UFRPE
Universidade
Federal Rural
de Pernambuco

Paradigmas de Programação

Introdução à linguagens funcionais com Scala

Kellyton Brito

UFRPE
Universidade
Federal Rural
de Pernambuco

Agenda

- O que é a linguagem
- Principais diferenças com Java
- Instalando
- Implementação

O que é a linguagem

Object-Oriented Meets Functional

Have the best of both worlds. Construct ***elegant class hierarchies*** for maximum ***code reuse and extensibility***, implement their behavior using ***higher-order functions***. Or anything in-between.

scala-lang.org

Scala – O que é

- Desenvolvida em 2001
  - École Polytechnique Fédérale de Lausanne (EPFL), Suíça.
- Linguagem de propósito geral
- Puramente Orientada a Objetos e Funcional
- Roda sobre a JVM

Linguagem Scala - Adoção

- Twitter (2009), Foursquare, Coursera, Apple, The Guardian, The New York Times, Airbnb, SoundCloud, Duolingo, Google

- Atualmente, Nubank contratando desenvolvedores (https://nubank.workable.com/j/8DFFD4EA22)

- Problemas: Em 2015, o engenheiro chefe do Twitter à época disse estar arrependido, pela curva de aprendizagem (2 meses)

UFRPE
Universidade
Federal Rural
de Pernambuco

Agenda

- O que é a linguagem
- **Principais diferenças com Java**
- Instalando
- Implementação

Principais diferenças de Java

- Sintaxe: métodos com um argumento podem ser usados com sintaxe simplificada

```
df.format(now) → df format now
```

Principais diferenças de Java

- TUDO é um objeto
  - Inclusive números

```
1 + 2 → (1).+(2)
```

```
1 + 2 * 3 / x → (1).+(((2).*(3))./(x))
```

Principais diferenças de Java

- Funções também são tratadas como objetos (funções de alta ordem)

```
object Timer {
  def oncePerSecond(callback: () => Unit) {
    while (true) { callback(); Thread sleep 1000 }
  }
  def timeFlies() {
    println("time flies like an arrow...")
  }
  def main(args: Array[String]) {
    oncePerSecond(timeFlies)
  }
}
```

Principais diferenças de Java

- Classes

```
class Complex(real: Double, imaginary: Double) {
  def re() = real
  def im() = imaginary
}
```

  - Uso

```
object ComplexNumbers {
  def main(args: Array[String]) {
    val c = new Complex(1.2, 3.4)
    println("imaginary part: " + c.im())
  }
}
```

Principais diferenças de Java

- Métodos sem parâmetros: sem parênteses

```
class Complex(real: Double, imaginary: Double) {
  def re = real
  def im = imaginary
}
```

Principais diferenças com Java

- Constantes: val; Variáveis: var
- Tipos implícitos ou explícitos

```
val x = 1 + 1
x = 3 // This does not compile.
```

```
var x = 1 + 1
x = 3 // This compiles
println(x * x) // 9
```

```
var x: Int = 1 + 1
```

Principais diferenças para Java

- Outras questões:
  - Herança, Casamento de Padrões, Traits…

- Documentação disponível em: http://docs.scala-lang.org/tutorials/scala-for-java-programmers.html

UFRPE
Universidade
Federal Rural
de Pernambuco

Agenda

- O que é a linguagem
- Principais diferenças com Java
- **Instalando**
- Implementação

Instalação

- http://docs.scala-lang.org/getting-started.html

- Necessário:
  - IntelliJ Community Edition
  - Java 8

- “Hello World”

```
object Hello extends App {
  println("Hello, World!")
}
```

UFRPE
Universidade
Federal Rural
de Pernambuco

Agenda

- O que é a linguagem
- Principais diferenças com Java
- Instalando
- **Implementação**

UFRPE
Universidade
Federal Rural
de Pernambuco

Implementação

- Meta: fazer o sistema todo sem usar NENHUMA variável

- Recursivamente

Passo 1 - pseudocódigo

No nosso problema, como contar os erros recursivamente?

```
Funcao contaErros(linha){
        if (linha == ultima){
                return temErro(linha)
        } else {
                return temErro(linha) + contaErros(linha+1)
        }
}
```

