

# Primo incontro con Scala



Edward Hopper - Summertime

Franco Lombardo

XP User Group - Bergamo

[franco@francolombardo.net](mailto:franco@francolombardo.net)

<http://www.francolombardo.net>



# Premessa

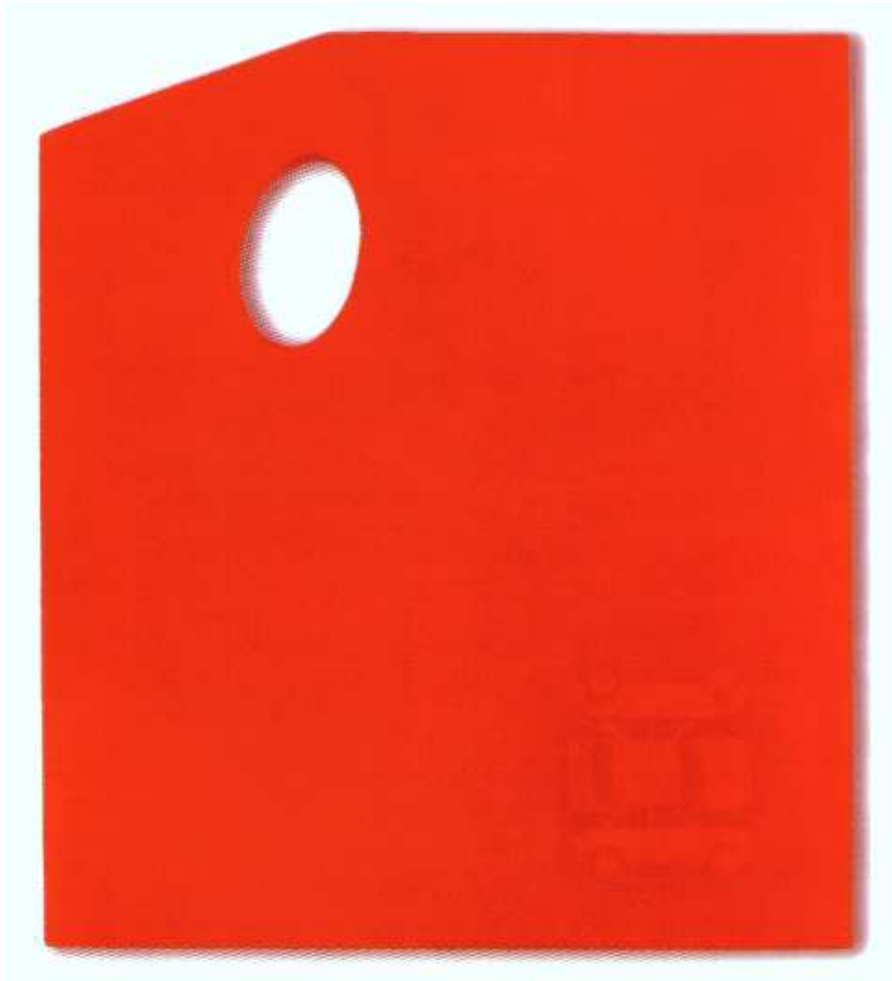


**Non sono un Guru!!!**

<http://www.francolombardo.net>



# Scala: carta di identità



Claudio Destito – Auto-ritratto

- Nasce nel 2001 al Politecnico di Losanna
- Rilasciato pubblicamente per la prima volta nel 2004
- Creato da Martin Odersky, il padre dei “Generics” di Java 5
- Il compilatore genera file *.class* per la JVM (si dice funzioni anche con *.NET* 😊)



# Scala = SCAlable LAnguage

Per “scalare” occorrono basi solide!

