**SCALA TRAITS:**

Motivazione

In Java, a class can implement an arbitrary number of *interfaces*. This model is very

useful for declaring that a class exposes multiple abstractions. Unfortunately, it has one

major drawback.

For many interfaces, much of the functionality can be implemented with boilerplate

code that will be valid for all classes that use the interface. Java provides no built-in

mechanism for defining and using such reusable code.

Instead, Java programmers must use ad hoc conventions to reuse implementation code for a given interface. In the worst case, the developer just copies and pastes the same code into every class that needs it.

Often, the implementation of an interface has members that are unrelated (“orthogonal”)

to the rest of the instance’s members. The term ***mixin***is often used for such focused

and potentially reusable parts of an instance that could be independently maintained.

Rules:

A trait is like an interface with a partial implementation.   
In scala, trait is a collection of abstract and non-abstract methods. You can create trait that can have all abstract methods or some abstract and some non-abstract methods.

A variable that is declared either by using val or var keyword in a trait get internally implemented in the class that implements the trait. Any variable which is declared by using val or var but not initialized is considered abstract.

Traits are compiled into Java interfaces with corresponding implementation classes that hold any methods implemented in the traits.

traits can inherit classes.

Basi sintassi su demo.  
Tutorial **UC**: da verificare su pdf Scala tutorial per java developers