith("%"))
       try {
        Some(fmt.parse(str.substring(0, str.length() - 1)).doubleValue /
100.0)
       } catch { case _ => None }
     else None
 }
```

Example: formatters API

```
object FormatUtils {
   val percents = Percents
   val currency = Currency

   def parse(str: String) = {
     str match {
       case Percents(num) => num
       case Currency(num) => num
       case _ => 0
     }
   }

   implicit def covert2Formatted(i: Double) = new
Formatted(i)
  }
```

Example: formatters API

```
def main(strs: Array[String]) {
  import FormatUtils._

  println(.012345678 as percents) // 1.23%
  println(2.01 as percents)       // 100.00%
  println(2.01234567 as currency) // $2.01
  println(123456789 as currency)  //$123,456,789.00

  println(parse("56.02%"))        // 0.5602
  println(parse("$1,224,356.02")) // 1224356.02
  println(parse("$NaN"))          //  0.0
}
```