La base di Scala è Java



- Virtual machine diffusissima
- Librerie/framework/application server Java sono utilizzabili naturalmente in Scala
- Sintassi di base simile a Java, quindi appetibile per un grande numero di sviluppatori



# Finalmente Hello World!!!

```
package hello

import java.util.Date

object Hello extends Application {
  println("Yet another Hello World program")

  val jvmVer = System.getProperty("java.version")
  println("Running on JMV " + jvmVer)

  println("On " + new Date())
}
```

```
Yet another Hello World program
Running on JMV 1.6.0_13
On Sat May 23 17:36:34 CEST 2009
```

Output



# Finalmente Hello World!!!

```
package hello

import java.util.Date

object Hello extends Application {
    println("Yet another Hello World program")

    val jvmVer = System.getProperty("java.version")
    println("Running on JVM " + jvmVer)

    println("On " + new Date())
}
```

Sintassi base molto simile a Java



# Finalmente Hello World!!!

```
package hello

import java.util.Date

object Hello extends Application {
  println("Yet another Hello World program")

  val jvmVer = System.getProperty("java.version")
  println("Running on JVM " + jvmVer)

  println("On " + new Date())
}
```

Oggetti di java.lang importati di default



# Finalmente Hello World!!!

```
package hello

import java.util.Date

object Hello extends Application {
  println("Yet another Hello World program")

  val jvmVer = System.getProperty("java.version")
  println('Running on JMV " + jvmVer)

  println('On " + new Date())
}
```

Oggetti librerie Java importabili facilmente



# Finalmente Hello World!!!

```
package hello

import java.util.Date

object Hello extends Application {
  println("Yet another Hello World program")

  val jvmVer = System.getProperty("java.version")
  println("Running on JVM " + jvmVer)

  println("On " + new Date())
}
```

Sintassi semplificata: punto e virgola non obbligatorio

<http://www.francolombardo.net>



# Finalmente Hello World!!!

```
package hello

import java.util.Date

object Hello extends Application {
  println("Yet another Hello World program")

  val jvmVer = System.getProperty("java.version")
  println("Running on JVM " + jvmVer)

  println("On " + new Date())
}
```

Sintassi semplificata: meno codice



# Finalmente Hello World!!!

```
package hello

import java.util.Date

object Hello extends Application {
  println("Yet another Hello World program")

  val jvmVer = System.getProperty("java.version")
  println("Running on JVM " + jvmVer)

  println("On " + new Date())
}
```

Sintassi semplificata: inferenza dei tipi



# Inferenza dei tipi: una novità?

```
//Java  
return cliente.getOrdine(40)  
            .getRiga(20)  
            .getArticolo()  
            .getPeso() * 2.5;
```

(OK, questo esempio non rispetta la legge di Demeter, ma è solo un esempio...☺)



# Strumenti per la scalabilità: tipizzazione statica


- Il compilatore fornisce gratuitamente una serie di test automatici sul codice
- Maggiori possibilità di refactoring
- Prestazioni mediamente superiori (OK, non sempre, ma Scala ha sostituito Ruby come motore di Twitter per problemi di prestazioni)
- La tipizzazione statica costituisce una forma chiara ed automatica di documentazione del codice



# Strumenti per la scalabilità: Object Orientation “pura”

- Ogni valore è un oggetto
- Ogni operazione è l’invocazione di un metodo (quindi la ridefinizione degli operatori è banale)

```
2 + 5  
//Equivale a...  
2.+(5)
```



Trucchi sintattici che aiutano la leggibilità:

- Metodi che non incominciano con una lettera
- Possibilità di tralasciare punti e parentesi in alcuni contesti



# Strumenti per la scalabilità: Object Orientation con Traits



- Un problema nei grossi sistemi Object Oriented: oggetti che possono ingrassare a dismisura

Fernando Botero – Bailerina na barra



# Strumenti per la scalabilità: Object Orientation con Traits

- Oggetti che ingrassano: il Cliente

```
//Java
public class Cliente {

    public String ragioneSociale() {
        //implementazione...
    }

    public String indirizzo() {
        //implementazione...
    }
}
```



# Strumenti per la scalabilità: Object Orientation con Traits

- Oggetti che ingrassano: il Cliente per la contabilità

```
//Java
public class Cliente {
    //...
    public Banca bancaDiAppoggio() {
        //implementazione...
    }

    public Solvibilita solvibilita() {
        //implementazione...
    }
}
```



