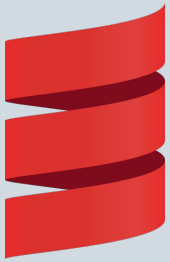


SCALA

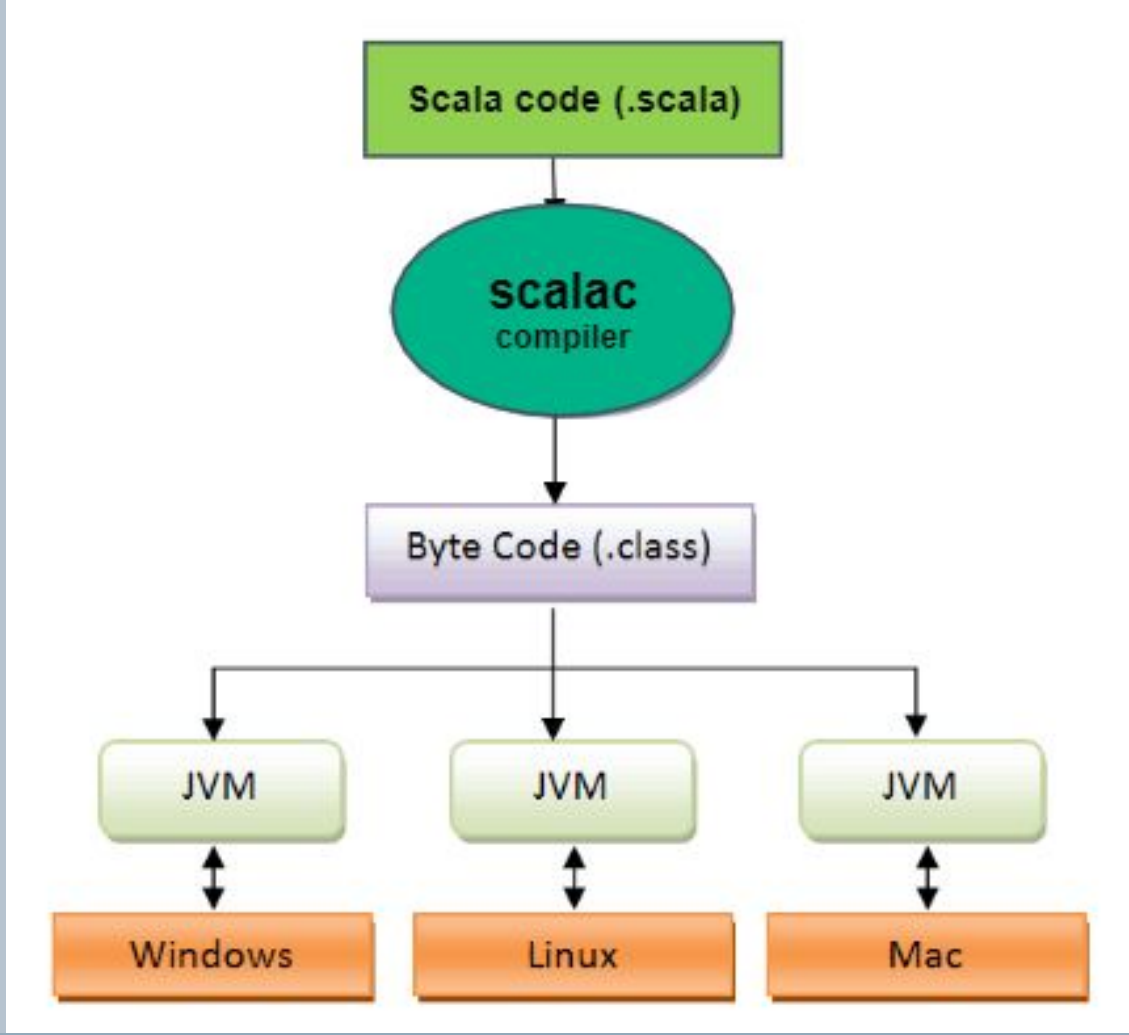
PROGRAMING LANGUAGE

História

- Scala foi criada por Martin Odersky
- Teve sua primeira versão lançada em 2003

Instalação Linux

- Java 1.8 ou superior
- sudo apt-get install scala



JVM

https://i.stack.imgur.com/XW1Gv.png

Hello, World

```
object Main{
    def main(args: Array[String]):Unit = {
        println(“Hello, World”);
    }
}
```

```
>> scalac main.scala
>> scala Main
>> Hello, World
```

