SCALA

Igor,Kaique, Lucas Bergantini, Mateus Souza, Rebeca

Histórico

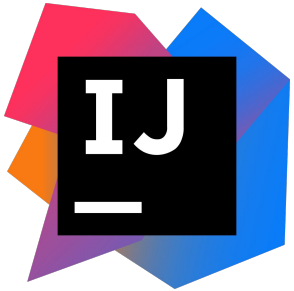
- Criada por Martin Odersky em 2001
- Primeira aparição em janeiro de 2004
- Influênciada por Java e C#
- Licenciada pela BSD 3-clause;

Visão Geral

- Multiparadigma
- Roda na JVM, Browser, LLVM, CLR
- Twitter, FourSquare, GitHub, PicPay
- Última versão estável é de 27 de Setembro de 2018

Primeiros passos

- Para desenvolver
  - JDK 8
  - IntelliJ ou terminal usando scalac
  - `$java -jar scala-2.9.0.1-installer.jar`
  - `scala`



Sintaxe Básica(1)

- Criando uma variável :
  - `var Nome`

- Criando metódos:
  - `def nomeMétodo (nomeParametro: Int) =`
  - `def nomeMétodo (nomeParametro: Int): Unit = {}`

Sintaxe Básica(2)

- Criando classes:

```
class Point (x: Int, y: Int) {
   def print (): Unit = println(x, y)

   def increment (dx: Int, dy: Int): Unit = {
       x = x + dx //erro: Reassignment to val
       y = y + dy //erro: Reassignment to val
   }
}

// removendo a função increment, poderíamos fazer:
val p = new Point(10, 10).x //error: value x is not a member of Point
```

```
class Point (val x: Int, val y: Int) {
   def print (): Unit = println(x, y)

   def increment (dx: Int, dy: Int): Unit = {
       x = x + dx //erro: Reassignment to val
       y = y + dy //erro: Reassignment to val
   }
}

//removendo a função increment, poderíamos fazer:
val p = new Point(10, 10).x // p: Int = 10
```

```
class Point (var x: Int, var y: Int) {
   def print (): Unit = println(x, y)

   def increment (dx: Int, dy: Int): Unit = {
       x = x + dx
       y = y + dy
   }
}

val p = new Point(10, 10)
p.increment(1, 2)
val j = p.x // j: Int = 11
```

Sintaxe Básica(3)

- Nome do arquivo da classe deve ser exatamente o nome da classe
  - `class Point`
  - Point.scala

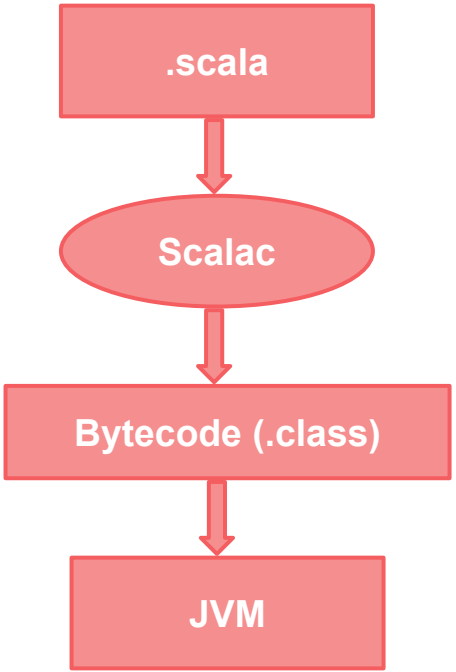
- Todos os programas em Scala começam pela main
  - `def main(args: Array[String])`

Scala X JAVA

```
class PrintOptions {
     public static void main(String[] args) {
          System.out.println("Options selected:");
          for (int i = 0; i < args.length; i++)
               if (args[i].startsWith("-"))
                    System.out.println(" "+args[i].substring(1));
     }
}

object PrintOptions {
     def main(args: Array[String]): Unit = {
          System.out.println("Options selected:")
          for (val arg <- args)
               if (arg.startsWith("-"))
                    System.out.println(" "+arg.substring(1))
     }
}
```

Scala X JAVA



Amarrações

Identificadores

- Tipos, valores, métodos, classes

- Case Sensitive
  - `var nome` é diferente de `var NOME`

Identificadores

- Alfanuméricos
- Operadores
- Misturas de operadores com caracteres alfanuméricos
- Literais