# Strumenti per la scalabilità: Object Orientation con Traits

- Oggetti che ingrassano: il Cliente per il CRM

```
//Java
public class Cliente {
    //...
    public Agente agenteDiZona() {
        //implementazione...
    }

    public Boolean daContattareViaEMail() {
        //implementazione...
    }
}
```



# Strumenti per la scalabilità: Object Orientation con Traits

- Oggetti che ingrassano: il Cliente per gli ordini

```
//Java
public class Cliente {
    //...
    public Vettori[] vettori() {
        //implementazione...
    }

    public Sconti[] sconti() {
        //implementazione...
    }
}
```



# Strumenti per la scalabilità: Object Orientation con Traits

- Oggetti che ingrassano: il Cliente per la produzione

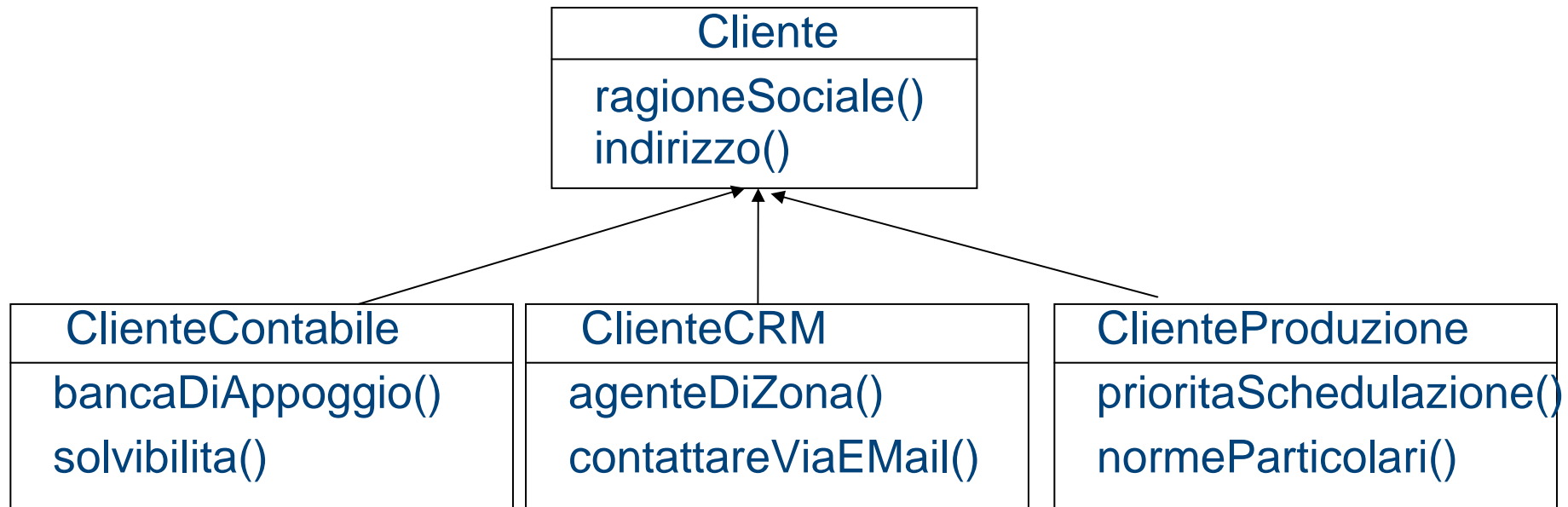
```
//Java
public class Cliente {
    //...
    public Priorita prioritaSchedulazione() {
        //implementazione...
    }

    public NormeTecniche[] normeParticolari() {
        //implementazione...
    }
}
```



# Strumenti per la scalabilità: Object Orientation con Traits

- Oggetti che ingrassano: l'ereditarietà aiuta?



- E se in qualche remota parte del modulo CRM mi occorresse la solvibilità?