Características

Características

- Multi-paradigma (Funcional e Orientada a Objetos)
- Híbrida
- Estaticamente tipada
- Inferência de tipo
- Tudo é objeto

Multi paradigma

```
class Disciplina(private var _codigo: String, private var _nome: String) {


…
}
```

Multi-paradigma

//scala.collection.**immutable**.Map X scala.collection.**mutable**.Map

var _disciplinas = Map[String, Disciplina]()
//Codigo, Objeto Disciplina

// Adicionar elementos:
_disciplinas += ("INF09307" -> LP)

// Remover elementos:
_disciplinas -= "INF09307"

// Atualizar elementos:
states("INF09307") = SO

//Retorna um Option (scala.Option): Some(x) ou None

_disciplinas.values.find(x => x.codigo == “INF09307”)

Multi-paradigma

```
def addAvaliacao(ava: Avaliacao, codDiscip: String) = {
        _disciplinas.get(codDiscip) match {
                case Some(d) =>
                        d.addAvaliacao(ava)
                case None =>
                        throw new IllegalArgumentException("Código de disciplina não definido usado na avalição " + ava.codigo + ": " + codDiscip + ".")
                }
    }
```

Amarrações

Case Sensitive:

```
var nome = 10;
var Nome = 20;
println(nome);
println(Nome);
```

```
>> 10
>> 20
```

Espaço de nomes:

```
//Invalid
123 //Não começa com uma letra
someString? //Não usa underscore depois da String
someString_a? //Não usa underscore depois da String
?a //Começa com operador e depois usa caracter
a_?_b //To match #1 any operator char must come last
```

```
//Valid
someString
SomeString
a123
someString_?
a_b_?
?
?+-<>:|!&%#\^@~*_
```