Identificadores

- Válidos
  - Sequência de números ou operadores precedidos de letras ou *underscore*
    - `a_C_?`
  - Sequência de operadores
    - `?<>!~`

Identificadores

- Nomes de **classes** primeira letra maiúscula
  - `class Aluno`

- Nomes de **métodos** primeira letra minuscula
  - **def** comparaAlunos

- Se forem compostos, cada inicial deve começar com maiúscula (*lower camelcase*)

Palavras Reservadas

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

Amarrações

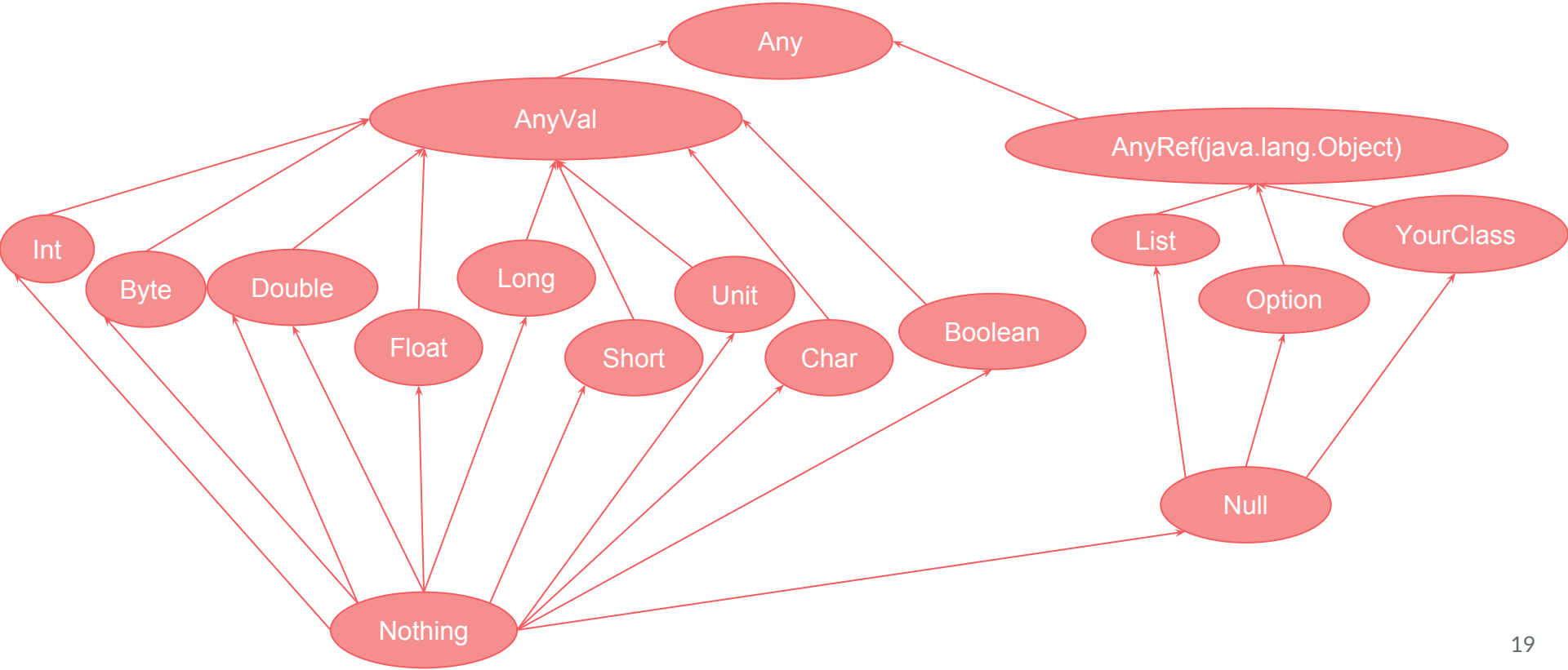
- Dois espaços de nomes diferentes (**tipos e nomes**)
- Escopo estático e aninhados
- Sombreamento resolvido

Valores e tipos de dados

Tipos de dados

- Tudo é objeto (sem tipos primitivos)
- Fortemente tipada
- Coversão implicita
- Inferencia de tipos

