The Scala Programming

Vol - 2

Bite-sized introductions to the most frequently used features of Scala.

Agenda

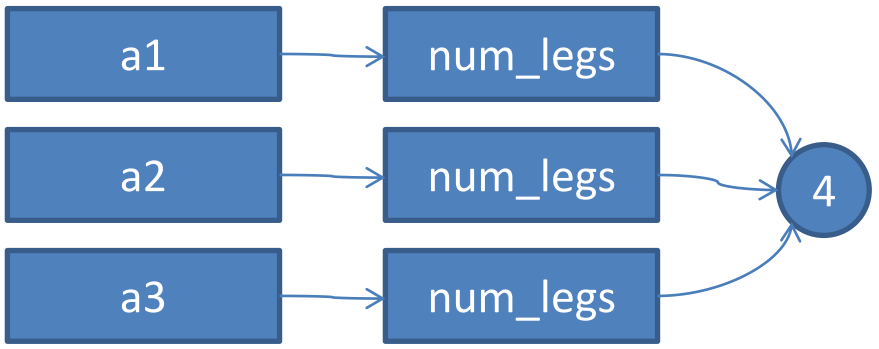
- String interpolation
- Classes
- Objects
- Companion object
- Case Classes
- Traits
- Pattern Matching
- Case Objects
- Implicit Parameters and Conversions

String interpolation

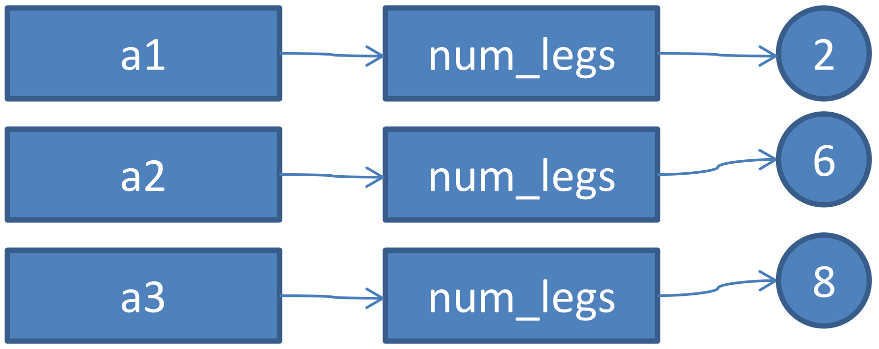
```
val i = 100.545866705

//Substitute Variable values
val str = s"Value Of i = $i"

println(str)

//Formatted Printing
println(f"Value Of i = $i%.4f")

//Prints An symbol within the String.
println(raw"Value Of i = $i%.4f")
```

Classes

```
class Animal{
  var num_legs = 4
  private val num_eyes = 2
}
val a1 = new Animal()
val a2 = new Animal()
val a3 = new Animal()
```



```
a1.num_legs = 2
a2.num_legs = 6
a2.num_legs = 8
```



Classes ...Contd

```
class Animal(var name: String) {
  val a = 100;

  def getA = a;

  //Validation
  require(name.startsWith("A"))

  //Constructor
  /*private*/ def this() = this("As")
  def this(a: String, b: String) = this(a)
  var num_legs = 4

  def make_noise: Unit = println("I don't know what does that
mean.")

  override def toString = s"a : $a getA : $getA num_legs :
$num_legs make_noise : $make_noise"
}

class Dog(name: String) extends Animal(name) {
  override def make_noise: Unit = { println("Dog : Bho bho") }
}
```

```
class Cat(name: String) extends Animal(name) {
  override def make_noise: Unit = { println("Cat : Meau") }
}

val a1: Animal = new Dog("AA")
val a2: Cat = new Cat("AB")
val a3 = new Animal("AC")
val a4 = new Animal()

a1.make_noise
a2.make_noise
a3.make_noise
a4.make_noise

println(a1.a)
println(a4.a)

a1.name = "2"
println(a1)
println(a2)
println(a3)
println(a4)
```

