Broaden Scala

André van Delft

Scala

SubScript

SlideMight

Lambda Days - Krakow

22..23 February 2018

Overview

1 - Notation Matters

1980: C

```
printf("%10.2f", x);
```

1988: C++

```
cout<< setw(10)<< setprecision(2)<< fixed<< x;
```

1996: Java

```
java.text.NumberFormat formatter
  = java.text.NumberFormat.getNumberInstance();
formatter.setMinimumFractionDigits(2);
formatter.setMaximumFractionDigits(2);
String s = formatter.format(x);
for (int i = s.length(); i < 10; i++)
   System.out.print(' ');
System.out.print(s);
```

2004: Java

```
System.out.printf("%10.2f", x);
```

2008: Scala&Groovy

```
printf("%10.2f", x)
```

2012: Scala 2.10

```
println(f"$x%10.2f")
```

Source: The March of Progress - www.horstmann.com

1.1 - GUI Controller - Scala

```
val searchButton = new Button("Go”) {
  reactions.+= {
    case ButtonClicked(b) =>
      enabled = false
      outputTA.text = "Starting search...”
      new Thread(new Runnable {
       def run() {
        Thread.sleep(3000)
        SwingUtilities.invokeLater(new Runnable{
          def run() {outputTA.text="Search ready”
                     enabled = true
        }})
      }}).start
  }
}
```

1.2 - GUI Controller - SubScript DSL

```
def liveScript = DSL._script[Any](None, Symbol("liveScript"))
                 {  (_node: subscript.vm.ScriptTrait[Any])
                 => implicit val script = _node
                    DSL._seq(
                      DSL._loop,
                      DSL._maybeCall("", (here: CallGraphTreeNode)
                            => DSL._maybeVarCall("searchSequence"))
                    )
                 }

def searchSequence = DSL._script[Any](None, Symbol("searchSequence"))
            {  (_node: subscript.vm.ScriptTrait[Any])
            => implicit val script = _node
               DSL._seq(
                 DSL._maybeCall("", (here: CallGraphTreeNode) => ...),
                 DSL._maybeCall("", (here: CallGraphTreeNode) => ...),
                 DSL._maybeCall("", (here: CallGraphTreeNode) => ...),
                 DSL._maybeCall("", (here: CallGraphTreeNode) => ...))
            }
```

1.3 - GUI Controller - SubScript syntax

```
script ..

  live              = searchSequence...

  searchSequence    = searchCommand
                      showSearchingText
                      searchInDatabase
                      showSearchResults

  searchCommand     = searchButton
  showSearchingText = @gui: {:outputTA.text = "…":}
  showSearchResults = @gui: {:outputTA.text = "…":}
  searchInDatabase  = {* Thread.sleep(3000) *}
```

1.4 - Parboiled2 - Scala

```
def TraitDef    : R1 = rule( `trait` ~ Id
                           ~ (TypeArgList.? ~> ExtractOpt)
                           ~ TraitTmplOpt ~> Concat4
                           )

def TraitTmplOpt: R1 = rule(  `extends`    ~ TraitTmpl ~> Concat
                           | (`extends`.? ~> ExtractOpt
                             ~ TmplBody   ~> Concat).? ~> ExtractOpt
                           )

def TraitParents: R1 = rule( AnnotType
                           ~ ( (`with` ~ AnnotType ~> Concat).*
                               ~> ConcatSeqNoDelim)
                           ~> Concat
                           )

def TraitTmpl   : R1 = rule( EarlyDefs.? ~> ExtractOpt
                           ~ TraitParents
                           ~  (TmplBody.? ~> ExtractOpt)
                           ~> Concat3
                           )
```

1.5 - Parboiled2 - Future?

```
Rule_1 ..

  TraitDef     = `trait` Id TypeArgList? TraitTmplOpt

  TraitTmplOpt = (`extends` TraitTmpl | `extends`? TmplBody)?

  TraitParents = AnnotType ( (`with` AnnotType)* ~> ConcatSeqNoDelim)

  TraitTmpl    = EarlyDefs? TraitParents TmplBody?
```

2 - Open up Scala

JVM >> Java

- Common platform for many languages.
- Scala could become common base language

Open up compiler frontend

- Like Flexmark
- ScalaParse
- `import language.mydialect`

Risks

- Pandora's box ~ Babylonian confusion

Benefits

- Evolution ~ Survival of the fittest
- Standardisation on Scala

3 - DSLs or Extensions

Internal DSL

- syntax within Scala limits

External DSL

- Requires special translator
- no mix with Scala

Language Extension

- Preprocessor or compiler plugins for parser & scanner
- Mutual embedding
- Language borders: `script`, `[]`, `()`, `{}`