Tipos de Dados



Tipos de Dados

Tipo de dado	Descrição
String	Cadeia de chars
Boolean	Literais true ou false
Unit	Corresponde a sem valor
Null	Referência null ou uma referência vazia
Nothing	Supertipo de todos os outros inclui nenhum valor
Any	Supertipo de qualquer tipo
AnyRef	Supertipo de todos os tipos referência

Tipo de Dado	Descrição
Byte	Valor de 8 bits com sinal (-128 a 127)
Short	Valor de 16 bits com sinal (-32768 a 32767)
Int	Valor de 32 bits com sinal (-2147483648 a 2147483647)
Long	Valor de 64 bits com sinal (-9223372036854775808 a 9223372036854775807)
Float	Ponto flutuante de 32 bits (IEEE 754)
Double	Ponto flutuante de 64 bits (IEEE 754)
Char	Unicode com 16 sem sinal

Variáveis e Constantes

Declaração

[escopo] [tipo de acesso] [nome da variável]: [tipo da variável] = [Valor padrão]

private val nome: String = “Mateus”

Declaração de variável

var
Mutável

```
scala> var x = 1
x: Int = 1
scala> println(x)
1
scala> x = 2

x: Int = 2
scala> println(x)
2
```

Declaração de variável

val

Imutável, constante

```
scala> val y = 1
y: Int = 1
scala> y = 2
<console>:12: error: reassignment
to val                    y = 2
^
```

Modificadores de Acesso

- **Public:** Visibilidade aberta. Padrão caso não esteja explícita;
- **Protected:** Visibilidade no pacote e subclasses;
- **Private:** Visibilidade apenas para a própria classe.

Inferência de Tipos

A inferência de tipos é a capacidade do compilador entender/adivinhar qual é o tipo de dados de determinada variável sem ela ter sido declarada no código escrito.

Inferência de Tipos

```
var x = 1

val nome = “LP”;
```

```
var x : Int = 1

val nome : String = “LP”;
```

Inferência de Tipos e Tipagem Estática

Uma vez determinado, o tipo da variável não pode ser alterado

```
scala> var x = 1
x: Int = 1
scala> x = "Um"
<console>:12: error: type mismatch;
found   : String("Um")
required: Int
x = "Um"
^
```

Alocação e Desalocação

- Alocação feita em pilha e/ou monte dependendo do caso
- Segue a estratégia utilizada pela JVM para alocar e desalocar recursos

Armazenamento em memória Secundária

Possui o trait *Serializable* que pode ser extendido pela classe que deseja-se serializar

```
import java.io._

@SerialVersionUID(123L)

class Teste extends Serializable
```

Escrita e Leitura em arquivo

Escrita

```
import java.io._

val file = new File("exemplo.txt" );

val writer = new PrintWriter(file);

writer.write("Exemplo");

writer.close();
```

Escrita e Leitura em arquivo

Leitura

```
import scala.io.Source

val filename = "exemplo.txt"

for (line <-
Source.fromFile(filename).getLines)
{

println(line)

}
```

Expressões e Comandos

Operadores

Aritméticos:	Relacionais:	Lógicos:	Bit a bit:
+	==	!	&
-	!=	||	^
/	<	&&	|
*	>		>>
%	<=		<<
	>=		>>>

- Os operadores são os mesmos da linguagem Java.

Operadores

- Scala não possui os operadores ++ e --

valor += 1
valor -= 1

- Operadores são métodos

```
println(i + 2)      =     println(i.+(2))

x = 10 % (8/2)      =     x = 10.%(8./(2))
```

Curto-circuito

- Algumas expressões lógicas podem sofrer curto-circuito em casos em que não há necessidade de verificar o resto da expressão.

```
var A = false
var B = true

if (A && x.funcao(y)){
  //...
}

if (B || w.funcao(i)){
  //...
}
```

curto!

