Introdução à linguagem JAVA

Leônidas de Oliveira Brandão *

21 de Fevereiro de 2003

Conteúdo

*Departamento de Ciência da Computação - IME - USP [Primeira versão: 7 de julho de 1997]

1 Estrutura da Linguagem Java

Nesta seção introduziremos os conceitos básicos da linguagem Java, adiando para os conceitos mais "sofisticados" como `threads` e `Applets` para a próxima seção.

Os exemplos que utilizaremos foram preparados a partir do JDK (Java Development Kit) da Sun Software.

Para exemplificarmos os conceitos envolvidos básicos de Java, trabalharemos principalmente com uma classe que denominaremos `Individuo`, e que descreveremos ao longo da seção, explorando construtivamente, os conceitos envolvidos. Ao final apresentaremos outros dois exemplos, específicos sobre manipulação de palavras ("strings").

1.1 Tipos de dados e comandos básicos

Inicialmente apresentaremos um programa Java, sem maiores comentários, seguindo com os tipos e comandos básicos de Java. Um programa Java de nome `X.java` deve conter sempre uma classe não privada como o mesmo nome, isto é `public class X ...{` (ou simplesmente `class X ...{`).

Exemplo Estrutura básica de programa Java

```
class exemplo1 {
 // ... variáveis da classe  <exemplo1>
 public static void main( String args [] ) { // Obrigatório em qualquer progr. nao graf.
          // variáveis do método  <main>
              A a = new A(); // var. <a> e uma instância da classe <A>
              B b = new B(); // <A()> e <B()> sao CONSTRUTORES das classes
           System.out.print("Início \n"); // caractere '\n': muda de linha
           a.imprime(); // invoca o método <imprime> da classe <A>
           b.imprime(); // invoca o método <imprime> da classe <B>
           System.out.println("Final");
           }
} // fim classe <exemplo1>

class A { // classe <A> contém: uma var. inteira + um método de impressão
 int a;
 void imprime(){ // método <imprime> de <A>
        a=13;
        System.out.print("A: a="+a);
        }
} // fim classe <A>
class B { // classe <B> contém: uma var. inteira + um método de impressão
 int a;
 void imprime(){ // método <imprime> de <B>
        a=14;
        System.out.println("B: a="+a); // <println> muda de linha ao final
        }
} //  fim classe <B>
```

Para compilar este programa, teríamos uma linha de comando como:

`milanesa:~/java/> /usr/local/java/bin/javac exemplo1.java`.

E para rodá-lo, algo como:

`milanesa:~/java/> /usr/local/java/bin/java exemplo1`.

Nesta seção nos deteremos no modo de programação não gráfica (fora dos ambientes de janelas). Neste modo, um programa Java, deve obrigatóriamente conter uma classe com o mesmo nome do arquivo `.java`.

Além disto, também é obrigatório a existência de um método (função) de nome `main`, obrigatoriamente `static` e `public` (conforme declarado no exemplo acima), por este método será iniciado a execução do programa (análogo ao `main` de C).

Os tipos básicos em JAVA são: `byte`, `short`, `int`, `long`, `float`, `double` e `char`.

Tipos	**Número de bytes**	**Tipos**	**Número de bytes**
`byte`	8	`short`	16
`int`	32	`long`	64
`float`	32	`double`	64
`char`	16	`boolean`	`true, false`

Em JAVA, expressão lógica resulta `true` ou `false` (como PASCAL) e `;` é finalizador de comando (como C). Vetores serão discutidos na sub-seção 1.8.

Variáveis podem ser declaradas em qualquer ponto, nas classes ou nos métodos. As variáveis declaradas nos métodos, só podem ser acessadas pelos mesmos (dentro de seu bloco de definição). Os blocos são delimitados por { e por }.

Os comandos básico de JAVA são:

`if`	`if` (*cond_log*) *comando* [`else` *comando*] ;
`for`	`for` (*inicia*; *cond_log* ; *incremento*) *comando*;
`while`	`while` (*cond_log*) *comando* ;
`break`	`break` [`rotulo`];

onde, *cond_log* ∈{ `true, false` } e *inicia*, *comando* e *incremento*, são comandos JAVA quaisquer. O comando `for` pode ter "vazios", inclusive em *cond_log* que neste caso seria assumida como `true`.

O comando `break` tem uma funcionalidade igual ao seu correspondente em C, como os demais comandos básico da JAVA que são análogos ao da linguagem C, mas também é um comando que cumpre parte da tarefa do "desestruturador de código" *goto*, que em JAVA não existe. Vide exemplo abaixo.