Classes ...Contd

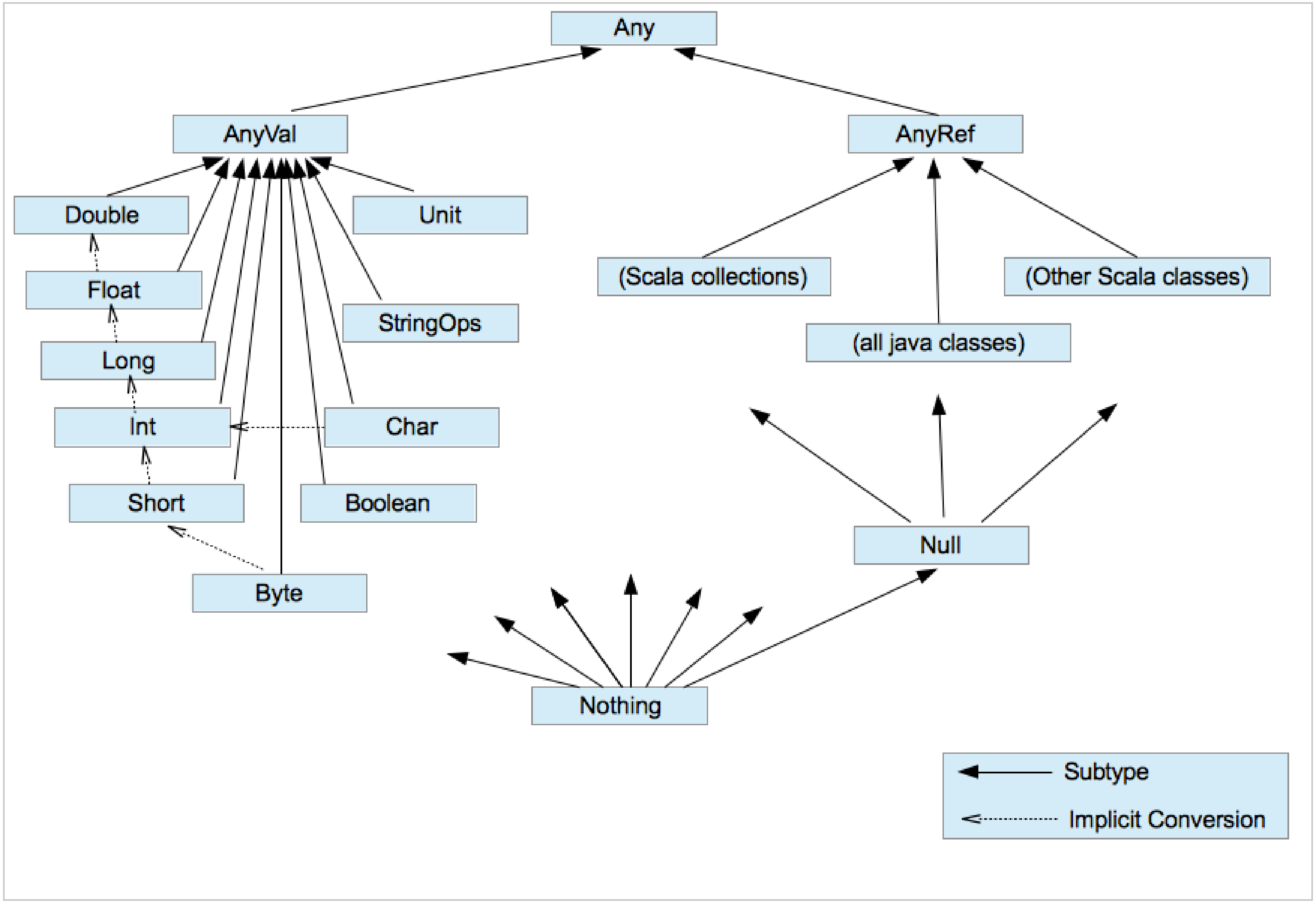
```
class Rational(n: Int, d: Int) {

  //Won't work
  //def add(that: Rational) = new Rational(n * that.d + that.n * d, d * that.d)

  require(d != 0)

  val numer = n
  val denom = d

  def +(that: Rational) = new Rational(numer * that.denom + that.numer * denom, denom * that.denom)

  override def toString : String = s"$n / $d"
}

val r1 = new Rational(10,3)
val r2 = new Rational(10,3)
```

```
println (s"$r1 + $r2 = ${r1 + r2}")
```