Condicionais

- **if** : Scala só aceita Boolean como condição, então se tentarmos usar um inteiro na condição dará erro de compilação.

```
var num = 10
  if(num)
    //..
```

```
if(num != 0)
   //..
```

- **Match** :

```
def matchTest(x: Any): Any = x match {
    case 1 => "um"
    case "dois" => 2
    case y: Int => "scala.Int"
}
```

Loops

```
for(i <- 0 to 10)
    println(i)

val frase = "ola, tudo bem?"

for(i <- 0 until frase.length)
  println(frase(i))

val lista = List(1, 2, 3, 4, 5, 6, 7)

for(i <- lista)
  println("numeros da lista" + i)

var impares = for { i <- 1 to 20 if (i % 2) == 0} yield i
```

O while usa o formato clássico while(Boolean)

Parâmetros

- Valor default:

```
def soma( num1:Int = 1, num2:Int = 1) : Int = {
  return num1 + num2
}
println("5 + 4 = " + soma(5,4))              //saida: 5 + 4 = 9
println("5 + 4 = " + soma())                 //saida: 5 + 4 = 2
```

- Argumentos nomeados(correspondência por palavra-chave):

```
println("5 + 4 = " + getSum(num2=5, num1=4))
```

Parâmetros

- Suporta parâmetros com tamanho variado:

```
def imprime(palavras: String*){
    palavras.map(println)
  }

  imprime("ola! ", "mas que ", "belo dia!")

//saida: ola! mas que belo dia!
```

Facilidades de Output

```
val frase1 = "tudo bem?"
val peso = 10.5

println(s"ola! $frase1")       //saida: ola! Tudo bem?

println(f"peso: $peso%.2f")    //saida: peso: 10.50

println(frase1(1))             //saida: u

val frase2 = "bom dia!"

println("ola! " + frase1 + " " + frase2)  //saida: ola! Tudo bem? bom dia!
```

Função de Ordem Superior

```
class Decorador(left: String, right: String) {
  def layout[A](x: A) = left + x.toString() + right
}

object Test extends App {
  def apply(f: Int => String, v: Int) = f(v)
  val decorator = new Decorador("--> ", " <--")
  println(apply(decorator.layout, 1234))
}


//saida: --> 1234 <--
```

Funções Aninhadas

- Em scala é possível aninhar definições de funções.

```
  def fatorial(x: Int): Int = {
    def fat(x: Int, acomula: Int): Int = {
      if (x <= 1) acomula
      else fat(x - 1, x * acomula)
    }
    fat(x, 1)
 }

 println("Fatorial de 5 = " + fatorial(5))    //saida: Fatorial de 5 = 120
```

Traits

```
trait Semelhante {
  def ehIgual(x: Any): Boolean
  def naoEhIgual(x: Any): Boolean = !ehIgual(x)
}

class Ponto(xc: Int, yc: Int) extends Semelhante {
  var x: Int = xc
  var y: Int = yc
  def ehIgual(obj: Any) =
    obj.isInstanceOf[Ponto] &&
    obj.asInstanceOf[Ponto].x == x &&
    obj.asInstanceOf[Ponto].y == y                    Uso: p1.ehIgual(p2)
}
```

Polimorfismo

Tipos de polimorfismo

- Coerção
- Sobrecarga
- Paramétrico
- Inclusão.

Coerção

```
val x = 4.0

val y = x.asInstanceOf[Int]
```

Sobrecarga

```
class CarroNormal

class CarroLuxo

 class Concessionária {

      var preço = 0

          def aumenta(x: CarroNormal) { preco += 200 }

          def aumenta(x: CarroLuxo) { preco += 5000 }

 }
```

Paramétrico

```
import java.util.{ArrayList}

val listint = ArrayList[Int]
```

Inclusão

```
class Pagina extends Livro {

        ...

}
```

Exceções

Try/Catch/Throw/Finally

- O tratamento de exceções em SCALA é muito similar ao de Java, apesar de não possuir exceções checadas.

```
object hello
{
  def main(args:Array[String])
    {
          var a:Int=12
          var b:Int=0
          print (a/b)
    }
}
```

Throw

- O tratamento de exceções em SCALA é muito similar ao de Java, apesar de não possuir exceções checadas.
- Para lançar um exception fazemos como em Java com a cláusula Throw

```
Throw new Exception
```

Try/Catch

- O tratamento de exceções em SCALA é muito similar ao de Java.
- Utiliza blocos Try - Catch

```
object hello
{
  def main(args:Array[String])
    {
          var a:Int=12
          var b:Int=0
          try{
                    print (a/b)
           }
          catch{
                    case ex:ArithmeticException => print("Ocorreu exceção")
          }
    }
}
```

