Operators, Method overloading and Implicit conversions

Niklas Fors

May 28, 2012

Introduction to operators

Operators are ordinary methods.

```
1 + 2
1.+(2)
```

But they have different precedence and associativity.

Different types of operators

Prefix operators: +, -, ! and ~
!false $\Rightarrow$ false.unary_!
Postfix operators: Arbitrary identifier (e op $\Rightarrow$ e.op)
1 toString $\Rightarrow$ 1.toString
Infix operators: Arbitrary identifier (e_1 op $e_2 \Rightarrow e_1$.op(e_2))
2.0 + 3.0 $\Rightarrow$ 2.0.+(3.0)

Infix operators, precedence

Rule: Precedence is determined by the *first* character.

For instance,

`2 + 3 * 5` $\Rightarrow$ `2 + (3 * 5)`

`2 +*** 3 *+++ 5` $\Rightarrow$ `2 +*** (3 *+++ 5)`

`2 max 1 + 2` $\Rightarrow$ `2 max (1 + 2)`

`i += 3 * 5` $\Rightarrow$ `i += (3 * 5)`

`(all other special characters)`

`* / %`

`+ -`

`:`

`= !`

`< >`

`&`

`^`

`|`

`(all letters)`

`(all assignment operators, eg += -= etc.)`

Infix operators, associativity

Rule: Associativity is determined by the *last* character.

`:` is right associative (and is invoked on its right operand!)

all other are left associative

For instance,

`1 + 2 + 3` $\Rightarrow$

`(1 + 2) + 3`

`a ::: b` $\Rightarrow$

`b.:::(a)`

Method overloading (1/2)

```
class C {
    def m(x: Int): Int = x
    def m(x: Int, xs: List[Int]): List[Int] = x :: xs
    def m(y: Int): Double = y.toDouble
}

val c = new C()
c.m(1)
c.m(1, List(2,3))
```

Method overloading (2/2)

```
class A
class B extends A

def f(a: A) { println("A") }
def f(b: B) { println("B") }

var r = new A()
f(r)
r = new B()
f(r)
```

Operators

```
class Rational(val n: Int, val d: Int) {
    def *(r: Rational)
        = new Rational(n * r.n, d * r.d)
    def *(i: Int) = new Rational(n * i, d)
}

val r1 = new Rational(1, 2)
val r2 = new Rational(1, 4)
r1 * r2
r1.*(r2)
r1 * 2
2 * r1
```

Implicit conversions (1/2)

```
val r = new Rational(2)
2 * r

implicit def intToRational(i: Int) = new Rational(i, 1)

2 * r =>
intToRational(2) * r <=>
intToRational(2).*(r)
```

Implicit conversions (2/2)

Other examples:

```
1 to 5 (1 is converted to RichInt)
1 max 2 (1 is converted to RichInt)
-2.5 abs (-2.5 is converted to RichDouble)

Map(
    1 -> "a"  (1 is converted to ArrowAssoc[Int])
)
```

RichInt, RichDouble, ..., are called rich wrappers

Summary

- Operators - ordinary methods
- Method overloading - similar to Java
- Implicit conversions