Palavras Reservadas

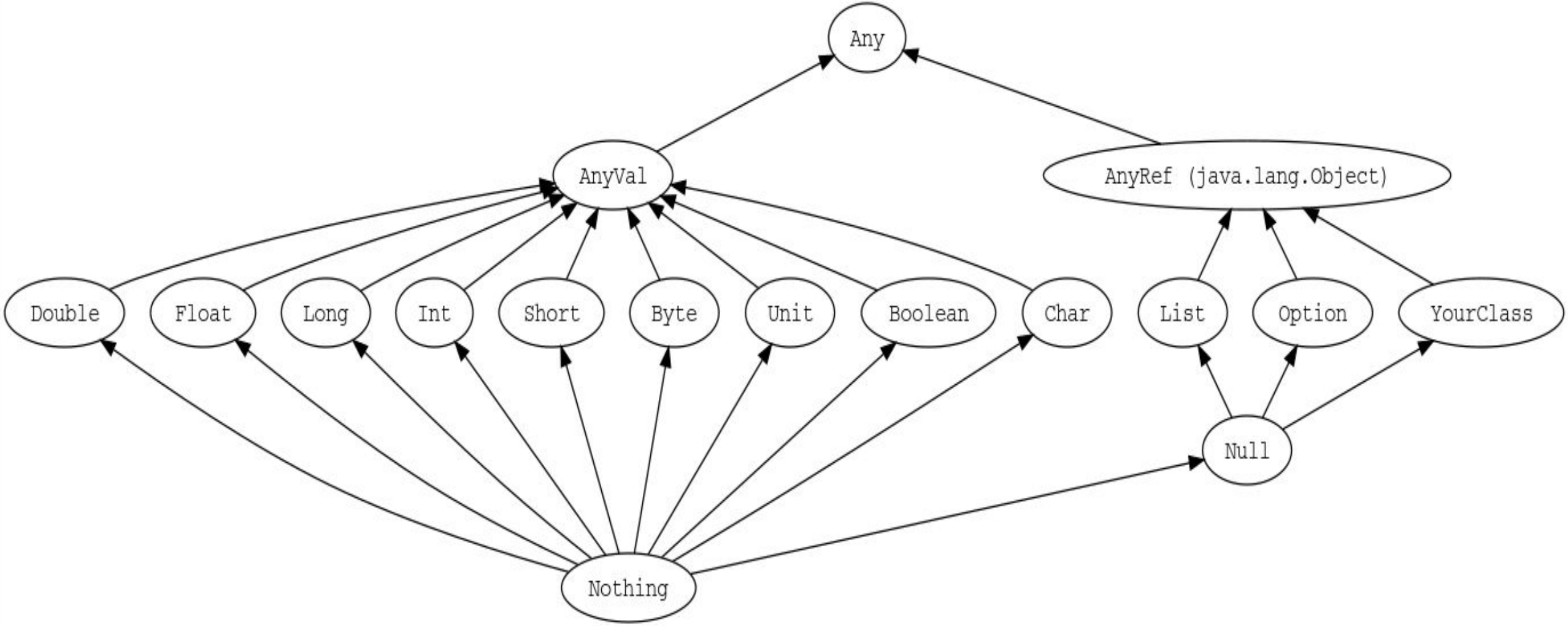
abstract	**case**	**catch**	**class**	**with**	**this**
def	**do**	**else**	**extends**	**yield**	**throw**
false	**final**	**forSome**	**if**	**while**	**trait**
finally	**for**	**implicit**	**import**	**var**	**try**
lazy	**match**	**new**	**null**	**val**	**super**
object	**override**	**package**	**private**	**type**	**sealed**
protected	**return**				

Escopo Estático:

```
var x = 1;
def sub1() = {
  println(x);
}
def sub2() = {
    var x = 3;
    sub1();
}
sub2();
sub1();
```

```
1
1
```

Valores e Tipos de Dados



Hierarquia de Tipos

http://docs.scala-lang.org/tour/unified-types.html

Tipagem Estática

```
var x = 1;
println(x);

x = "LP";
println(x);
```

```
>> type mismatch; found   : String("LP")
required: Int
```

Variáveis e Constantes

Inferência de Tipos

```
var x = 1;
var x1:Double = 1;
var y = "Programação"
var z = List(1,2,3,4,5);

println("Valor: " + x + " Tipo: " + x.getClass());
println("Valor: " + x1 + " Tipo: " + x1.getClass());
println("Valor: " + y + " Tipo: " + y.getClass());
println("Valor: " + z + " Tipo: " + z.getClass());
```

```
>> Valor: 1 Tipo: int
>> Valor: 1.0 Tipo: double
>> Valor: Programação Tipo: String
>> Valor: List(1, 2, 3, 4, 5)
Tipo:collection.immutable.
```

Constantes

```
val x:String = "Constante";
x = "LP";

println(x);
```

>> reassignment to val

Escrita em memória secundária

```
import java.io._

val file = new File("teste.txt" );
val writer = new PrintWriter(file);

writer.write("Hello Scala");
writer.close();
```

teste.txt

>> Hello Scala

Leitura em memória secundária

```
import scala.io.Source

val filename = "teste.txt"
for (line <- Source.fromFile(filename).getLines) {
    println(line)
}
```

```
>> Hello Scala
```

Escrita em memória secundária

```
import java.io._
@SerialVersionUID(123L)
class Pessoa(var nome: String, var idade: Int) extends Serializable

val pessoa = new Pessoa("Gustavo", 20)

val obj = new ObjectOutputStream(newFileOutputStream("pessoa"))
obj.writeObject(pessoa)
obj.close
```

Leitura em memória secundária

```
import java.io._
@SerialVersionUID(123L)
class Pessoa(var nome: String, var idade: Int) extends
Serializable

val obj = new
ObjectInputStream(newFileInputStream("pessoa"))
val pessoa = obj.readObject.asInstanceOf[Pessoa]
obj.close

println(pessoa.nome)
println(pessoa.idade)
```

```
>> Gustavo
>> 20
```

Expressões e Comandos

Operadores Aritméticos

- +
- -
- *
- /
- %

Operadores Relacionais

- ==
- !=
- >
- <
- >=
- <=

Operadores Lógicos

- &&
- ||
- !