Pattern Matching

- Uma diferença é que a linguagem utiliza o “pattern matching” no bloco catch para identificar as exceções

```
catch{
        case ex:ArithmeticException => print("Ocorreu exceção")
}
```

Pattern Matching

- Uma diferença é que a linguagem utiliza o “pattern matching” no bloco catch para identificar as exceções

```
import scala.util.Random

val x: Int = Random.nextInt(10)

x match {
  case 0 => "zero"
  case 1 => "one"
  case 2 => "two"
  case _ => "many"
}
```

Outro exemplo

- No caso de estarmos manipulando arquivos, várias coisas podem dar errado

```
import scala.swing.FileChooser
import java.io.FileInputStream
import java.io.FileNotFoundException
import scala.swing.Dialog

object Exemplo{
	def main (args: Array[String]): Unit = {
		val chooser = new FileChooser
		if (chooser.showOpenDialog(null)) == FileChooser.Result.Approve) {
			val file = chooser.selectedFile
			val fis = new FileInputStream(file)
			val buf = new Array[Byte](file.length.toInt)
			fis.read(buf)
			fis.close()}
	}
}
```

Outro exemplo

- No caso de estarmos manipulando arquivos, várias coisas podem dar errado

```
import scala.swing.FileChooser
import java.io.FileInputStream
import java.io.FileNotFoundException
import scala.swing.Dialog

object Exemplo{
        def main (args: Array[String]): Unit = {
                val chooser = new FileChooser
                if (chooser.showOpenDialog(null)) == FileChooser.Result.Approve) {
                        val file = chooser.selectedFile
                        try{
                                val fis = new FileInputStream(file)
                                val buf = new Array[Byte](file.length.toInt)
                                fis.read(buf)
                        fis.close()
                        }catch{
                                Case ex: FileNotFoundException => print("Arquivo nao encontrado")
                                Case ex: IOException=> print("Erro ao ler arquivo")
                        }
        }
}
```

Finally

- Como em Java, também há o Finally

```
import java.io.FileReader
import java.io.FileNotFoundException
import java.io.IOException

object Demo {
   def main(args: Array[String]) {
      try {
         val f = new FileReader("input.txt")
      } catch {
         case ex: FileNotFoundException => {
            println("Missing file exception")
         }

         case ex: IOException => {
            println("IO Exception")
         }
      } finally {
         println("Exiting finally...")
      }
   }
}
```

Concorrência

Concorrência

- É dito que linguagens funcionais são boas para solução de concorrência devido à imutabilidade de suas estruturas.

- No momento a linguagem SCALA possui dois principais estratégia para abordar a concorrência
  - As *Threads* (derivada de Java)
  - Abordagem *type safe* utilizando *Actors* (inspiradas em Earlang)

Actors

- Neste caso o programa printa de 0 a 10, um número por segundo.

```
import scala.actors._
object CountingActor extends Actor {
    def act() {
        for (i <- 1 to 10) {
            println("Number: "+i)
            Thread.sleep(1000)
        }
    }
}
CountingActor.start()
```

Actors

- A última linha inicia o *Actor*, que implicitamente chama o método *act*

```
import scala.actors._
object CountingActor extends Actor {
    def act() {
        for (i <- 1 to 10) {
            println("Number: "+i)
            Thread.sleep(1000)
        }
    }
}
CountingActor.start()
```

Actors

- Esse *actor* não responde mensagens de outros *actors*

```
import scala.actors._
object CountingActor extends Actor {
    def act() {
        for (i <- 1 to 10) {
            println("Number: "+i)
            Thread.sleep(1000)
        }
    }
}
CountingActor.start()
```

Actors

- Um exemplo de actor que responde mensagens

```
import scala.actors._
val echoActor = Actor {
      while (true) {
          receive{
          case msg => println(“received: “ + msg)
       }
   }
}

echoActor ! “Hello”
echoActor ! “world”
```

Actors

- O objeto fica em loop, que é bloqueado enquanto não chega uma mensagem
- O receive da match em qualquer mensagem.

```
import scala.actors._
val echoActor = Actor {
      while (true) {
          receive{
          case msg => println(“received: “ + msg)
       }
   }
}
```

Outros métodos de concorrência

- Além das threads e dos actors já citados, SCALA também conta com outros métodos para concorrência, dentre eles:
    - Semáforos
    - Monitores
    - Futures
    - Computação paralela