Objects

```
object Logger {
  var line_num = 0
  def log(s: String) = { println(s"$line_num : $s"); line_num += 1 }
 }

 Logger.log("This is a start of program")

 Logger.log("Arguments are : a, b, c")

 Logger.log("Calculating a")

 Logger.log("Performing EOD procedures")

 Logger.log("This is a end of program")
```

Data Types


Any
AnyVal
AnyRef
Double
Unit
Float
StringOps
(Scala collections)
(Other Scala classes)
Long
(all java classes)
Int
Char
Short
Boolean
Null
Byte
Nothing
Subtype
Implicit Conversion

Data Types …contd

```
  class A1(val a: Int) extends AnyRef
{
  }

  class A2(val a: Int) extends AnyVal
{
  }

  val a1_1 = new A1(1)
  val a1_2 = new A1(1)

  val a2_1 = new A2(1)
  val a2_2 = new A2(1)

  if (a1_1 == a1_2) println("Equal")
else println("Not Equal")
  if (a2_1 == a2_2) println("Equal")
else println("Not Equal")

  //Reference comparison only for
AnyRef tree
  if (a1_1 eq a1_2) println("Equal")
else println("Not Equal")

  //object value comparison only for
AnyVal tree. Below won't compile
  //if (a2_1 eq a2_2) println("Equal")
else println("Not Equal")
```

Companion objects

```
class C1(a: String, b: Int) {
  def getA = this.a
  def getB = b
  override def toString : String = s"a : $a, b : $b"
}

object C1 {
  def apply(): C1 = new C1("Str1", 5)
  def apply(str: String): C1 = new C1(str, 5)
  def apply(intVal: Int): C1 = new C1("Str1", intVal)

}

val a: C1 = C1()
val b: C1 = C1("XYZ")
val c: C1 = C1(100)

println(a)
println(b)
println(c)
```

Case Classes

A Packed Class with

- Equality
- Nice toString
- Getters and Setters

```
case class CoOrdinates(val x: Int, val y: Int, var z: Int)

val p1 = CoOrdinates(10, 20, 30)

val neighbour_of_p1 = p1.copy(y = p1.y + 1)

println(p1)
println(neighbour_of_p1)

p1.z = 31

println(p1)
println(neighbour_of_p1)
```

Traits

Used to share interfaces and attributes between classes.

```
trait Color {
  val r: Int; val g: Int; val b: Int
  def paint = println(s"Painting with Color : RGB($r,$g,$b)")
}

trait Style {
  val size: Int; val bold: Boolean; val italic: Boolean = false
  def applyStyle = println(s"Setting Style => (Size : $size, Bold : $bold, Italic : $italic)")
}

class FontColor(r_ : Int, g_ : Int, b_ : Int) extends Color {
  val r = r_; val g = g_; val b = b_
  override def paint = println(s"Setting Pen Color to : RGB($r,$g,$b)")
}

class BgColor(r_ : Int, g_ : Int, b_ : Int) extends Color {
  val r = r_; val g = g_; val b = b_
}

class ColorAndStyle(r_ : Int, g_ : Int, b_ : Int, bold_ : Boolean, size_ : Int, italic_ : Boolean) extends Color with Style {
  val r = r_; val g = g_; val b = b_; val bold = bold_; val size = size_; override val italic = italic_;
}

val backGround = new BgColor(10, 20, 30); backGround.paint

val cursorStyle = new FontColor(10, 20, 30) with Style() { val bold: Boolean = true; val size: Int = 20 }
cursorStyle.paint; cursorStyle.applyStyle
new ColorAndStyle(10, 20, 30, bold_ = true, 20, italic_ = true)
```

