Java - Introdução à Linguagem

Fabio Gagliardi Cozman

PMR2300
Escola Politécnica da Universidade de São Paulo

Histórico

1. Início com projeto da Sun para criar software para utilitários (1990);
2. Linguagem Oak, projetada por Gosling (1994);
3. Linguagem Java é criada para atuar na World Wide Web (1995);
4. Versões: 1.0.2, 1.1.x, 1.2 (Java2), 1.3, 1.4...

Java é

Simples, orientada a objetos, distribuída, interpretada, segura, neutra a arquiteturas, portável, de alta performance, dá suporte a múltiplas linhas de processamento, dinâmica. (conforme propaganda da Sun, no Java Whitepaper, www.javasoft.com)

Explicando as Palavras

1. Neutra a arquiteturas, portável -> Interpretada -> De alta performance
2. Segura
3. Simples
4. Orientada a objetos
5. Robusta

Neutra a Arquiteturas

1. Source (.java) -> Compilador (javac)
2. Bytecode (.class) -> Interpretador (java)
   Java é interpretada, mas de um jeito especial

Java é interpretada

1. Código fonte é compilado (transformado em bytecodes) e então interpretado pela JVM
2. Em linhas gerais:
   1. Linguagens compiladas: código fonte é transformado em código de máquina e armazenado para execução
   2. Linguagens interpretadas: cada instrução é transformada em código de máquina e executada imediatamente e individualmente

Java é Realmente Portável

1. Tipos são padronizados: byte tem 8 bits, short tem 16 bits, int tem 32 bits, long tem 64 bits
2. Operações matemáticas são padronizadas float, double tem representação IEEE754 NaN funciona (!)
3. Caracteres são padronizados char é caracter Unicode com 16 bits

Java é Segura

1. Nenhum acesso direto à memória (ponteiros)
2. Controle sobre operações de sistema
3. Verificação de bytecodes (conceito de sandbox)

Java é Simples

1. Não há controle de memória pelo programador
2. Não há: pointer, union, enum, typedef
3. Não há prototypes
4. Não há pre-processador
5. Não há sobrecarga de operadores, herança múltipla, templates (para C++)

Java é Simples e Familiar

```
public class Echo {
    public static void main (String[] args) {
        for (int i = 0; i < args.length; i++)
            System.out.println(args[i]);
        }
}
```

Java é Robusta

1. Sistema de tipos é rígido
2. Verificação de tipos em tempo de execução:
   LinkedList a;
   b = (HashTable)a;

Java é Realmente Robusta

1. Verificação de vetores em tempo de execução
2. Verificação de null-pointer em tempo de execução
3. Manipulação de exceções é parte da linguagem
4. Inicialização automática para qualquer variável

Em Resumo, Java...

1. Simples, Robusta, Orientada a Objetos
2. Portável
3. Segura
4. e tem uma variedade de libraries: interfaces gráficas (AWT, Swing) suporte para linhas de processamento, rede, imagens, sons vetores, stacks, hashtables, etc

Ferramentas

1. JDK:
   1. appletviewer, jar, java, javac, javadoc, jdb javah, javakey, javap, native2ascii
2. Livre
   1. NetBeans
   2. Eclipse

Informações

1. Documentação e download em site da Oracle
2. Concentre-se em Java Standard Edition (SE), versão JDK 7 (não use beta!)
3. Tutorial (em Learning): http://docs.oracle.com/javase/tutorial/index.html

Linguagem

```
/* Classe Echo para teste. */
public class Echo {
    // Entrada: main
    public static void main (String[] args) {
        for (int i = 0; i < args.length; i++)
            System.out.println(args[i]);
            System.exit(0);
    }
}
```

Tipos de Dados Primitivos

- boolean (valores true ou false).
- char (caracteres em Unicode, usando 2 bytes).
- byte, short, int, long (inteiros usando respectivamente 1, 2, 4 e 8 bytes).
- float, double (reais em formato IEEE usando respectivamente 4 e 8 bytes).

