Hosted by 7PEAKS

SCALA WORKSHOP

CHANGING YOUR CODING WAY

MAR 20, 2019 18.30 - 21.00

Giorgio Desideri
Cloud Software Architect

7 Peaks Software Dhammalert Building , 2nd Floor

HOUR 1

- Scala Language Concepts
- State(less/full) in Scala
- DSL and REPL
- Function vs Method
- Call (By-Value/By-Name)
- High-Order Function

SCALA

PEAKS

The design of Scala started in 2001 at the École Polytechnique Fédérale de Lausanne (EPFL) (in Lausanne, Switzerland) by Martin Odersky.

https://en.wikipedia.org/wiki/Scala_(programming_language)#History

Why “SCALA” ?

PEAKS

- 1st Reason: "scala" is the Italian word for STAIRWAY, appropriated since Scala helps you to ascend to a better programming language. The Scala logo is an abstraction of a stairway.

https://www.scala-lang.org/old/node/250.html

Why “SCALA” ?

PEAKS

- 2nd Reason: Scala stands for scalable language, because Scala's concepts scale well to large programs.

https://www.scala-lang.org/old/node/250.html

Scala Language Concepts

PEAKS

Scala is defined as:

- Object-oriented language,
- Functional language,
- Blended language,
- Scalable and Extensive

Object Oriented

- Encapsulation/information hiding
- Inheritance
- Polymorphism/dynamic binding
- All predefined types are objects
- All user-defined types are objects
- All operations are performed by sending messages to objects

Object Oriented

- Modular mixin composition, Scala has TRAITS ~ Java Interfaces + Abstract classes.

Object Oriented

PEAKS

DEFINITION. "A mixin is a class that provides certain functionality to be inherited by a subclass and isn't meant for instantiation by itself. A mixin could also be viewed as an interface with implemented methods."

Object Oriented

PEAKS

Breath ...

Example is coming soon ...

Just only one concept more

Object Oriented

PEAKS

- Self-type, Self-types are a way to declare that a trait must be mixed into another trait, even though it doesn't directly extend it, making the members of the dependency available without imports.

Object Oriented

PEAKS

Example

Object Oriented

- Type abstraction, there are two principles of abstraction in programming languages: parameterization and abstract members.

Scala supports both forms of abstraction uniformly for types and values.

Functional Language

PEAKS

"Functional programming is a programming paradigm that treats computation as the evaluation of mathematical functions and avoids state and mutable data."

(Scala In Action, Nilanjan Raychaudhuri, Manning)

Functional Language

PEAKS

An object is called MUTABLE when you can alter the contents of the object if you have a reference to it.

Functional Language

PEAKS

An object is called IMMUTABLE when this object can't be altered if you have a reference to it.

Benefits of immutables

PEAKS

- No need lock/synchronization techniques for concurrent access
- No need to keep track of the changes
- All objects are accessible by others despite of the visibility modifiers (private, sealed)

Others

- Blended : Object Oriented + Functional language
- Scalable and Extensible, for the nature of Scala language.

PEAKS

Question(s) ?

State(less/full) in Scala

PEAKS

In Scala object can be Mutable or Immutable, but it IS NOT the definition of the state.

STATE := "*the particular condition that someone or something is in at a specific time*"

Stateless

PEAKS

It is an object that doesn't change its state across the time and its usage.

It means that from its initialization until destruction the object never changes its status.

Stateful

PEAKS

It is an object that change its state across the time and its usage.

It belongs to the switch the implementation from immutable to mutable, that, in Scala, makes its prone to errors or concurrency issues.

DSL and REPL

PEAKS

- DSL => Domain Specific Language,
- REPL => Read Evaluate Print Loop, interactive shell as Python or Java (only from Java9 to upper version)

DSL examples

PEAKS

- HTML,
- Unix shell scripts,
- MATLAB and Logo,
- SQL,
- YACC grammars for creating parsers

DSL with Scala

PEAKS

- All types are objects
- All functions are objects, with own methods
- Infix syntax

```
x.method1(y) => x method1 y

Giorgio loves { pizza && pasta }
Giorgio.loves(pizza && pasta)
```

Consideration

PEAKS

- Define / Understand your domain
- Manage the changes (not-regression tests)
- mathematical functions, than no cut&paste.
- F(function + state + im/mutable types)
  => DSL

State(full/less)

PEAKS

Question(s) ?

