2018

Scala

INFORMATICA III – MODULO A

LORENZO MILESI – 1030232
Università degli studi di Bergamo

Introduzione

Il piccolo progetto in Scala ha l'obiettivo di utilizzare alcuni dei costrutti tipici del linguaggio al fine di comprenderne meglio le potenzialità. In particolare, si è pensato di costruire un oggetto *vectors* che tratta appunto vettori con componenti di tipo double e consenta di effettuare semplici operazioni su di essi, quali addizioni e moltiplicazioni su ciascuna componente vettoriale.

Costrutti utilizzati

Traits

Il trait è una sorta di interfaccia, che non può essere istanziata ma utilizzata facilmente per estendere classi e oggetti. Esso diviene poi molto utilizzato in contesti pratici qualora si voglia definire un tipo generico che possa avere anche metodi astratti.

```
trait Vector[N] {
    type Self <: Vector[N]
    def *(x: N): Self
    def +(x: N): Self
    def /(x: N): Self
    def -(x: N): Self
    def apply(i: Int): N
    def length: Int

    override def toString(): String = {
      […]
    }
  }
```

Nel caso appena visto è stato definito Vector generico che overrida un metodo toString(), utile per interfacciarsi all'utente. Da notare anche l'utilizzo di type che dichiara un tipo astratto e conforme (un sottotipo insomma) al trait Vector.

Classi final

Semplicemente una classe che non può essere estesa.

```
final class Raised[N](n: N) {
    def *(x: Vector[N]): Vector[N] = x * n
    def +(x: Vector[N]): Vector[N] = x + n
    def /(x: Vector[N]): Vector[N] = x / n
    def -(x: Vector[N]): Vector[N] = x - n
  }
```

Classi case

Scala supporta la notazione case per quanto riguarda le classi. Le classi case sono classi regolari che esportano i propri parametri costruttori.

```
case class DoubleVector(v: Array[Double]) extends Vector[Double] {
    type Self = DoubleVector
    def *(x: Double) = rep(k => k * x)
```

```
    […]
    private def rep(f: Double => Double): Self = {
      […]
    }
  }
```

Logica

La funzione che si occupa di applicare l'operazione matematica alle componenti vettoriali è la seguente:

```
private def rep(f: Double => Double): Self = {
    val nv = new Array[Double](v.length)
    var i = 0
    while (i < nv.length) {
      nv(i) = f(v(i))
      i += 1
    }
    DoubleVector(nv)
}
```

Con un semplice ciclo scorre l'array appena costruito applicando la funzione a tutti gli elementi dell'array passato come parametro e casta infine a DoubleVector l'array ottenuto.

Output

Come esempio si è scelto di costruire un vettore a 6 dimensioni con i numeri interi da 1 a 6 e applicate ad esso un paio di funzioni d'esempio, quali divisione e addizione.

Main:

```
def main(args: Array[String]): Unit = {
    val v = DoubleVector(1, 2, 3, 4, 5, 6)
    println(v)
    println(v / 2)
    println(3.2 + v)
  }
```

Console:

```
DoubleVector(1.0,2.0,3.0,4.0,5.0,6.0)
DoubleVector(0.5,1.0,1.5,2.0,2.5,3.0)
DoubleVector(4.2,5.2,6.2,7.2,8.2,9.2)
```