Tipos de Dados Primitivos

Uma variável cujo tipo é um tipo primitivo está armazenada no Stack do programa em execução.
Além desses tipos primitivos existem os tipos criados por recursos de orientação a objetos. Um desses tipos é especial: o tipo String permite o uso do operador "+" para concatenação
Ou seja,

"oi" + "pessoal" resulta na String "oi pessoal".

Fluxo de Controle

1. if, else
2. while
3. do, while
4. switch
5. for
6. break
7. continue

Operadores

Aritméticos

1. Soma +
2. Subtração -
3. Multiplicação *
4. Divisão /
5. Resto da Divisão

Operadores

Atribuição

1. ’=’
2. ’+=’ : op1 += op2 equivale a op1 = op1 + op2
3. ’-=’ : op1 -= op2 equivale a op1 = op1 - op2
4. ’*=’ : op1 *= op2 equivale a op1 = op1 * op2
5. ’/=’ : op1 /= op2 equivale a op1 = op1 / op2
6. ’%’=’ : op1 %= op2 equivale a op1 = op1 % op2

Operadores

Comparação

1. '>': Maior que
2. '<': Menor que
3. '==': Iguais
4. '!=': Diferentes
5. '>=': Maior ou igual que
6. '<=': Menor ou igual que

Operadores

Lógicos

1. '&&' : devolve true se ambos operandos forem true.
2. '||' : devolve true se algum dos operandos for true
3. '!' : Nega o operando que se passa.
4. '&' : devolve true se ambos operandos forem true, avaliando ambos.
5. '|' : devolve true se um dos operandos for true, avaliando ambos.

Arranjos

1. Unidimensionais: int[2], Button[10], Object[122]
2. Multidimensionais: int[3][44][3]
3. Criados com new: int[] a = new int[2];
4. Não precisam ser removidos (“coleta de lixo”)

Declaração e Alocação

Todo arranjo deve ser declarado e então alocado: Considere a linha de código:
int[] a= new int [10];
Temos:

- *int[]* $\rightarrow$ tipo;
- *a* $\rightarrow$ nome;
- *new* $\rightarrow$ necessário para reservar memória;
- *[10]* $\rightarrow$ tamanho.

Uma sintaxe alternativa, mais próxima da sintaxe da linguagem C, é:
int a[]= new int [10];

Exemplos: Declaração e Alocação

```
int b[]; // declara.
b= new int[20]; // aloca
double c[][]= new double [2][2]; // declara e aloca
double d[][]= new double [2][]; // declara e aloca
c[0]= new double[20];  aloca
c[1]= new double[30];   aloca
```

Referências

As variáveis que denotam arranjos são *referências*; estas variáveis apenas guardam endereços de memória. Portanto, se fizermos:

```
int x[] = new int[10];
```

temos que `x` guarda o endereço de um arranjo com 10 elementos.

length

Todo arranjo `a` é automaticamente associado a uma variável `a.length` que retorna o tamanho do arranjo. O primeiro elemento de um arranjo `a` tem índice 0 e o último elemento de `a` tem índice `a.length`.

Exemplos: length

```
a[i]=138;
a[i][i+1]=2.3 + x[i][i];
i=a.length;
j=x[4].length;
s=new string[5];
y=s[(s.length - 1)];
```

Exemplo 1

O seguinte exemplo inicializa um arranjo bidimensional:

```
int t[][] = new int[10][];
for (int i=0; i<t.length; i++) {
        t[i]=new int[i+1];
        for (int j=0; j<(i+1); j++)
                t[i][j]=i+j;
}
```

Exemplo 2

O seguinte exemplo copia o conteúdo de um arranjo em outro:

```
int a[] = new int[b.length];
for (int i=0; i<b.length; i++)
        a[i]=b[i];
```

Orientação a Objetos

Programas são organizados em torno de classes e objetos

```
public class Circle {
    public double x,y,r;
    public double area() {
        return( Math.PI * r * r );
    }
}
```

Declaração e Criação de Objetos

1. Declare: Circle c;
2. Crie: c = new Circle();
3. Declare, crie e inicialize: Circle c = new Circle(1,2,3);

Métodos e Construtores

1. Chamando um método: c.area();
2. Construtor simples: public Circle ()
3. Outro construtor (sobrecarga):
   public Circle(double x, double y, double r)

Polimorfismo (sobrecarga)

- Dois métodos tem nomes diferentes se tiverem parâmetros de tipos diferentes.
- Exemplo: read(int), read(int, int), read(double), read(char, int, int).
- Métodos com valor de retorno diferente não podem receber o mesmo identificador.