Call By-Value

PEAKS

The variable is initialized when the method is called and it will be constant in whole scope.

(case of stateless and immutability)

Call By-Value

PEAKS

```
def exec(t: Long): Long = {
        println("Entered exec, calling t …")
        println("t = " + t)
        println("Calling t again …")
        t
}
```

Call By-Name

The variable t is initialized when it is called, so its values will be different every time.

(case of stateful, both mutable or immutable).

Call By-Name

PEAKS

```
def exec(t: => Long): Long = {
      println("Entered exec, calling t ...")
      println("t = " + t)
      println("Calling t again ...")
      t
}
```

Call By-Name

PEAKS

The Call-by-Name technique is useful to instantiate a variable (immutable or not) reducing the time and space complexity.

Call by Name/Value

PEAKS

Example

Method vs Function

PEAKS

A Scala method, is a part of a class. It has a name, a signature, optionally some annotations, and some bytecode.

Method

PEAKS

```
def myFunction(param1 : Int, ….) : Long = {
    // code
    …
}
```

Method vs Function

PEAKS

A function in Scala is a complete object. Scala has a series of traits represent functions (`Function0`, `Function1`, etc.).

As an instance of a class that implements one of these traits, a function object has methods

Local Function

PEAKS

```
def multiply(a : Int, b : Int) : Int = {
  // local function
  def sum(value : Int, times : Int) : Int = {
    // for 1 -> times ( value + value )
  }

  // this is the body of 'multiply'
  sum(times = b, value = a)
}
```

1st class Function

PEAKS

The 1st-class function means that you can express functions in function literal syntax, and that functions can be represented by objects, which are called “function values”.

1st class Function

PEAKS

```
// use val or var to store a function
scala> var increase = (x: Int) => x + 1
increase: (Int) => Int = <function1>

// use the function
scala> increase(10)
res0: Int = 11
```

1st class Function

PEAKS

```
// define a list of integer
scala> val list : List[Int] = List(1, 2, 3,4, 5,6,7,8,9)
list: List[Int] = List(1, 2, 3, 4, 5, 6, 7, 8, 9)

// filter the list
scala> list.filter(x => x % 2 == 0 )
res0: List[Int] = List(2, 4, 6, 8)
```

Partial(ly) Function

PEAKS

A partially applied function is an expression in which no need to supply all of the arguments needed by the function. Instead, you supply some, or none, of the needed arguments

To make a function literal even more concise, can be used the underscores as placeholders.

Partial(ly) Function

PEAKS

```
// As before
scala> list.filter( x => x % 2 == 0 )
res0: List[Int] = List(2, 4, 6, 8)

// As now : partial function
scala> list.filter( _ % 2 == 0 )
res0: List[Int] = List(2, 4, 6, 8)
```

Closures

PEAKS

Closure is a function object that captures "free" variables, and is said to be "closed" over the variables visible at the time it is created.

Closures

```
// this is the closure
def exec( f : (String) => Unit, name: String ) {
  f(name)
}
// generic function
def sayHello(name: String) {
  println(s"Hello, ${name}")
}
// call the closure with my name
exec(sayHello, “Giorgio”)
```

Closures

Difference between lambda and closure?

Lambda is an anonymous function, Closure can have a name and it closes over the environment.

PEAKS

Question(s) ?

High Order Function

PEAKS

A function is called “higher order”

- if it takes a function as an argument,
- or returns a function as a result

map()

PEAKS

It allows you to build a new List by applying a function to all elements of a given List.

```
def map[B](f: (A) ⇒ B): Traversable[B]
```

flatMap()

Equivalent of map, where apply the flatten() to the resulting list.

```
def flatMap[B](
    f: (A) ⇒ GenTraversableOnce[B]):
        TraversableOnce[B]
```

foldLeft() - foldRight()

PEAKS

The foldLeft takes an associative binary operator function as parameter and will use it to collapse elements from the collection.

```
def foldLeft[B](z: B)(op: (B, A) ⇒ B): B
def foldRight[B](z: B)(op: (B, A) ⇒ B): B
```

For comprehension

PEAKS

Scala “for comprehensions” is nothing more than a syntactic sugar for composition of multiple monadic operations (foreach, map, flatMap, filter and filterWith).