Operadores bit a bit

- &
- |
- ^
- ~
- <<
- >>

Chamadas de função

```
def add(x: Int, y: Int): Int = x + y

def mult(x:Int, y:Int) = x*y

def add1(x: Int, y: Int):Int = {
  return x + y;
}

println(add1(1,2));
println(add(1,2));
println(mult(1,2));
```

```
>> 3

>> 3

>> 2
```

Funções como cidadãs de primeira classe

```
val add = (x:Int, y:Int) => x + y

println(add(1,4));
```

```
>> 5
```

Closure

```
val addMore = (x: Int) => x + more

val more = 9999

addMore(10) // 10009


val someNumbers = List(-11, -10, 0 , 10)

var sum = 0

someNumbers.foreach(sum += _)

sum // -11
```

```
def make increaser(more: Int) = (x: Int) =>
x+more

val inc1 = increaser(1)

val inc9999 = increaser(9999)

inc1(10) // 11

inc9999(10) // 10009
```

Passagem de parâmetro

```
def imprime( a:Int, b:Int ) = {
   println("Valor a: " + a );
   println("Valor b: " + b );
}

imprime(7, 5);
imprime(b = 5, a = 7);
```

```
>> Valor a: 7
>> Valor b: 5
>> Valor a: 7
>> Valor b: 5
```

Parâmetros Default

```
def add( a:Int = 5, b:Int = 7 ) : Int = {
        var soma:Int = 0;
        soma = a + b
        return soma;
}
println(add(10));
```

```
>> 17
```

Argumentos variáveis

```
def printStrings(args:String*) = {
        var i : Int = 0;
        for( arg <- args ){
                println("Arg[" + i + "] = " + arg );
                i = 1 + i;
        }
}

printStrings("Gustavo", "Felipe", "Gilmarllen");
```

```
>> Arg[0] = Gustavo
>> Arg[1] = Felipe
>> Arg[2] = Gilmarllen
```

Condicionais

```
import scala.util.Random
val x: Int = Random.nextInt(10)
if(x < 5){
       println("if -> " + x);
}
else if(x <= 9){
        println("else if -> "+ x);
}
else{
 println("else -> " + x);
}
```

```
>> else if -> 5
```

Condicionais

```
var cor = if(a == b)
                "vermelho"
          else
              "azul"

var cor = if(a == b) "vermelho" else "azul"
```

Condicionais/Pattern Matching

```
def matchTest(x: Int): String = {
        x match {
              case 1 => "um"
              case 2 => "dois"
              case _ => "outro"
        }
}
var x = matchTest(3)
println(x);
```

```
>> outro
```

Pattern Matching

```
case class Person(name: String, age: Int)

var person = Person ("Gilmarllen", 20)
person match {
      case Person("Felipe", 26) => println("Oi Felipe!")
      case Person("Gilmarllen", _) => println("Oi Gilmarllen!")
      case _ => println("Quem é você")
}
```

```
>> Oi Gilmarllen!
```

while

```
var x = 1;
while( x <= 5 ){
    println( "Valor: " + x);
    x = x + 1;
}
```

```
>> Valor: 1
>> Valor: 2
>> Valor: 3
>> Valor: 4
>> Valor: 5
```

do-while

```
var x = 1;
do{
    println( "Valor: " + x );
    x = x + 1;
}while( x <= 5)
```

```
>> Valor: 1
>> Valor: 2
>> Valor: 3
>> Valor: 4
>> Valor: 5
```

for

```
for(x <- 1 to 5){
    println( "Valor: " + x);
}
```

```
>> Valor: 1
>> Valor: 2
>> Valor: 3
>> Valor: 4
>> Valor: 5
```

for

```
val list =List("a","b","c","d");
for(x <- list){
      println( "Valor: " + x);
}
```

```
>> Valor: a
>> Valor: b
>> Valor: c
>> Valor: d
```

for

```
val list = List(1,2,3,4,5);
var a = for(x <- list if x%2 == 0) yield x
println(a);
```