Exemplo Exemplos de `for` e `break`, a ser gravado com nome `for_break.java`

```
class testa_break {
  public static void main( String [] in ) {
    int i=0,j=0;

    r1: { boolean b=true;
    r2:   {
    r3:     {
            System.out.println("Antes do <break>");
            if ( b ) break r2; // vai para final do bloco <r2>
            System.out.println("Note: isto não é executado!");
            } // final bloco <r3>
          System.out.println("Isto também não é executado!");
          } // final do bloco <r2>
        System.out.println("Depois do rótulo <r2>");
        } // final do bloco <r1>

    a: for ( ; ; )
           for (  ;  ; System.out.println("i="+i+"  j="+j) )
               { i++; j++; if (j==10) break a; } // interrompe quando ''j=10''
  }
}
```

1.2 Classe = Métodos + Variáveis

Um progama em Java é basicamente uma coleção de classes, sendo que, uma *classe* é composta por variáveis e por métodos. Estas classes são os *objetos*, as variáveis podem ser vistas como as "características" do objeto e os métodos como as ações definidas sobre o objeto.

A classe `Individuo` possui como característica principal o `peso` do `Individuo` e a quantidade destes `num_Individuos_criados`.

```
class Individuo {
  static int num_Individuos_criados; // a ser compartilhado por todas as instâncias
  private int peso; // cada instância pode ter seu peso

  public Individuo ( int peso ) {
    num_Individuos_criados++; // mais um Individuo criado
    this.peso = peso;
    }

  public int peso() {
    return peso; // retorne o peso da instância
    }
}
```

1.3 Componentes Estáticos e Não Estáticos

As variáveis e os métodos de uma classe podem ser *estáticos* (`static`) ou não estáticos. Um componente da classe deve ser declarado `static` quando for uma característica intrinseca à classe como um todo e não a cada instância desta classe.

Em nosso exemplo, as variáveis `num_Individuos_criados` e `peso` apresentam uma diferença essencial, a primeira é característica da classe enquanto a segunda é própria de cada elemento da classe. Assim, destas duas, apenas `num_Individuos_criados` deve ser declarada `static`. O mesmo se aplica aos métodos, veja exemplo acima e seu complemento na subseção 1.5.

1.4 Instâncias

Criar instâncias de objetos é muito parecido com a "criação" de variáveis dinâmicas em C como o `malloc`. Entretanto, ao contrário de linguagens tipo C, em Java é obrigatório *instanciar* uma classe para de fato poder usá-la: não instância-la praticamente equivale em C a declarar um apontador e usá-lo antes atribuir algum endereço (`int *p; printf("%d",*p);`).

Usando a classe exemplo, poderíamos criar muitos "representantes" da classe `Individuo` durante a execução do programa Java, cada qual com seu próprio `peso`, para isso é necessário o comando `new`: `Individuo ind1 = new Individuo(10); Individuo ind2 = new Individuo(25)`. Cada um destes elementos (objetos) será uma *instância* da classe `Individuo`, contendo cópias separadas de cada variável (atributo) e compartilhando as mesmas variáveis estáticas.

1.5 Construtores

Outra característica importante relativa às classes, são os `construtores`. São métodos especiais, com o mesmo nome da classe, utilizados para iniciar um nova instância. Se o programador não providenciar um construtor para determinada classe, é automaticamente assumido a existência de um construtor vazio para esta classe.

Genericamente, um elemento é declarado ser do tipo `nome_Classe` fazendo-se

```
nome_Classe identificador;
```

e uma nova instância da classe é criada através do comando `new`

```
identificador = new nome_Classe([lista de parâmetros, separados por vírgulas]);
```

é a partir deste ponto que será alocado espaço para as variáveis da instância `identificador` da classe `nome_Classe`.

No exemplo da `class Individuo`, o construtor é o método `public Individuo ( int peso )`, no qual notamos a presença do comando `num_Individuos_criados++;`, significando que a cada instância de `Individuo` que é criado incrementa-se o total existente de "membros" da classe.