PEAKS

Example

PEAKS

End of HOUR 1

HOUR 2

PEAKS

- Monoids
- Monads
- Hierarchy and Polymorphism
- Options and Futures
- Code Format / UAP principle

Monoids

PEAKS

In abstract algebra, a monoid is an algebraic structure with a single associative binary operation and an identity element.

Monoids

PEAKS

- Some type A
- A binary associative operation that takes two values of type A and combines them into one
- A value of type A that is an identity for that operation.

PEAKS

Example

Monads

PEAKS

Monads are structures that represent sequential computations. A Monad is not a class or a trait; it is a concept.

A Monad is an object that wraps another object in Scala

Monads

PEAKS

- flatMap, where it adds the flatten operation
- For-comprehension, where there are a single block (structure) of operation visible defined. “Visible” means that you can see and not hide into a function (as flatMap).

PEAKS

Example

Hierarchy & Polymorphism

- Hierarchy tree
  - See Image of tree
  - Focus on `Nothing` and `Null`

PEAKS

Example

Options

- `Option[A]` ,is the abstract class to represents the implementations
- `Some` ,options with value.

  E.g. `Option[Int] = Some(10)`
- `None` ,options without value, aka `null`

Benefits of Options

PEAKS

- Options are better than null. No `NullPointerException`, or if/else
- Options are objects with methods
- Options can provide information about exception (`Either`, `Left`, `Right` as `Option` / `Some` / `None`)

Futures

PEAKS

- Futures provides a way to perform operations in parallel in an efficient and non-blocking way.
- A Future is a placeholder object for a value that may not yet exist.

Callbacks

PEAKS

- onComplete()

```
val f : Future[List[Int]] = someFunct()

f onComplete {
   case Success(result) => { … }
   case Failure(e) => { … }
}
```

Callbacks

PEAKS

- onSuccess()

```
val f : Future[List[Int]] = someFunct()

f onSuccess {
   case result => { … }
}
```

Callbacks

PEAKS

- onFailure()

```
val f : Future[List[Int]] = someFunct()

f onFailure {
   case e => { … }
}
```

Callbacks

PEAKS

- Await()

```
val f : Future[List[Int]] = someFunct()

val res : List[Int] =
    Await.result(f, Duration.Inf)
```

Code Format / UAT

PEAKS

The uniform access principle of computer programming is *"All services offered by a module should be available through a uniform notation, which does not betray whether they are implemented through storage or through computation"*.

Bertrand Meyer in Object-Oriented Software Construction

Code Format / UAT

PEAKS

Empty parentesis methods:

- there are no parameters,
- the method that accesses to state of the referenced object (with mutable or immutable types).

Getter

```
class Person {
   val name : String = “My Name”
}

val c = new Person
c.name

// this is the method name() to access
// to “name” variable
```

Setter

PEAKS

```
class Person {
   var name : String = “My Name”
}

val c = new Person
c.name = “My Other Name”

// def name_=(value: String) = name = value
```

Method

```
class Person {
   def name : String = “Giorgio”
}

val c = new Person
c.name

// method “name” is accessed in the same
way of getter and setter.
```

PEAKS

End of HOUR 2

PEAKS

Question(s) ?