Avaliação de Linguagens

Critérios Gerais	C	C++	Java	Scala
Aplicabilidade	Sim	Sim	Parcial	Parcial
Confiabilidade	Não	Não	Sim	Sim
Aprendizado	Não	Não	Não	Parcial
Eficiência	Sim	Sim	Parcial	Parcial
Portabilidade	Não	Não	Sim	Sim
Método de Projeto	Estruturado	Estruturado e OO	OO	Funcional e OO
Evolutibilidade	Não	Parcial	Sim	Sim
Reusabilidade	Parcial	Sim	Sim	Sim
Integração	Sim	Sim	Parcial	
Custo	Depende da Ferramenta	Depende da Ferramenta	Depende da Ferramenta	Depende da Ferramenta

Critérios Gerais	C	C++	Java	Scala
Aplicabilidade	Sim	Sim	Parcial	Parcial
Confiabilidade	Não	Não	Sim	Sim
Aprendizado	Não	Não	Não	Parcial
Eficiência	Sim	Sim	Parcial	Parcial
Portabilidade	Não	Não	Sim	Sim
Método de Projeto	Estruturado	Estruturado e OO	OO	Funcional e OO
Evolutibilidade	Não	Parcial	Sim	Sim
Reusabilidade	Parcial	Sim	Sim	Sim
Integração	Sim	Sim	Parcial	
Custo	Depende da Ferramenta	Depende da Ferramenta	Depende da Ferramenta	Depende da Ferramenta

Falta controle direto de Hardware

Critérios Gerais	C	C++	Java	Scala
Aplicabilidade	Sim	Sim	Parcial	Parcial
Confiabilidade	Não	Não	Sim	Sim
Aprendizado	Não	Não	Não	Parcial
Eficiência	Sim	Sim	Parcial	Parcial
Portabilidade	Não	Não	Sim	Sim
Método de Projeto	Estruturado	Estruturado e OO	OO	Funcional e OO
Evolutibilidade	Não	Parcial	Sim	Sim
Reusabilidade	Parcial	Sim	Sim	Sim
Integração	Sim	Sim	Parcial	
Custo	Depende da Ferramenta	Depende da Ferramenta	Depende da Ferramenta	Depende da Ferramenta

Centraliza operações para reduzir problemas

Critérios Gerais	C	C++	Java	Scala
Aplicabilidade	Sim	Sim	Parcial	Parcial
Confiabilidade	Não	Não	Sim	Sim
Aprendizado	Não	Não	Não	Parcial
Eficiência	Sim	Sim	Parcial	Parcial
Portabilidade	Não	Não	Sim	Sim
Método de Projeto	Estruturado	Estruturado e OO	OO	Funcional e OO
Evolutibilidade	Não	Parcial	Sim	Sim
Reusabilidade	Parcial	Sim	Sim	Sim
Integração	Sim	Sim	Parcial	
Custo	Depende da Ferramenta	Depende da Ferramenta	Depende da Ferramenta	Depende da Ferramenta

Para usos mais avançados se torna complexa

Critérios Gerais	C	C++	Java	Scala
Aplicabilidade	Sim	Sim	Parcial	Parcial
Confiabilidade	Não	Não	Sim	Sim
Aprendizado	Não	Não	Não	Parcial
Eficiência	Sim	Sim	Parcial	Parcial
Portabilidade	Não	Não	Sim	Sim
Método de Projeto	Estruturado	Estruturado e OO	OO	Funcional e OO
Evolutibilidade	Não	Parcial	Sim	Sim
Reusabilidade	Parcial	Sim	Sim	Sim
Integração	Sim	Sim	Parcial	
Custo	Depende da Ferramenta	Depende da Ferramenta	Depende da Ferramenta	Depende da Ferramenta

Semelhante a Java

Critérios Gerais	C	C++	Java	Scala
Aplicabilidade	Sim	Sim	Parcial	Parcial
Confiabilidade	Não	Não	Sim	Sim
Aprendizado	Não	Não	Não	Parcial
Eficiência	Sim	Sim	Parcial	Parcial
Portabilidade	Não	Não	Sim	Sim
Método de Projeto	Estruturado	Estruturado e OO	OO	Funcional e OO
Evolutibilidade	Não	Parcial	Sim	Sim
Reusabilidade	Parcial	Sim	Sim	Sim
Integração	Sim	Sim	Parcial	
Custo	Depende da Ferramenta	Depende da Ferramenta	Depende da Ferramenta	Depende da Ferramenta