Pattern Matching

A mechanism for checking a value against a pattern

```
import scala.util.Random

val x: Int = Random.nextInt(10)

x match {
  case 0 => "zero"
  case 1 => "one"
  case 2 => "two"
  case _ => "many"
}
```

```
abstract class Device
case class Phone(model: String) extends Device {
  def screenOff = "Turning screen off"
}
case class Computer(model: String) extends Device {
  def screenSaverOn = "Turning screen saver on..."
}

def goIdle(device: Device) = device match {
  case p: Phone => p.screenOff
  case c: Computer => c.screenSaverOn
  case _ => "Unknown Device"
}

println(goIdle(new Phone("Avaya")))
println(goIdle(new Computer("HP")))
println(goIdle(new Device(){}))
```

Case Objects

A Packed Object with

- Equality
- Nice toString
- Getters and Setters
- Mostly used in Pattern matching

```
trait Dimension
case class Dimension_2(x: Int, y: Int) extends Dimension
case class Dimension_1(x: Int) extends Dimension // Case class
case object Dimension_0 extends Dimension // Case object

def callCase(f: Dimension) = f match {
  case Dimension_2(f, g) => println("2 D CoOrdinates - x = " + f + " y =" + g)
  case Dimension_1(f) => println("1 D CoOrdinates = " + f)
  case Dimension_0 => println("Dimension 0")
}

callCase(Dimension_2(10, 10))
callCase(Dimension_1(10))
callCase(Dimension_0)
```

Implicit Parameters and Conversions

A way to pass parameters without specifying explicitly

```
  implicit val pi = 3.14

  def area_of_circle(radius: Int)(implicit value_of_pi: Double) =
value_of_pi * radius * radius

  println(s"Area of Circle = ${area_of_circle(5)}")
```

```
  case class CoOrdinates(x: Int, y: Int) {
   def +(that: CoOrdinates): CoOrdinates = CoOrdinates(this.x +
that.x, this.y + that.y)
  }

  implicit def stringToCoOrdinates(s: String) = {
   val splitted_vals = s.split(",")
   if (splitted_vals.length > 1) {
     CoOrdinates(
       Integer.parseInt(splitted_vals(0)),
       Integer.parseInt(splitted_vals(1)))
   } else {
     CoOrdinates(0, 0)
   }
  }

  val nextCoOrdinate = CoOrdinates(55, 2) + "2,2"

  println(s"${nextCoOrdinate.x}, ${nextCoOrdinate.y}")
```

Exception Handling

```
scala> def half (n : Int) =
     | {
     | if (n % 2 == 0)
     |   n / 2
     | else
     |   throw new RuntimeException("n must be even")
     | }
half: (n: Int)Int

scala> half(6)
res1: Int = 3

scala> half(5)
java.lang.RuntimeException: n must be even
  at .half(<console>:16)
  ... 28 elided
```

```
scala> def half (n : Int) = {
     |  if (n % 2 == 0)      n / 2
     |  else       throw new RuntimeException("n must be even")
     |  }
half: (n: Int)Int

scala>

scala> def get_half_or_default_val(m : Int) = {
     |   val h = try { half(m)  }
     |   catch { case x : Exception => 1  }
     |   println(s"Value of half = $h")
     |   }
get_half_or_default_val: (m: Int)Unit

scala> get_half_or_default_val(6)
Value of half = 3

scala> get_half_or_default_val(5)
Value of half = 1
```

Closure

The function value (the object) that's created at runtime from this function literal

```
scala>   def makeIncreaser(more: Int) = (x: Int) => x + more
makeIncreaser: (more: Int)Int => Int

scala> def inc1=makeIncreaser(1)             //This is a closure
inc1: Int => Int

scala> inc1(20)
res21: Int = 21

scala> def inc7=makeIncreaser(7)                //This is a closure
inc7: Int => Int

scala> inc7(20)
res22: Int = 27

scala>
```

Default function arguments

```
 def printSomething(msg: String = "Something"): Unit =
    println(msg)                              //> printSomething: (msg:
String)Unit

  printSomething()                            //> Something

  printSomething("ABCD")                      //> ABCD

  printSomething(msg = "ABCD")                //> ABCD
```