# Strumenti per la scalabilità: Object Orientation con Traits

- Oggetti che ingrassano: decomposizione con Traits

```
//Scala
case class Cliente(ragioneSociale: String,
                  indirizzo: String)

trait ClienteContabile {
  def bancaDiAppoggio = //Implementazione
  def solvibilita = //Implementazione
}

trait ClienteCRM {
  def agenteDiZona = //Implementazione
  def contattareViaEMail = //Implementazione
}
```

# Strumenti per la scalabilità: Object Orientation con Traits

- Oggetti che ingrassano: decomposizione con Traits

```
//Scala
val cliente = new Cliente("Prova s.p.a",
                          "Via Roma, 1")
                      with ClienteCRM
                      with ClienteContabile
```

Composizione “dinamica” del tipo a seconda delle necessità



# Strumenti per la scalabilità: programmazione funzionale

## *Higher order functions*

- Le funzioni sono **oggetti** base del linguaggio, come stringhe e numeri
- Le funzioni possono quindi essere parametri e valori di ritorno dei metodi



Il sistema è decomponibile in termini di funzioni generiche riutilizzabili, riducendo le duplicazioni



# Strumenti per la scalabilità: programmazione funzionale

## *Referential transparency*

- Le funzioni calcolano i risultati esclusivamente in base ai valori di ingresso, senza fare riferimento ad uno stato
- La valutazione di una funzione non dovrebbe avere alcun effetto collaterale diverso dal calcolo del risultato



E' più facile riutilizzare le funzioni se non è necessario riprodurre uno stato

E' più facile testare il codice



# Strumenti per la scalabilità: programmazione funzionale

Un esempio: l'integrazione numerica...

# Strumenti per la scalabilità: programmazione funzionale

...scherzavo, facciamo un esempio più “concreto”

```
//Ecco una riga d'ordine
case class OrderRow(description: String,
                    price: Double,
                    qty: Double) {
  def amount = price * qty
}

//Ed un ordine
val order = List(OrderRow("Beer", 5.0, 2),
                 OrderRow("Chips", 2.5, 1))
```



# Strumenti per la scalabilità: programmazione funzionale

```
//Un approccio tradizionale per il totale IVA
var total = 0.0
for (row <- order) {
  total += row.amount *0.2
}
println("VAT (Total): " + total);
```



# Strumenti per la scalabilità: programmazione funzionale

```
//Un approccio tradizionale per il totale IVA
var total = 0.0
for (row <- order) {
  total += row.amount * 0.2
}
println("VAT (Total): " + total);
```

In realtà uniamo 3 operazioni

- **Ciclo**



# Strumenti per la scalabilità: programmazione funzionale

```
//Un approccio tradizionale per il totale IVA
var total = 0.0
for (row <- order) {
  total += row.amount * 0.2
}
println("VAT (Total): " + total);
```

In realtà uniamo 3 operazioni

- Ciclo
- Accumulazione



# Strumenti per la scalabilità: programmazione funzionale

```
//Un approccio tradizionale per il totale IVA
var total = 0.0
for (row <- order) {
  total += row.amount * 0.2
}
println("VAT (Total): " + total);
```

In realtà uniamo 3 operazioni

- Ciclo
- Accumulazione
- **Calcolo**



# Strumenti per la scalabilità: programmazione funzionale

Scomponiamo le 3 operazioni

```
//Il calcolo (è una funzione che assegnamo)
val vat = (row: OrderRow) => row.amount * 0.2

//Composizione di aggregazione e calcolo
def composition(aggr: (Double, Double) => Double,
               calculus: (OrderRow => Double))
               (partial: Double, row: OrderRow) =
  aggr(partial, calculus(row))

val totalVat =
  order.foldLeft(0.0)(composition(_+_, vat))
```



# Strumenti per la scalabilità: programmazione funzionale

Scomponiamo le 3 operazioni

```
val totalVat =  
    order.foldLeft(0.0)(composition(_+_, vat))
```