>> List(2,4)

break

```
import util.control.Breaks._
val list = List(1,2,3,4,5);
breakable {
      for(x <- list){
            println( "Valor: " + x);
            if(x == 4){
                  break;
            }
       }
}
```

```
>> Valor: 1
>> Valor: 2
>> Valor: 3
>> Valor: 4
```

break

```
import util.control._
val list = List(1,2,3,4,5);
val primeiro = new Breaks;
val segundo = new Breaks;
primeiro.breakable {
      for(x <- list){
           println("Primeiro: " + x);
           segundo.breakable {
      segundo breakable.
                for(y <- list){
                  println("Segundo: " + y);
                  if(y == 2){
                    primeiro.break;
                  }
                }
              } // segundo breakable
            }
} //primeiro breakable
>>Primeiro: 1
>>Segundo: 2
>>Segundo: 2
```

Operações com listas

```
var x = 1::2::3::Nil

var y = List.range(1,5)

var z = List.range(0,10,2)

var a  = List.fill(3)("LP")

var b = List.tabulate(5)(n => n + 1)

var fil = b.filter(a => a % 2 == 0)

val c = fil.map(a => a * 2)
```

```
>> x: List[Int] = List(1, 2, 3)
>> y: List[Int] = List(1, 2, 3, 4)
>> z: List[Int] = List(0, 2, 4, 6, 8)
>> a: List[String] = List(LP, LP, LP)
>> b: List[Int] = List(1, 2, 3, 4, 5)
>> fil: List[Int] = List(2,4)
>> c: List[Int] = List(4,8)
```

Currying

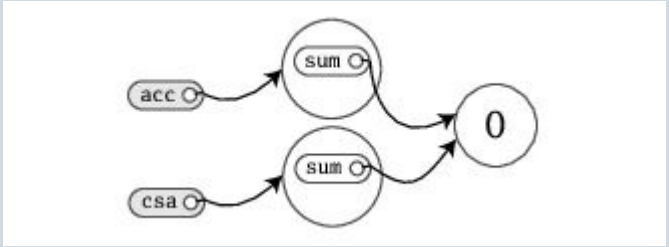
```
def filter(xs: List[Int], p: Int => Boolean): List[Int] =
    if (xs.isEmpty) xs
    else if (p(xs.head)) xs.head :: filter(xs.tail, p)
    else filter(xs.tail, p)

def modN(n: Int)(x: Int) = ((x % n) == 0)
val nums = List(1, 2, 3, 4, 5, 6, 7, 8)
println(filter(nums, modN(2)))
```

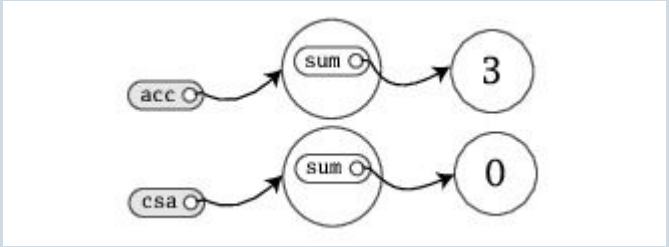
```
>> List(2,4,6,8)
```

Modularização

Classes

```
class ChecaAcumuladorSoma {
    var soma = 0 // private var soma = 0
    def add(b: Byte) { sum += b }
    def checaSoma: Int = ~(soma & 0XFF) + 1
}
val acc = new ChecaAcumuladorSoma()
val acs = new ChecaAcumuladorSoma()
acc.soma = 3
```