Exemplo Instâncias e construtor da classe `Individuo`

```
class testaIndividuo {

 public static void main (String args[]) { // método estático

  Individuo ind1, ind2, ind3 = new Individuo(10); // ind3 é uma instância da classe Individuo

  Individuo.num_Individuos_criados = 0; // como é estática, pode ocorrer ANTES de qq instancição

  ind1 = new Individuo(60);  // agora, ind1 e ind2, também passam a ser instâncias da classe Individuo,
  ind2 = new Individuo(25);  // cada qual com seus atributos (no caso apenas o peso)

  System.out.println("Número de \"Individuo\"'s criados: " + Individuo.num_Individuos_criados +
                     " = " + ind1.num_Individuos_criados + " = " + ind2.num_Individuos_criados +
                     " = " + ind3.num_Individuos_criados);
  System.out.println("\"Individuo\" 1 tem peso: " + ind1.peso());
  System.out.println("\"Individuo\" 2 tem peso: " + ind2.peso());
  System.out.println("\"Individuo\" 3 tem peso: " + ind3.peso());
  }
}
```

Após definirmos os tipos e comandos básicos de Java, vamos apresentar por completo este exemplo na sub-seção 1.8.

1.6 Como invocar variáveis e métodos

Existe dois modos para se invocar componente, variáveis ou métodos, de uma classe, usando o nome da instância ou o nome da classe, dependo de ser o mesmo estático ou não estático.

Estático `nome_da_class.nome_objeto`

Não Estático `nome_da_instância.nome_objeto`

onde `nome_objeto` é a variável ou método almejado.

Exemplo Invocando variáveis `(a)` não estáticas e `(b)` estáticas

```
...
class Exemplo1 {
      ...
      static public void main( String [] args ){
             ...
             a=new Individuo();
             a.peso = 100;          // (a) define peso do Individuo <a>
             // (b) referência <ESTÁTICA>
                System.out.println("Número de Individuo's criados = "+
                                   Individuo.num_Individuos_criados);
             // (a) referência <NÃO ESTÁTICA>
                System.out.println("Um dos Individuo's: "+ a.peso);
             ...
             }
      ...
      }
```

1.7 Tipos de acessos

Estas variáveis e métodos tem diferentes tipos de acessos, com isto queremos dizer que, um determinado método de uma classe pode ou não conseguir invocar um método ou fazer uso de uma variável de outra classe.

1.8 Vetores

Apesar de Java "parecer" ser uma linguagem "meramente interpretada", tem uma fase inicial onde o programa deve ser compilado. Assim, não é possível definir um vetor com dimensão fornecida por uma variável, a menos que esta seja estática (vide `limite` no exemplo a seguir).

A sintaxe para se definir um vetor é

1	`Objeto [] nome_variável;`	define o identificador como vetor
2	`Objeto nome_variável = { V_objeto`$_1$`, ..., V_objeto`$_N$` };`	aloca o espaço e "inicializa"
3	`a = new Objeto[ limite_estático ];`	aloca o espaço

Na linha 2 acima, `Objeto` deve ser um tipo básico, como `int`, `float` ou `String`.

A seguir apresentamos um exemplo completo de programa Java, ilustrando variáveis estáticas e não estáticas, construtores, os comando de repetição `while` e `for` e o uso de um vetor de objetos.

Exemplo Programa `Exemplo1.java`

```
class Individuo{
  static int num_Individuos_criados=0;   // comum a TODAS instâncias da classe
      int peso;                  // variável inteira, uma para cada instância
      Individuo(){             // método que retorna NADA e não tem parâmetros
           num_Individuos_criados++;              // incrementa de uma unidade
           }
      }

class Exemplo1 {
      static int limite = 10;           // Não pode estar dentro de <main>
      static public void main( String [] args ){
             Individuo [] a=new Individuo[limite]; // apenas dimensiona !!
             System.out.println("    Exemplo 1");
             int i=0;
             while ( i<limite ) {
                   a[i]=new Individuo();            // cria novo Individuo
                   a[i].peso = 100 + i;        // define peso do Individuo
                   i++;
                   }
             System.out.println("Número de Individuo's criados = "+
                                 Individuo.num_Individuos_criados);
             for (i=0; i<limite; i++)
                 System.out.println(i+"-ésimo Individuo tem peso = "+
                 a[i].peso);
             }
}
```

1.9 Classe `System` e `Strings`

Java providencia uma série de métodos para manusear "strings", nesta seção veremos os mais básicos. Também discutiremos um pouco sobre a classe `System` que providencia métodos para entrada e saída de dados.

Uma palavra, `pal`, em Java é um `String` ou um `StringBuffer`. Assim, `pal` será uma instância do tipo, ou uma referência à este tipo. A diferença básica entre ambos é que o primeiro tipo produz uma referência *estática*, isto é, uma vez definido o valor da instância, quanto ao "espaço" alocado, este não mais poderá ser alterado. Se fizermos

```
pal = "01234"; // (1)
pal = "6789";  // (2)
```

a segunda atribuição implica alocação de novo espaço na memória para a segunda palavra `(2)` e perda da primeira palavra, de `(1)`, que será disponibilizada para uma posterior coleta de lixo automática do sistema de execução Java (seja ele o `java` do JDK-Sun ou o sistema Java do Netscape.