Passo 2 - pseudocódigo

Processamento da diferença

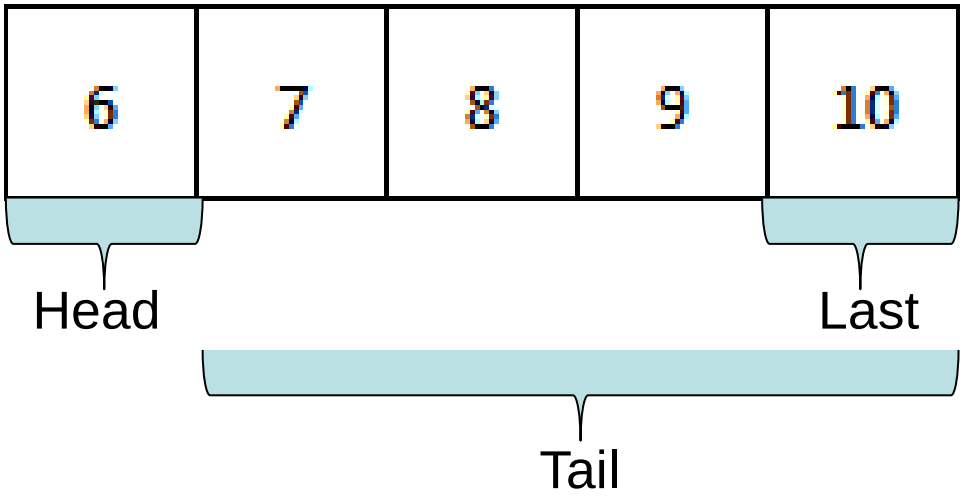
```
Funcao temErro(linha) {

        return comparaMaos(linha) != pegaResultado(linha)

}
```

Processamento das mãos

```
Funcao comparaMaos(linha){

    val mao1 = lerMao1; val mao2 = lerMao2;

    //Lógica da aplicação

}
```

Implementação

- Forte trabalho com listas:
  - lista.head (primeiro), lista.tail (todos menos o primeiro), lista.last(último)
  - lista.count, lista.max, lista.min



Implementação

- Sintaxe das funções

1 parâmetro

```
val addOne = (x: Int) => x + 1
println(addOne(1)) // 2
```

sem parâmetros

```
val getTheAnswer = () => 42
println(getTheAnswer()) // 42
```

2 parâmetros

```
val add = (x: Int, y: Int) => x + y
println(add(1, 2)) // 3
```

sem nome

```
(x: Int) => x + 1
```

- Fortemente baseado em recursão

Implementação

- Testa sequência
  - O que é uma sequência?

6	7	8	9	10

Implementação

- Testa sequência

6	7	8	9	10

```
/** Straight - Testa se as cartas estão em sequência */
def JogadaSequencia(cartas: List[String]): Boolean = {
  if (cartas.length == 2) {
    getValor(cartas.head) + 1 == getValor(cartas.last)
  } else {
    (getValor(cartas.head) + 1 == getValor(cartas.tail.head)) && JogadaSequencia(cartas.tail)
  }
}
```

Implementação

- Função List.forall

`override def` *forall*(*p* : (A) => Boolean) : Boolean

Tests if the predicate `p` is satisfied by all elements in this list.

Parameters

`p` - the test predicate.

Returns

`true` iff all elements of this list satisfy the predicate `p`.

List(1,2,3).forall(x => x < 3)

False

UFRPE
Universidade Federal Rural de Pernambuco

List(1,2,3).forall(x => x < 3)

6	7	8	9	10

Implementação

- Como testar o flush?

```
/** Flush - Testa se todos os naipes são iguais */
def JogadaFlush(cartas: List[String]): Boolean = {
  cartas.forall(c => getNaipe(c) == getNaipe(cartas.head))
}
```

override def *forall*(*p* : (A) => Boolean) : Boolean

Tests if the predicate p is satisfied by all elements in this list.

Parameters

p - the test predicate.

Returns

true iff all elements of this list satisfy the predicate p.

Implementação

- List.count

`def` *count*(*p* : (A) => Boolean) : Int

Count the number of elements in the list which satisfy a predicate.

Parameters

`p` - the predicate for which to count

Returns

the number of elements satisfying the predicate `p`.

List(1,2,3).count(x => x < 3)

UFRPE
Universidade Federal Rural de Pernambuco

List(1,2,3).count(x => x == 4)

6	7	8	8	10

Implementação

- Como testar o par?

```
/** One Pair */
def JogadaPar(cartas: List[String]): Boolean = {
  if(cartas.length == 2)
    cartas.count(c => getValor(c) == getValor(cartas.head)) == 2
  else
    cartas.count(c => getValor(c) == getValor(cartas.head)) == 2
      ||  JogadaPar(cartas.tail)
}
```

def *count*(*p* : (A) => Boolean) : Int

Count the number of elements in the list which satisfy a predicate.

Parameters

p - the predicate for which to count

Returns

the number of elements satisfying the predicate p.

UFRPE
Universidade
Federal Rural
de Pernambuco

Demais Jogos?

- Pequenas variações dos apresentados. A implementar no trabalho

UFRPE
Universidade
Federal Rural
de Pernambuco

Dúvidas?