4 - Wild Ideas

4.1 - Factor out "def" etc

```
def m1 = ???
def m2 = ???

def ...
  m1 = ???
  m2 = ???
```

```
sealed trait Statement
case class Print (name: String              ) extends Statement
case class Assign(name: String, value: Int  ) extends Statement
case class IfZero(name: String, thenn: Statement,
                  els: Option[Statement]    ) extends Statement
case class Block(statements: List[Statement]) extends Statement
```

```
sealed trait Statement
case class ... extends Statement
  Print (name: String)
  Assign(name: String, value: Int)
  IfZero(name: String, thenn: Statement, els: Option[Statement])
  Block(statements: List[Statement])
```

4.2 - Text Macros

```
case Sequence(Nil              ) => Success()
case Sequence(Sequence(a) :: x) => Sequence(a ++ x)
case Sequence(Call(t)     :: x) => Sequence(t() :: x)
case Sequence(Success()   :: x) => Sequence(x)
```

```
...
  case Sequence(`Params`) => `Result`

  Params            | Result
  ------------------+---------
  Nil               | Success()
  Sequence(a) :: x  | Sequence(a ++ x)
  Call(t)     :: x  | Sequence(t() :: x)
  Success()   :: x  | Sequence(x)
```

```
...
  case Sequence(`Nil`) => `Success()`

  Nil               | Success()
  Sequence(a) :: x  | Sequence(a ++ x)
  Call(t)     :: x  | Sequence(t() :: x)
  Success()   :: x  | Sequence(x)
```

4.3 - Parentheses >> layout

SIP-12

```
if x < y
then {
  a1
  a2
}
else {
  b1
  b2
}
```

Python style for Scala - Li Haoyi

```
if x < y
then
  a1
  a2
else
  b1
  b2
```

4.4 - Function call syntax

Smalltalk

```
a > b
ifTrue: [ 'greater' ]
ifFalse:[ 'less or equal' ]
```

SubScript

```
doWithMessage(fetchInformation, "Please wait...")

do: fetchInformation, withMessage: "Please wait..."
```

For substring

```
s.substring(3, 8)
s.substringFrom: 3 to: 8
s.substringFrom: 3 length: 5
```

4.5 - Arrows: data flow

```
def compute: Int = ???

compute match {
  case 0       => println: "none found"
  case i:Int   => println: s"$i found"
}
def show = {
  case 0       => println: "none found"
  case i:Int   => println: s"$i found"
}
show(compute)



compute ~(0    )~> println: "none found"
      , ~(i:Int)~> println: s"$i found"



show = ~(0    )~> println: "none found"
     , ~(i:Int)~> println: s"$i found"

compute ~> show
```

4.6 - Arrows: data+exception flow

```
{ try compute
  catch {case e: IOException => FAIL->e}
}
match {
  case (FAIL,e:IOException) => e.printStackTrace
  case 0                    => println: "none found"
  case i:Int                => println: s"$i found"
}


compute ~(0             )~> println: "none found"
      , ~(i:Int         )~> println: s"$i found"
      , ~/e:IOException/~> e.printStackTrace



show =  ~(0             )~> println: "none found"
      , ~(i:Int         )~> println: s"$i found"
      , ~/e:IOException/~> e.printStackTrace

compute ~> show
```

4.7 - Binary (?) Operators

- n-ary operators
- new line, space and ';' as operators
- drop infix notation: `1 to 10`

Prefix notation

```
special =  "break" + "break?" + "..." + "..?"
special =+ "break"   "break?"   "..."   "..?"

simpleTerm = number
           + string
           + "(" expression ")"

simpleTerm =;+ number
               string
               "(" expression ")"
```

4.8 - Drop Infix calls1

```
for (i <- 1 to    n)
for (i <- 1 until n by 2)
```

- Make spaces available for sequence etc
- Allow symbols starting with '..'

```
for (i <- 1 ..<=.. n)
for (i <- 1 ..<..  n.by:2)
for (i <- n ..>=.. 1.by:2)
for (i <- n ..>..  1)
```

Conclusion

Let's make coding better!
See also [languageengineering.io]

1	Notation Matters
2	Open up Scala
3	DSLs or Extensions
4	Wild Ideas

Made by SlideMight

Data file:

```
Title  : Broaden Scala
Author : André van Delft
Remarks: |
           Scala
           SubScript
           SlideMight
Event   : Lambda Days
Location: Krakow
Date    : 22..23 February 2018

ConclusionMsg: |
     Let's make coding better!
     See also [languageengineering.io](languageengineering.io)

MainSlides: |

  # Notations: conciser, clearer

      1980: C             printf("%10.2f", x);

      1988: C++           cout<< setw(10)<< setprecision(2)<< fixed<< x;

      1996: Java          java.text.NumberFormat formatter
                            = java.text.NumberFormat.getNumberInstance();
                          formatter.setMinimumFractionDigits(2);
                          formatter.setMaximumFractionDigits(2);
                          String s = format[illegible]rma[illegible];
                          for (int i = s.leng[illegible]; i[illegible]0; i++)
```