Variáveis de Classe

Variável associada à classe (não a objetos)

```
public class Circle {
    static int num_circles;
    ...
    a = Circle.num_circles;
    ...
}
```

Exemplo: Math.PI

Métodos de Classe

1. Método associado à classe
2. Exemplos importantes: Math.sqrt, Math.max, Math.min, Math.cos, etc
3. Método declarado como static: public static void main(String args[]) public static double sqrt(double v)

Hierarquias e Heranças

1. Classes podem ser estendidas
2. Objetos herdam variáveis e métodos
3. Superclasse é referida por super
4. Métodos são superpostos, variáveis não (chamada do método é associada a implementação durante execução!)

Coleta de lixo

1. Java não tem recursos na linguagem para liberar memória
2. Memória é liberada automaticamente através de um processo denominado “coleta de lixo” (garbage collection)
3. Quando coletados, objetos podem ser finalizados pelo método finalize()

Além disso

1. Packages: grupos de classes (package)
2. Import: inclusão de classes
3. Importante: através das regras de uso de packages e imports, Java mantem uma estrutura de nomes que impede colisões

Acesso a Objetos

1. Possíveis situações:
   1. public - todas as classes
   2. (package) - só no pacote
   3. protected - só no pacote e subclasses
   4. private - só na classe
2. Dica: construa packages pequenos para garantir encapsulamento de dados

Copiando Objetos

Para copiar referências:

```
Object a, b;
a = b;
```

Um exemplo:

```
public void swap(Object a, Object b) {
    Object temp = a;
    a = b;
    b = temp;
}
```

Pacotes

1. Packages: grupos de classes (package)
2. Inclusão de classes e pacotes (import)
3. Através do uso de pacotes, Java mantém uma estrutura de nomes que evita colisões.

Pacotes Java

1. Java 1.0:
   1. lang (automático!)
   2. io, applet, awt (image, peer), net, util
2. Java > 1.0:
   1. beans, math, security, text, swing, sql...

Java e OO: Resumo

1. Variáveis: primitivas e de referência; Strings; locais; final.
2. Objetos: this; super; new; finalize; coleta de lixo
3. Classes: public; extends; final
4. Arrays: new; coleta de lixo
5. Métodos: this; super; construtores; sobrecarga; superposição; public, private, protected; static; final

Classes Abstratas

1. Objetivo: implementar alguma funcionalidade, mas deixar funcionalidade para sub-classes
2. Classes abstratas não geram objetos

Interfaces

1. Uma “classe abstrata” que só tem métodos abstratos e constantes (static final) para sub-classes
2. Define a “interface” entre classes (!)
3. Uma das mais poderosas ferramentas para especificação de software, e um substituto para herança múltipla

Manipulação de Exceções

Novos termos:

1. try - uma região de código
2. catch - uma condição de exceção
3. finally - uma região de código

Além disso, um método pode throw/throws exceções

Exceções são Objetos

1. Classes de exceções podem ser definidas e estendidas
2. java.lang Throwable java.lang.Error java.lang.Exception
3. Em geral, nunca use catch para tratar um objeto Error

Exceções devem ser Tratadas

1. Tratamento:
   1. Uso de try e catch
   2. Uso de throws
2. Java exige tratamento de toda exceção que não estenda Error ou Exception.RuntimeException

Blocos try-finally

1. É possível usar try e finally sem tratar exceções
2. Objetivo: garantir finalização de procedimentos (fechar arquivos, conexões, etc), mesmo com break, continue ou return
3. Uso de throws

Pacotes Importantes

1. java.beans
2. java.io
3. java.lang
   1. ref
   2. reflect
4. java.math
5. java.net
6. java.security
   1. acl
   2. cert
   3. interfaces
   4. spec

Pacotes Importantes

1. java.text
2. java.util
   1. jar
   2. zip
3. java.awt

Além disso temos SWING (interface gráfica), criptografia, conectividade, bases de dados, etc.