Exemplo Programa (`tipos.java`) com os tipos básicos de Java e alguns métodos prévios.

```
import java.io.*;
class tipos {
 // Se nao definir a variável <limite> como <static>, teremos o erro
 // "Can't make a static reference to nonstatic variable limite..." em (*)
 static int limite=10;
  public static void main( String args [] ) { //throws IOException{
          int i;
          todos var=new todos();
          /* <String>  e <StringBuffer> sao classes (ja definidas) com alguns
                                       métodos associados, como <valueOf>,
             <+>, <append>, <capacity> e <setLength>
             (os 2 primeiros para <String>)                           */
          String s1 = new String(); // ou apenas: String s1="";
          StringBuffer s2 = new StringBuffer();

          System.out.println("Digite seu nome ");
          byte [] aux=new byte[10];
          try { System.in.read(aux); } // Obrigatório este comando,
          catch(IOException e) {}      // veremos depois!

          String nome = new String(aux,0); // copia a partir da posição 0
          System.out.print("Lá vamos nós "+nome);
          var.imprime(); // notar o valor "default" dos tipos
          for (i=0; i<limite; i++) { // (*)
               s1 = s1.concat( String.valueOf( i ));
               s2 = s2.append(i);
               }
          System.out.println("s1="+s1+" s2="+s2);
          var.inicia();
          var.imprime(); //System.out.print("\n");
          }
}  // fim <main>

class todos {
  int i;     boolean boole;
  long l;   String s1; // String: uma vez definida nao pode ser alterada
  float f;  StringBuffer s2;
  double d; char c;
  void imprime() {
         System.out.print("i="+i+" l="+l+" f="+f+" d="+d+" boole="+boole+
                                 " s1="+s1+" s2="+s2+" c="+c+"\n");
         }
  void inicia() {
         i=123;
         l=123;
        s1="123"; // equivale a: s1 = new String("123");
        s2=new StringBuffer();  // s2.capacity() = capacidade maxima
        f=123;
        s2=s2.append(1234); s2.setLength(3); // apaga ultimo digito!
        d=123;
        c='a';
        } // fim <inicia>
} // fim <class todos>
```

O exemplo acima apresenta os diversos tipos de Java, além de alguns métodos básicos para manipulação de palavras, para entrada e para saída de dados.

A classe `System` contém as variáveis públicas `in` e `out` e os métodos `read` para a primeira e `print` e `println` para a segunda. O método `read` têm o formato

`read()`	retorna próximo `byte` disponível de entrada
`read( byte b[], int inicio, int tam)`	tenta lê `tam` `bytes` para `b` iniciando na posição `inicio` e retornando a quantidade de `bytes` efetivamente lida

O método `print(ln)` é um método com muita "sobrecarga", isto é, aceita todo tipo de parâmetro, dentre às variáveis básicas, convertendo-os para o formato de saída. O operador `+` é o único como *sobrecarga* em Java, define a soma quanto aplicado a valores numéricos e define concatenação quando aplicado à `Strings`.

`String s = "Ele tem "+ idade +" anos;"`

equivale à forma não amigável

`String s = new StringBuffer("Ele tem ").append(idade).append(" anos").toString();`

Observação O equivalente ao `read` para saída de dados é o `write` com a mesma sintaxe indicada acima para o `read`.

Apresentamos a seguir mais um exemplo de `Strings`.

Exemplo Programa `maisStrings.java`

```
class maisStrings {
      static byte a[] = {65,66,67, 68};
      static byte b[] = {97,98,99,100};
      static byte c[] = {48,49,50, 51};
      public static void main( String [] args ){
             System.out.println("a = "+new String(a,0));
             System.out.println("b = "+new String(b,0));
             System.out.println("c = "+new String(c,0));
             System.arraycopy(a,0,b,0,a.length);
             System.out.println("a = "+new String(a,0));
             System.out.println("b = "+new String(b,0));
             }
      }
```

Saída Como fica o saída em um sistema UNIX

```
milanesa:~/java/> /usr/local/java/bin/javac maisStrings.java
milanesa:~/java/> /usr/local/java/bin/java maisStrings
a = ABCD
b = abcd
c = 0123
a = ABCD
b = ABCD
milanesa:~/java/>
```