- **Ciclo**

```
//Java  
B b = start;  
for(final A a : listOfA) {  
    b = method(b, a);  
}  
return b;  
  
//Scala  
listOfA.foldLeft(start)(method)
```



# Strumenti per la scalabilità: programmazione funzionale

Scomponiamo le 3 operazioni

```
val totalVat =  
    order.foldLeft(0.0)(composition(_+_ , vat))
```

- Ciclo
- Accumulazione



# Strumenti per la scalabilità: programmazione funzionale

Scomponiamo le 3 operazioni

```
val totalVat =  
    order.foldLeft(0.0)(composition(_+_, vat))
```

- Ciclo
- Accumulazione
- **Calcolo**

# Strumenti per la scalabilità: programmazione funzionale

Possiamo ricomporre in modo differente

```
val totalAmount =  
  order.foldLeft(0.0)(composition(_+_,  
                                   _.amount))  
  
val maxAmount =  
  order.foldLeft(0.0)(composition(Math.max,  
                                   _.amount))  
  
val maxVat =  
  order.foldLeft(0.0)(composition(Math.max,  
                                   vat))
```



# Tipi strutturali

Come testare questo metodo?

```
def deleteAllRows(statement: java.sql.Statement) =  
    statement.execute("DELETE FROM MYTABLE")
```

Un mock fatto a mano?

L'interfaccia Statement ha 41 metodi!!!



# Tipi strutturali

Modifichiamo il nostro metodo dichiarando solo quello che ci occorre

```
def deleteAllRows(  
  statement: {def execute(sql: String): Boolean}) =  
  statement.execute("DELETE FROM MYTABLE")
```



# Tipi strutturali

Ora possiamo testare facilmente

```
def testDeleteAllRows() {  
    val mockStatement = new {  
        def execute(sql: String) = {  
            println(sql) //Oppure qualsiasi cosa x test  
            true         //Valore di ritorno di execute  
        }  
    }  
  
    deleteAllRows(mockStatement)  
}
```



# Conversioni implicite

Se trovassimo una libreria per eseguire query remote?

```
class RemoteStatement {  
  def remoteExecute(sql: String) = {  
    println(sql + " executed remotely :-)")  
    true  
  }  
}
```



# Conversioni implicite

Nessun problema: convertiamo al volo!

```
implicit def normalize(remote: RemoteStatement) =  
  new {  
    def execute(sql: String) =  
      remote.remoteExecute(sql)  
  }  
  
//Posso invocare anche se il tipo non è corretto!  
deleteAllRows(new RemoteStatement)
```

# Domain Specific Languages

In un programma mi piacerebbe scrivere

```
val tenDollars = (4.0 USD) + (6.0 USD)
```



Andy Warhol– Dollar sign

# Domain Specific Languages

Nessun problema!

```
abstract class Currency {  
    def name: String;  
    override def toString = name  
}  
  
object Euro extends Currency {  
    def name = "EUR"  
}  
  
object Dollar extends Currency {  
    def name = "USD"  
}
```



# Domain Specific Languages

Nessun problema!

```
case class Money[C <: Currency](amount: Double,
                                  currency: C) {
  def + (otherMoney: Money[C]) =
    new Money(amount + otherMoney.amount,
              currency)

  override def toString = amount + " " + currency
}
```



# Domain Specific Languages

Nessun problema!

```
object Money {  
  implicit def doubleConverter(d: Double) = new {  
  
    def EUR = {  
      new Money(d, Euro)  
    }  
  
    def USD = {  
      new Money(d, Dollar)  
    }  
  }  
}
```



# Domain Specific Languages

Con questi strumenti possiamo scrivere DSL interni molto interessanti!

```
object Playground extends BasicClass {  
  def main(args: Array[String]) {  
    10 PRINT "SCALA ROCKS!"  
    20 GOTO 10  
  
    RUN  
  }  
}
```

Esempio di Szymon Jachim

Vedi <http://www.scala-lang.org/node/1403>