Meet our speaker

GIORGIO DESIDERI

Cloud Software Architect

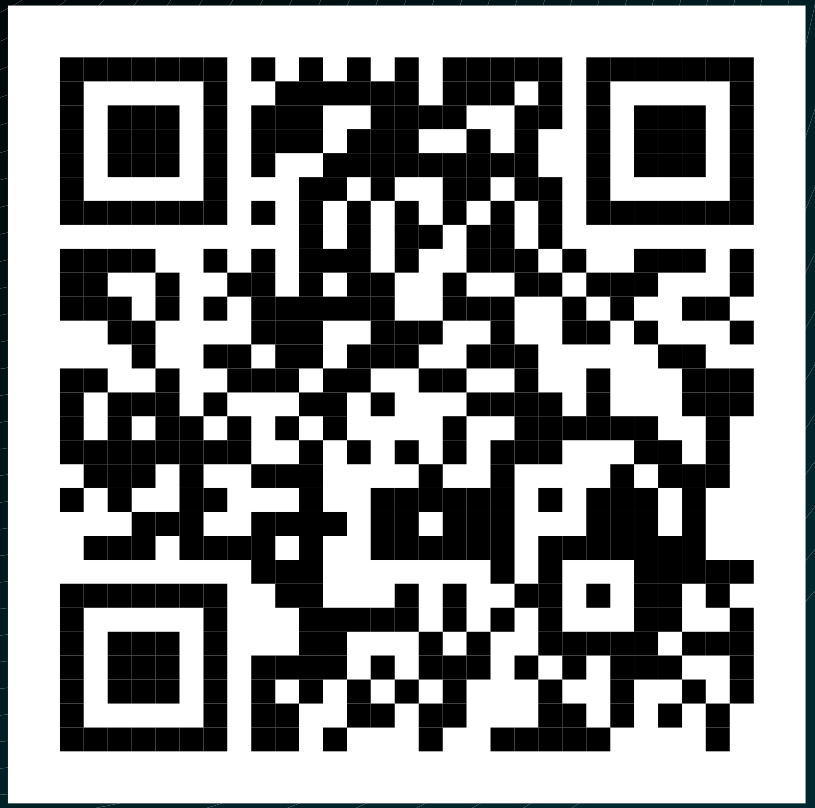
Giorgio has been working in the IT industry for over 13 years. Working in positions ranging from System Engineer to Project Manager. Initially moving to Thailand to join Agoda, he has since taken a bigger role as a Cloud Software Architect at 7 Peaks Software.

PEAKS

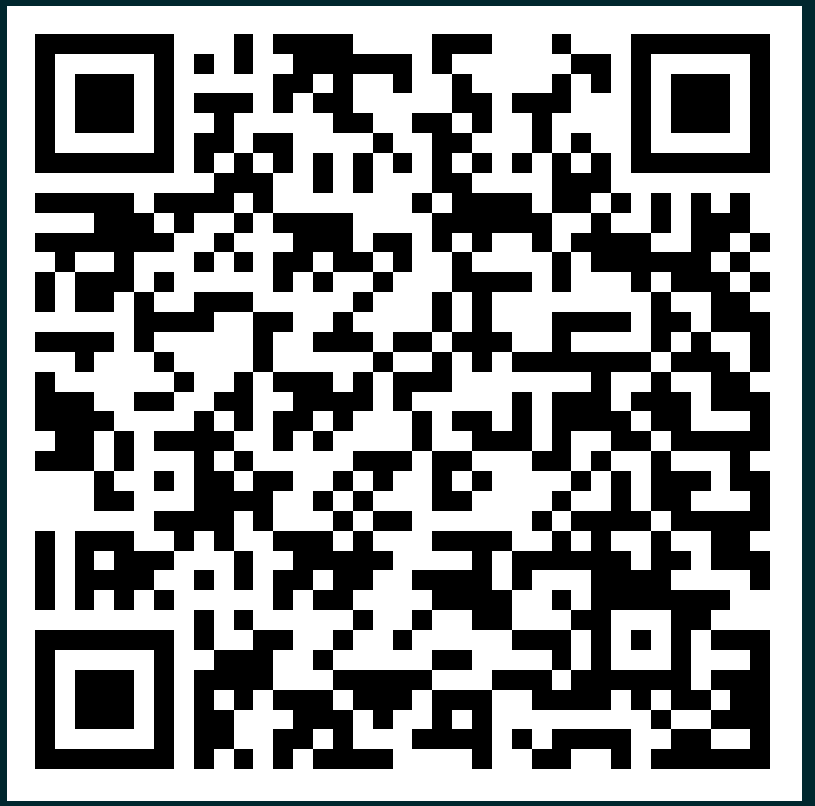
Who we are

PEAKS

Links



Materials



Event Feedbacks