JVM e .NET

Critérios Gerais	C	C++	Java	Scala
Aplicabilidade	Sim	Sim	Parcial	Parcial
Confiabilidade	Não	Não	Sim	Sim
Aprendizado	Não	Não	Não	Parcial
Eficiência	Sim	Sim	Parcial	Parcial
Portabilidade	Não	Não	Sim	Sim
Método de Projeto	Estruturado	Estruturado e OO	OO	Funcional e OO
Evolutibilidade	Não	Parcial	Sim	Sim
Reusabilidade	Parcial	Sim	Sim	Sim
Integração	Sim	Sim	Parcial	
Custo	Depende da Ferramenta	Depende da Ferramenta	Depende da Ferramenta	Depende da Ferramenta

Linguagem escalável

Critérios Gerais	C	C++	Java	Scala
Aplicabilidade	Sim	Sim	Parcial	Parcial
Confiabilidade	Não	Não	Sim	Sim
Aprendizado	Não	Não	Não	Parcial
Eficiência	Sim	Sim	Parcial	Parcial
Portabilidade	Não	Não	Sim	Sim
Método de Projeto	Estruturado	Estruturado e OO	OO	Funcional e OO
Evolutibilidade	Não	Parcial	Sim	Sim
Reusabilidade	Parcial	Sim	Sim	Sim
Integração	Sim	Sim	Parcial	
Custo	Depende da Ferramenta	Depende da Ferramenta	Depende da Ferramenta	Depende da Ferramenta

Classes, pacotes, polimorfismo universal

Critérios Gerais	C	C++	Java	Scala
Aplicabilidade	Sim	Sim	Parcial	Parcial
Confiabilidade	Não	Não	Sim	Sim
Aprendizado	Não	Não	Não	Parcial
Eficiência	Sim	Sim	Parcial	Parcial
Portabilidade	Não	Não	Sim	Sim
Método de Projeto	Estruturado	Estruturado e OO	OO	Funcional e OO
Evolutibilidade	Não	Parcial	Sim	Sim
Reusabilidade	Parcial	Sim	Sim	Sim
Integração	Sim	Sim	Parcial	Parcial
Custo	Depende da Ferramenta	Depende da Ferramenta	Depende da Ferramenta	Depende da Ferramenta

Critérios Específicos	C	C++	Java	Scala
Escopo	Sim	Sim	Sim	Sim
Expressões e comandos	Sim	Sim	Sim	Sim
Tipos primitivos e compostos	Sim	Sim	Sim	Parcial
Gerenciamento de memória	Programador	Programador	Sistema	Sistema
Persistência de Dados	Biblioteca e funções	Biblioteca de classes e funções	JDBC, biblioteca de classes, serialização	Biblioteca de classes, serialização
Passagem de parâmetros	Lista variável e por valor	Lista variável, default, por valor e por referência	Lista variável, por valor e por cópia de referência	Lista variável, por nome, por cópia de referência

Critérios Específicos	C	C++	Java	Scala
Encapsulamento e proteção	Parcial	Sim	Sim	Sim
Sistema de tipos	Não	Parcial	Sim	Sim
Verificação de tipos	Estática	Estática / Dinâmica	Estática / Dinâmica	Estática / Dinâmica
Polimorfismo	Coerção e sobrecarga	Todos	Todos	Todos
Exceções	Não	Parcial	Sim	Sim
Concorrência	Não	Sim	Sim	Sim

Referências

- https://www.tutorialspoint.com/scala/index.htm
- https://www.scala-lang.org/docu/files/ScalaOverview.pdf
- https://www.scala-lang.org/
- https://www.devmedia.com.br/conheca-a-linguagem-scala/32850
- http://www.newthinktank.com/2015/08/learn-scala-one-video/
- https://accbel.wordpress.com/2013/09/02/o-basico-em-scala-parte-2/
- https://web.archive.org/web/20121018000226/http://blog.objectmentor.com/articles/2008/08/14/the-seductions-of-scala-part-iii-concurrent-programming