

# SCALA Notes



Ha connet.

Scala è nato con lo scopo di ampliare Java, migliorandolo.

Scala è leggero ne ha una sintassi tipizzata

- permette, a seconda delle esigenze di usare programmazione OOP o funzionale.
- un linguaggio scalabile, perfetto per aggredire DSLs
- stabile e innovativo
- compilabile con SAWA

REPL è una shell interattiva per Scala, capace di compilare ed eseguire codice. Utile per test.

VARIABILI → Immutabili, mutabile. (C'è Type Inference)

Uno Statement viene eseguito senza return, ma i suoi side effects, un'espressione ritorna un valore.

In Scala non sono necessarie parentesi greche per espressioni a riga singola, le annotazioni dei tipi possono essere omesse, non servono tratt & la parentesi i. Non c'è bisogno di RETURN.

## OO Basics

### Object-orientation in Scala

- Classes and traits
  - Fields keep state
  - Methods provide operations (encapsulation)
  - Access modifiers declare visibility (information hiding)
- Singleton objects are first-class objects
- Inheritance
  - Single inheritance, i.e. extend exactly one superclass
  - Multiple traits can be mixed in

In Scala il costruttore primario è di default, se ne vuole definire uno si usa `def this`.

I parametri di una classe non sono fields, possono essere usati nel body della classe ma non possono essere scritti.

Essi possono diventare fields antecedendo val o var al nome. I metodi provvedono operazioni in una classe: `def nome.metodo(parametri): tipo ritorno`

I metodi con un parametro possono usare la notazione infix. Usare infix per metodi con nomi indescriptibili. Usare prefix per un arg.

Private → solo classe / Protected → nell'alfabeto a subclassi

Gli OBJECT vengono usati per oggetti statici e singleton.

Un companion object è una classe o un'interfaccia che ha lo stesso nome di un oggetto. Passano accadono ai membri privati dell'object.

**CASE CLASS** = - Non c'è bisogno di NEW per inizializzazione

- crea un companion Object (usare .apply)
- crea toString, equals e hashCode, copy
- tutti i campi sono immutabili

Le classi FINAL sono estensibili, SEALED → solo nello stesso file

## - SCALA ADVANCED

- Una buone pratica è rendere le funzioni ricorsive TAIL → @tailrec una funzione, Bene nona user il tail dentro una funzione esposta come local method.

Una **funzione parziale** è una funzione che restituisce una risposta solo per un'osservazione. Sono:

- uno o più operatori (prendono un solo parametro)
- si applicano ad un sottoinsieme di valori
- possono avere una funzione **isDefinedAt** per capire se applicabile

Una **PF Literal** è letta da un blocco di cases, bisogna stare attenti ai match non esaurienti.

**SLIDING** → raggruppare gli elementi di una collezione tramite una window.

**FOLDLEFT<sup>right</sup>** → riducono una collezione ad un valore singolo. Una funzione binaria è applicata dal **right** agli altri elem.

**CUSTOM VALUE CLASS** → Si deve estendere AnyVal. Bisogna:

- abbracci solo un parametro (dove essere val)

Le classi, interfacce e metodi possono essere **parametrically polymorphic**. Uno o più parametri possono essere posti tra parentesi quadre.

**VARIANCE** → Avviene per via della sostituzione. Nessuna relazione → **INVARIANT** → Real-Write → **invariante**  
Allineati → **covariant** → Real-only → **con +**  
Opposti → **contravariant** → Write-only → **con -**

**VALS** sono covariant, **VAR** sono contravariant. Parametri delle funzioni **COVARIANT** mentre i return **INVARIANT**  
essi possono essere gestiti dai **LOWER BOUND** con **>:** e **UPPER BOUND** **<:**

Un tipo esistente può essere **difinito** usando dichiarazioni o definizioni, una dichiarazione da aggiungere  
un nuovo membro è **lettore STRUCTURAL**. Le performance ne possono risentire.

## Static duck typing

- The duck test: If it looks like a duck, swims like a duck, and quacks like a duck, then it probably is a duck
- Omitting the type to be refined you get a purely **structural type**:

```
type Closeable = { def close(): Unit }

scala> val c1: Closeable = new java.io.StringWriter
c1: Closeable =

scala> val c2: Closeable = new Door
c2: Closeable = Door@4cd43058
```

## phantom types

```

object Power {
    sealed trait On extends Power
    sealed trait Off extends Power
}

class LightSwitch[State <: Power] private {
    def on[S >: State <: Power.On] = new LightSwitch[Power.On]
    def off[S >: State <: Power.Off] = new LightSwitch[Power.Off]
}

object LightSwitch {
    def on = new LightSwitch[Power.On]
    def off = new LightSwitch[Power.Off]
}

```

represent il  
flusso

Si usano anche :=

## Implicit conversion

```
implicit def stringToInt(value: String): Int = Integer.parseInt value
```

Per essere applicati deve essere in scope (accessibile senza prefix)

Sono costituiti da due membri legati agli associati. Devono essere solo uno

### Type classes

- A type class is a polymorphic trait declaring an interface:

```
trait Bool[A] {
    def bool(a: A): Boolean
}
```

- A type class instance is an implicit value defining the interface for some type:

```
implicit val intBool = new Bool[Int] {
    override def bool(n: Int): Boolean =
        n != 0
}
```

### Type class glue

- A type class could be used standalone:

```
scala> intBool bool 0
res0: Boolean = false
```

- Using an implicit parameter for the type class instance, supported types can be extended:

```
implicit class BoolOps[A](val a: A) extends AnyVal {
    def bool(implicit instance: Bool[A]): Boolean =
        instance bool a
}

scala> 0.bool
res0: Boolean = false
```

### Using ClassTags

- To access an implicit ClassTag, use the `classTag` method
- The wrapped runtime class can be accessed via the `runtimeClass` method
- A `ClassTag` is also an extractor, i.e. defines an `unapply` method which is used in pattern matching:

```
import scala.reflect._

def conforms[A : ClassTag](any: Any): Boolean =
    (classTag[A] unapply any).isDefined

scala> conforms[java.util.Date](new java.sql.Date(0))
res0: Boolean = true

scala> conforms[java.util.Date]("")
res1: Boolean = false
```

Un DSL è un modo particolare di utilizzare un linguaggio esistente

per lavorare alla lingua ospite la sensazione di una lingua particolare.

Tecnicamente non c'è differenza tra DSL e una libreria, tuttavia un DSL ha le seguenti proprietà:

- Comprensibile intuitivamente per gli esperti di dominio.
- Di alto livello e facile da usare
- Robusto