String

String S = “exemplo” + “!”;
Método toString() em todo objeto → usado para imprimir.

1. s.length();
2. s.substring(4); // de 4 ao final
3. s.substring(4,6); // caracteres 4 e 5
4. s.charAt(3);
5. s.toCharArray();
6. s.toUpperCase();
7. s.toLowerCase();

String

1. s.startsWith(“ab”);
2. s.endsWith(“ab”);
3. s.compareTo(“ab”);
   1. < 0 se $s <$ “ab”
   2. $= 0$ se $s =$ “ab”
   3. > 0 se $s >$ “ab”
4. s.compareToIgnoreCase();
5. s.indexOf(“ab”);

Ainda sobre Strings

1. Método toString() aplicado a todo objeto: String s = o.toString();
2. Classe StringBuffer: – capacity, length, charAt, append, insert, reverse
3. Classe StringTokenizer (muito útil!)
4. Escapes: – \b, \t, \n, \",\xxx...

Classes “Empacotadoras”

1. Boolean Byte, Short, Integer, Long, Float, Double, Character
2. Métodos: X.parseX(String s)
3. Character.isLetter, Character.toUpperCase.
4. Integer.toBinaryString(int x), Integer.toOctalString(int x), Integer.MAX_VALUE, Integer.MIN_VALUE.

Math, System

1. Math.cos/sin/tan/acos/asin/atan/exp/log/pow/ceil/floor/round/toDegrees.
2. Note: java.math contém BigInteger, BigDecimal.
3. System: exit, currentTimeMillis, arraycopy
4. System:
    1. PrintStream out, err
    2. InputStream in

java.io

1. Pacote para Entrada/Saída de dados (arquivos, teclado, monitor)
2. Classe File: permite manipular arquivos e diretórios (tamanhos, datas, permissões).
3. InputStream e OutputStream: ler e escrever sequências de bytes.
4. Reader e Writer: ler e escrever sequências de caracteres.

Ler do teclado

Variável System.in:

```
import java.io.*;
BufferedReader br =
    new BufferedReader(new
        InputStreamReader(System.in));
String n;
try {
    n = br.readLine();
} catch(IOException e) { }
```

Ler de arquivo

```
import java.io.*;
try {
    BufferedReader br =
     new BufferedReader(new FileReader(''h.txt''));
    String line = br.readLine();
    while (line != null) {
        System.out.println(line);
        line = br.readLine();
    }
    br.close();
} catch (IOException e) { }
```

Escrever em arquivo

Classe PrintWriter:

```
import java.io.*;
try {
    File f = new File(''dir'', ''hey.txt'');
    PrintWriter pw =
       new PrintWriter(new FileWriter(f));
   pw.println(''HEY! File with '' + f.length());
   pw.close();
} catch (IOException e) { }
```

Outros Métodos

Existem muitos outros métodos para:

1. ler de arquivos binários
2. ler de strings
3. ler da rede
4. ler dados comprimidos
5. ler dados criptografados
6. escrever nas mesmas condições

java.net

```
import java.net.*;
import java.io.*;
public class Download {
    public static void main(String args[]) {
        String linha;
        if (args.length>0) {
        try {
            URL u = new URL(args[0]);
            DataInputStream html =
                new DataInputStream(u.openStream());
            while ((linha = html.readLine()) != null)
          System.out.println(linha);
          html.close();
      } catch (Exception e) { System.err.println(e); }
    }
}
```

Collections

Até Java 1.1, tinhamos na plataforma Java:

- Vector: sequência de itens com tamanho variável, implementada com arranjo;
- Stack: implementação de Pilha;
- Hashtable: implementação de Hashtable.

Após Java 1.1 foram introduzidas várias classes: