Deficiências da linguagem Java e soluções atuais

Renato Pelizzari da Silva Ricardo Yamamoto Abe

4 de novembro de 2005

Histórico

- A plataforma e a linguagem Java começaram a ser projetadas em dezembro de 1990 na Sun Microsystems.
- A versão 1.0 foi lançada em 1996.
- Atualmente o Java encontra-se na versão 5.0 – codinome *Tiger*, originalmente 1.5.

Histórico

- A plataforma e a linguagem Java começaram a ser projetadas em dezembro de 1990 na Sun Microsystems.
- A versão 1.0 foi lançada em 1996.
- Atualmente o Java encontra-se na versão 5.0 – codinome *Tiger*, originalmente 1.5.

Histórico

- A plataforma e a linguagem Java começaram a ser projetadas em dezembro de 1990 na Sun Microsystems.
- A versão 1.0 foi lançada em 1996.
- Atualmente o Java encontra-se na versão 5.0 – codinome *Tiger*, originalmente 1.5.

O que compõe a plataforma Java

- Linguagem Java
- Java API
- Java Virtual Machine (JVM)

Linguagem Java

Java é uma linguagem que foi criada com os seguintes objetivos:

1. Ser orientada a objetos;
2. Os programas podem ser executados em várias plataformas;
3. Suporte intrínseco para utilização de redes de computadores;
4. Suporte à execução de código remotamente.

Linguagem Java

Java é uma linguagem que foi criada com os seguintes objetivos:

1. Ser orientada a objetos;
2. Os programas podem ser executados em várias plataformas;
3. Suporte intrínseco para utilização de redes de computadores;
4. Suporte à execução de código remotamente.

Linguagem Java

Java é uma linguagem que foi criada com os seguintes objetivos:

1. Ser orientada a objetos;
2. Os programas podem ser executados em várias plataformas;
3. Suporte intrínseco para utilização de redes de computadores;
4. Suporte à execução de código remotamente.

Linguagem Java

Java é uma linguagem que foi criada com os seguintes objetivos:

1. Ser orientada a objetos;
2. Os programas podem ser executados em várias plataformas;
3. Suporte intrínseco para utilização de redes de computadores;
4. Suporte à execução de código remotamente.

Java API

Conjunto de classes fornecidas pela Sun e as especificações de suas estruturas, funcionalidades e como utilizá-las.

Java Virtual Machine (JVM)

1. Código Java compilado (*bytecode*)
2. A JVM interpreta e executa o *bytecode*

Java Virtual Machine (JVM)

1. Código Java compilado (*bytecode*)
2. A JVM interpreta e executa o *bytecode*

Outras linguagens para plataforma Java

Existem mais de 200 linguagens diferentes que geram *bytecode* Java, podendo ser dividas em 2 conjuntos:

1. Utilizam API Java;
2. Alteram *bytecode*.

Exemplos

- Nice
- Groovy
- Jython
- JRuby
- Guaraná

Exemplos divertidos

- Pizza
- Yoyo
- Bolero
- Qexo
- Zigzag
- JAMES 007
- Misty Beach Forth

Questão filosófica

Com tantas linguagens que trabalham com a JVM, pode surgir o seguinte questionamento:

> "Java é a melhor linguagem para a plataforma Java?"
>
> *Steve Yegge*

Como responder à questão filosófica

Iremos discutir os seguintes pontos:

- Deficiências da linguagem Java 1.4;
- Aprimoramentos da linguagem Java 1.5;
- Outras linguagens.

Verborragia

Verborragia *S. f.* **1.** *Deprec.* Grande abundância de palavras, mas com poucas idéias, no falar ou discutir. **2.** Logorréia (2). [Sin. ger.: verborréia.]

Verborragia

Java definitivamente é uma linguagem verborrágica. Quem já programou em Java alguma vez na vida sabe o quão verborrágica ela pode ser. Veja nosso próximo exemplo.

Exemplo

```
try { BufferedReader file =
          new BufferedReader(
             new InputStreamReader(
                new DataInputStream(
                   new BufferedInputStream(
                      new FileInputStream("nomeDoArquivo");
                   )
                )
             )
          );
   String prog;
   if((prog = file.readLine()) != null)
      applet.loadProg(prog);
   file.close();
} catch(Exception e) {}
```

Proteção contra o mau programador

A Sun acabou removendo algumas funcionalidades e impondo regras, criando alguns empecilhos para os bons programadores: Exemplos:

- Tratamento de exceções
- Falta de macros
- Ausência de sobrecarga de operadores

Proteção contra o mau programador

A Sun acabou removendo algumas funcionalidades e impondo regras, criando alguns empecilhos para os bons programadores: Exemplos:

- Tratamento de exceções
- Falta de macros
- Ausência de sobrecarga de operadores

Proteção contra o mau programador

A Sun acabou removendo algumas funcionalidades e impondo regras, criando alguns empecilhos para os bons programadores: Exemplos:

- Tratamento de exceções
- Falta de macros
- Ausência de sobrecarga de operadores

Sobrecarga de operadores

Sobrecarga de operadores (*operator overloading*) é um caso de polimorfismo em que alguns operadores como $+$, - ou $==$ possuem diferentes implementações de acordo com os tipos de seus argumentos.

Exemplo

```
// vectors: overloading operators example
#include <iostream.h>

class CVector {
   public:
   int x,y;
   CVector () {};
   CVector (int,int);
   CVector operator + (CVector);
};

CVector::CVector (int a, int b) {
   x = a;
   y = b;
}
```

Exemplo – Continuação

```
CVector CVector::operator+ (CVector param) {
   CVector temp;
   temp.x = x + param.x;
   temp.y = y + param.y;
   return (temp);
}

int main () {
   CVector a (3,1);
   CVector b (1,2);
   CVector c;
   c = a + b;
   cout << c.x << "," << c.y;
   return 0;
}
```

Sobrecarga de métodos

Java exige que exista uma implementação de método para cada combinação possível de parâmetros. Além disso, não é possível criar parâmetros opcionais de forma elegante.

Herança múltipla

- Java só permite herança de uma única classe;
- Contornamos o problema com implementação de interfaces.

Conseqüência: não há reutilização de código.

Fechamentos

- Um fechamento é uma estrutura que armazena os parâmetros e o corpo de uma função.
- Essa estrutura pode ser utilizada como parâmetro ou resposta de um método, além de ser guardada em uma variável.

Fechamentos

Em Java, funções não são expressões de primeira ordem. Em linguagens como Scheme temos a função lambda.

Exemplo

```
(define fibonacci
   (lambda (n)
      (case n
         ((1 2) 1)
         (else (+ (fibonacci (- n 1))
                  (fibonacci (- n 2)))))))
```

Definição

Em C e C++, temos enumeradores que permitem a declaração de um conjunto de nomes como um grupo relacionado de valores. Exemplo:

```
enum Direction North,East,South,West;
```

Até o Java 1.4

Uma maneira de termos um enumerador em Java 1.4 é utilizando um `public final int`:

```
public interface Direction {
   public final int North = 0;
   public final int East = 1;
   public final int South = 2;
   public final int West = 3;
}
```

No Java 1.5

Java 1.5 permite a criação de enums similares aos de C e C++.
Por exemplo:

```
enum Season { WINTER, SPRING, SUMMER, FALL }
```

Exemplo mais complexo

```
public enum Planet {
   MERCURY (3.303e+23, 2.4397e6),
   VENUS   (4.869e+24, 6.0518e6),
   EARTH   (5.976e+24, 6.37814e6)

   private final double mass;   // in kilograms
   private final double radius; // in meters
   Planet(double mass, double radius) {
      this.mass = mass;
      this.radius = radius;
   }
   public double mass()   { return mass; }
   public double radius() { return radius; }

   // universal gravitational constant  (m^3 kg^{-1} s^{-2})
   public static final double G = 6.67300E-11;

   public double surfaceGravity() {
      return G * mass / (radius * radius);
   }
   public double surfaceWeight(double otherMass) {
      return otherMass * surfaceGravity();
   }
}
```

Continuação do exemplo

```
public static void main(String[] args) {
   double earthWeight = Double.parseDouble(args[0]);
   double mass = earthWeight/EARTH.surfaceGravity();

   for (Planet p : Planet.values())
         System.out.println("Your weight on %s is %f%n",
                            p, p.surfaceWeight(mass));
}

\$ java Planet 175
Your weight on MERCURY is 66.107583
Your weight on VENUS is 158.374842
Your weight on EARTH is 175.000000
```

Introdução

Nice é uma linguagem fortemente integrada ao ambiente Java, fornecendo diversas vantagens. Temos:

- Ela permite utilizar toda a API Java
- Bibliotecas escritas em Nice podem ser chamadas por um programa escrito em Java
- Classes parametrizadas
- Funções como parâmetros e respostas de métodos
- Multi-métodos
- Um método pode devolver uma tupla
- Chamadas de métodos com parâmetros opcionais
- Chamadas de métodos com parâmetros nomeados

Observação

Nice necessita de uma JVM versão 1.2 ou superior.

Classes parametrizadas

- Nice permite o uso de classes parametrizadas; elas são similares aos *templates* de C++.
- Uma classe parametrizada é simplesmente uma classe com um parâmetro associado. Nesse caso o parâmetro é um tipo ao invés de um valor.

Exemplo em Java

```
class Stack {
   List contents = new LinkedList();
   void push(Object o) {
      contents.add(o);
   }

   //... omitted methods

   public static void main(String[] args) {
      Stack st = new Stack();
      st.push("Test");
      Integer num = (Integer)st.pop(); // Runtime error
  }
}
```

Exemplo convertido em Nice

```
class Stack<T> {
   List<T> contents = new LinkedList();
   void push(T t) {
      contents.add(t);
   }

   //... omitted methods
}

void main(String[] args) {
   Stack<String> st = new Stack();
   st.push("Test");
   Integer num = st.pop(); // Compile time error!
}
```

Funções como parâmetros e respostas de métodos

Nice permite que funções sejam passadas como parâmetros e devolvidas como respostas de métodos.

Exemplo

```
//First define the method we're going to call:
//Like an 'if' statement
void when(boolean condition, void->void action)
{
   if (condition)
      action();
}

//Now exercise our method with block syntax:
void main(String[] args)
{
   when(1 > 0)
   {
      println("Math is working correctly!");
   }
}
```

Tupla

- Uma tupla é um grupo de vários valores em uma única expressão.
- Um método pode devolver mais de um valor por meio de uma tupla.

Exemplo

```
(int, int) minMax(int x, int y) = x < y ? (x, y) : (y, x);

void printTuple((int x, int y))
{
   println("(" + x + ", " + y + ")");
}

void main(String[] args)
{
   printTuple(minMax(14, 17));
   printTuple(minMax(42, 41));
}
```

Métodos com parâmetros nomeados

- É possível invocar um método especificando o nome do parâmetro, seguido de : antes do valor do parâmetro.
- Dessa forma, podemos invocar o método utilizando-os em qualquer ordem.

Exemplo

```
void copy(File from, File to) { ... }
...
copy(from: f1, to: f2);
copy(to: f3, from: f4);
```

Métodos com parâmetros nomeados

Note que ainda é possível omitir o nome dos parâmetros, nesse caso devemos seguir a ordem utilizada na declaração do método.

Métodos com parâmetros opcionais

Em Nice, é possível que um parâmetro tenha um valor padrão, de modo que esse parâmetro seja omitido da chamada do método.

Exemplo

```
void copy(File from, File to, int bufferSize = 1000) { ...
...
copy(from: f1, to: f2); // with a buffer size of 1000
copy(from: f1, to: f2, bufferSize: 4000);
```

Introdução

- Groovy é uma linguagem compatível com a plataforma Java 2.
- Ela possui características apreciadas em outras linguagens como:
  - Python
  - Ruby
  - Smalltalk

Introdução

Vamos nos concentrar em 2 elementos encontrados na linguagem:

- Fechamentos
- Sobrecarga de operadores

Fechamentos

Em Groovy, podemos armazenar uma função em uma variárvel.

Exemplo

```
{numberToSquare ->
   numberToSquare * numberToSquare
};

// initiate variables
int x, y;
x = 2;

// create closure and assign it to variable C
def c = {numberToSquare -> numberToSquare * numberToSquare }

// using C as the identifer for the closure, make a call on that closure
y = c.call(x);

// y will equal 4;
```

Sobrecarga de operadores

Vários operadores em Groovy estão mapeados sobre métodos Java já existentes. Isso permite que o programador sobrescreva tais métodos, tendo a sobrecarga de operadores.

Sobrecarga de operadores

As próximas tabelas mostram os operadores disponibilizados e os métodos neles mapeados.

Tabela de operadores

Operador	Método
$a + b$	a.plus(b)
$a - b$	a.minus(b)
$a * b$	a.multiply(b)
a/b	a.divide(b)
$a++$ ou $++a$	a.next()
$a--$ ou $--a$	a.previous()
$a[b]$	a.getAt(b)
$a[b] = c$	a.putAt(b,c)
$a << b$	a.leftShift(b)

Tabela de operadores

Operador	Método
$a == b$	a.equals(b)
$a! =$	! a.equals(b)
$a === b$	$a == b$ em Java
$a > b$	a.compareTo(b) > 0
$a >= b$	a.compareTo(b) $>= 0$
$a < b$	a.compareTo(b) < 0
$a <= b$	a.compareTo(b) $<= 0$

Referências


http://www.docrampage.net/tech/javacrit1.htm


http://opal.cabochon.com/ stevey/sokoban/


http://en.wikipedia.org/wiki/Java_programming_language


http://nice.sourceforge.net/


http://www-128.ibm.com/developerworks/library/j-alj10064.html


http://groovy.codehaus.org/


http://www.robert-tolksdorf.de/vmlanguages.html

E quando não restarem soluções... Cachaça Java!

E quando não restarem soluções... Cachaça Java!