Objetos Singletons

```
import scala.collection.mutable.Map
object ChecaAcumuladorSoma {
    private val cache = Map[String, Int]()
    def calcula(s: String) =
        if (cache.contains(s))
            cache(s)
        else {
            val acc = new
            ChecaAcumuladorSoma
            for (c <- s)
                acc.add(c.toByte)
            val cs = acc.checaSoma()
            cache += (s -> cs)
            cs
        }
}
```

Main

```
import ChecaAcumuladorSoma.calcula
object Summer{
    def main(args: Array[String]) {
        for (arg <- args)
            println(arg + ": " + calcula(arg))
    }
}
```

```
scalac Summer.scala Checa.scala
// fsc Summer.scala Checa.scala
scala Summer of love
// of: -213
// love: -182
```

Traits

```
trait Philosophical {
    def philosophize() {
        println("I consume memory.")
    }
}
class Frog extends Philosophical {
    override def toString = "green"
}
```

```
val frog = new Frog
frog.philosophize() // I consume memory.

val phil: Philosophical = frog
phil.philosophize() // I consume memory.
```

Traits e SuperClass

```
class Animal

trait HasLegs

class Frog extends Animal with Philosophical
with HasLegs {

    override def toString = “green”

    override def philosophize() {

        println(“It ain’t easey being “ +
toString

    }

}
```

```
val phrog: Philosophical = new Frog

phrog.philosophize() // It ain’t easy being
green
```

Ordered

```
class Rational(n: Int, d: Int) extends
Ordered[Rational] {
    // …
    def compare(that: Rational) =
        (this.numere * that.denom) -
(that.numer * this.denom)
}
```

```
val half = new Rational(1,2)
val third = new Rational(1,3)
half < third // false
half > third // true
```

Modificações Empilháveis

```
abstract class IntQueue {
    def get(): Int
    def put(x: Int)
}
```

```
import scala.collection.mutable.ArrayBuffer

class BasicIntQueue extends IntQueue {
    private val buf = new ArrayBuffer[Int]
    def get() = buf.remove(0)
    def put(x: Int) { buf += x }
}
```

Empilhando Traits

```
trait Doubling extends IntQueue {
    abstract override def put(x: Int) {
super.put(2*x)
}
trait Filtering extends IntQueue {
    abstract override def put(x: Int) {
        if(x >= 0) super.put(x)
    }
}
```

```
class MyQueue extends BasicIntQueue with
Doubling with Filtering
val queue = new MyQueue
queue.put(-1); queue.put(0); queue.put(1)
queue.get() // 1
queue.get() // 2
```

Herança Múltipla?

```
// Hipotético

trait MyQueue extends BasicIntQueue with
Doubling with Filtering {

    def put(x: Int) {

        Doubling.super.put(x)

        Filtering.super.put(x)

    }

}
```

```
// Hipotético

val q = new BasicIntQueue with Doubling
with Filtering

q.put(42) // Qual super seria chamado?
```

Trait ou Classe

Não Reutilizável	Classe
Reutilizável	Trait
Utilizar no Java	Trait
Distribuição em forma compilada*	Classe abstrata
Eficiência**	Classe
Nenhuma das opções anteriores	Traits

Case Class

```
sealed abstract class Expr

case class Var(name: String) extends Expr

case class Number(num: Double) extends Expr

case class UnOp(operator: String, arg: Expr) extends Expr

case class BinOp(operator: String, left: Expr, right: Expr) extends Expr
```

```
val v = Var("x")

val op = BinOp("+", Number(1), v)

v.name // String = x

op.left // Expr = Number(1.0)

println(op) // BinOp(+, Number(1.0), Var(x))

op.right == Var("x") // Boolean = true

op.copy(operator = "-") // BinOp(-, Number(1.0), Var(x))
```

Pacotes

```
package bobsrockets {
    package navigation {
        package launch {
            class Booster1
        }
        class MissionControl {
            val booster1 = new
launch.Booster1
            val booster2 = new
            bobsrockets.launch.Booster2
        }
    }
    package launch {
        class Booster2
    }
}
```

Import

import java.io.File	importa a classe File
import java.io._	importa todas as classes no pacote
import java.io.{File, IOException}	importa as classes File e IOException
import java.util.{List => UtilList}	importa classe List e a renomeia
import java.util.{Random => _, _}	importa todas as classes, menos Random

Polimorfismo

Inclusão e Paramétrico

```
// Exemplo de Inclusão
class NowUCMe extends NowUDont {
    // …
}
```

```
// Exemplo de paramétrico
import java.util.{ArrayList}
val listint = ArrayList[Int]
```

Sobrecarga e Coerção

```
// Exemplo Sobrecarga
class Bicicleta
class Carro
class Viação {
    var dist = 0
    def movimenta(x: Carro) { dist += 10 }
    def movimenta(x: Bicicleta) { dist += 5 }
}
```

```
// Exemplo Coerção
val x = 2.0
val y = x.asInstanceOf[Int]
```

Concorrência

Actors

- Thread-like;
- Mailbox;
- scala.actors;
- Função parcial receive;
- Não bloqueante;
- Função parcial react;

Actor exemplo

```
object SeriusActor extends Actor {
    def act() {
        loop {
            react {
                case x: Int =>
                    println(“Got “ + x)
            }
        }
    }
}
```

```
SeriousActor.start()
SeriousActor ! “Hello Actor”
SeriousActor ! math.Pi
SeriousActor ! 12 // Got 12
```

Exceções

```
import java.io.FileReader
import java.io.FileNotFoundException
import java.io.IOException
try {
      val f = new FileReader("input.txt");
}
catch {
      case ex: FileNotFoundException => {
            println("Arquivo não encontrado");
      }
      case ex: IOException => println("IOException");
}
finally {
 println("finally...");
}

// Arquivo não encontrado
// finally...
```

Throw

```
try{
  throw new IllegalStateException("Exceção Lançada");
  println("Essa linha não é executada");
}
catch {
  case e: Exception => println(e);
}
```

```
java.lang.IllegalStateException:
Exceção Lançada
```

```
Inicio
java.lang.IllegalStateException: Exceção Lançada
Fim
```

```
object Main {
  @throws(classOf[IllegalStateException])
  def sub1() = {
    throw new IllegalStateException("Exceção
Lançada");
    println("Essa linha não é executada");
}
  def sub2() = {
    println("Inicio");
    sub1();
    println("Essa linha não é executada");
 }
```

```
def sub3() = {
  try{
    sub2();
}
  catch{
     case e:Exception => println(e);
  }
}
def main(args: Array[String]): Unit ={
sub3();
println("Fim");
}
```

Avaliação da linguagem

Critérios Gerais	C	JAVA	Scala
Aplicabilidade	Sim	Sim	Parcial
Confiabilidade	Não	Sim	Sim
Aprendizado	Não	Não	Não
Eficiência	Sim	Não	Não
Portabilidade	Não	Sim	Sim
Método de projeto	Estruturado	OO	OO, Funcional
Evolutibilidade	Não	Sim	Sim

Critérios Gerais	C	JAVA	Scala
Reusabilidade	Parcial	Sim	Sim
Integração	Sim	Parcial	Parcial
Escopo	Sim	Sim	Sim
Expressões e Comandos	Sim	Sim	Sim
Tipos primitivos e compostos	Sim	Sim	Sim
Memória	Programador	Sistema	Sistema

Critérios Gerais	C	JAVA	Scala
Persistência de dados	Biblioteca de funções	JDBC, biblioteca de classes, serialização	Serialização, biblioteca de classes
Passagem de parâmetros	Lista variável e por valor	Lista variável, por valor e por cópia de referência	Lista variável, por nome, por cópia de referência
Encapsulamento e proteção	Parcial	Sim	Sim
Sistema de tipos	Não	Sim	Sim
Verificação de tipos	Estática	Estática / Dinâmica	Estática / Dinâmica

Critérios Gerais	C	JAVA	Scala
Polimorfismo	Coerção e sobrecarga	Todos	Todos
Exceções	Não	Sim	Sim
Concorrência	Não	Sim	Sim