Noções básicas da linguagem Java

Profs. Marcel Hugo e
Jomi Fred Hübner

Departamento de Sistemas e Computação
Universidade Regional de Blumenau - FURB

Introdução

Origem

Funcionamento

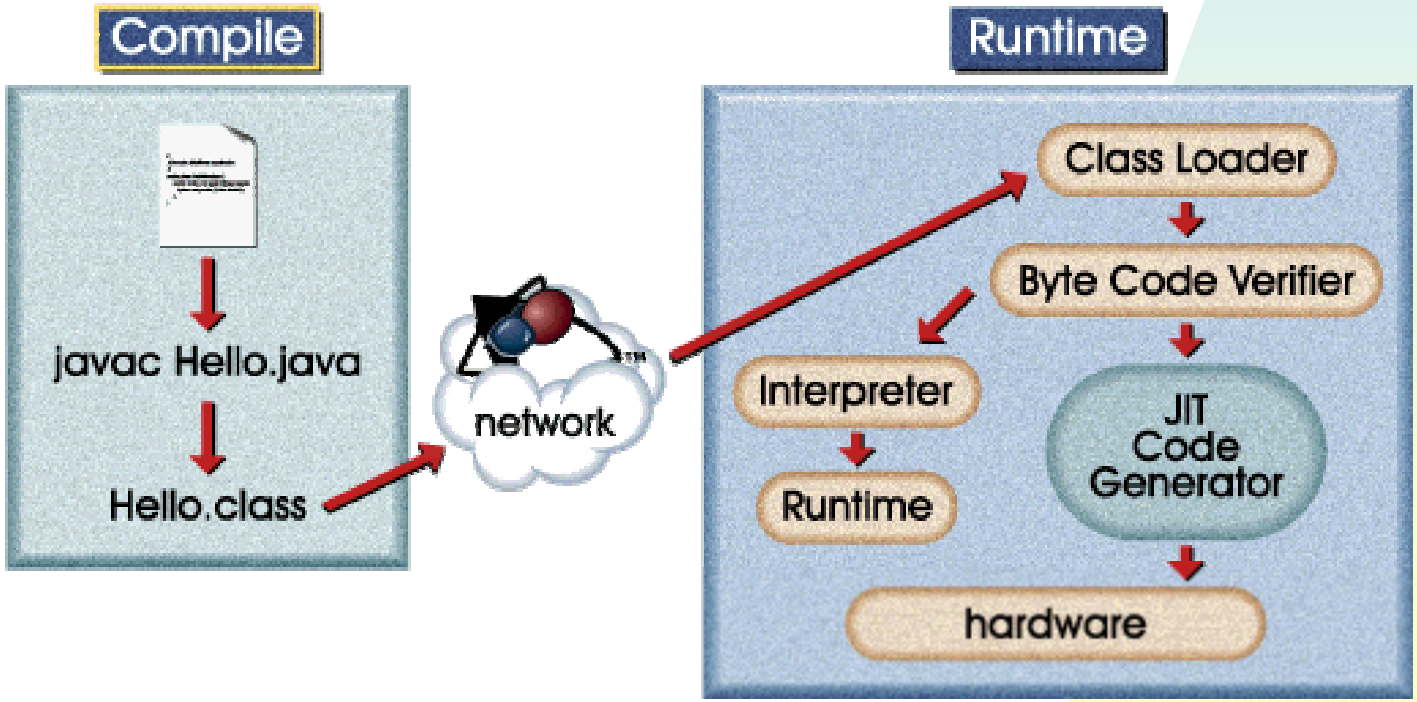
Vantagens

Linguagem de Programação Java

- Origem
  - linguagem originalmente desenvolvida para eletrodomésticos, portanto, simples e portável
  - foi projetada para ser uma linguagem com características modernas de programação
  - nasceu considerando a Internet como ambiente operacional.
- Principais características
  - Propósito geral
  - Orientada a objetos e fortemente tipada
  - Robusta
    - sem ponteiros e alocação direta de memória
    - tratamento de exceções
  - Concorrente

Funcionamento

- Compilação do Fonte (.java) para *bytecode* da Java Virtual Machine (JVM)
- Interpretação e execução do *bytecode* (.class)
- “Escreva uma vez, execute em qualquer lugar”



Outras vantagens da linguagem Java

- Facilidades para desenvolvimento de aplicações em redes com protocolo TCP/IP (sockets, datagrama)
- Gerência automática de memória (*garbage collection*)
- Vários fornecedores de ambientes de desenvolvimento
- Portabilidade
  - Independência de plataforma de hardware e software
- Escalabilidade
  - se for necessário colocar o sistema construído numa máquina mais robusta, provavelmente terá java naquela máquina

Programação em Java

Comentários
Tipos de dados
Literais
Operadores
Expressões
Variáveis
Fluxo de execução

Comentários

- De única linha //
  - **// Exibe na tela**
- De uma ou mais linhas /* */
  - **/* comentário que vai se estendendo até fechar com */**
- De documentação /** */
  - **/** Indicam que o comentário deve ser inserido em qualquer documentação gerada automaticamente, por exemplo pelo javadoc . */**

Ponto-e-vírgulas, blocos, espaços em branco e *case-sensitive*

- Em Java as instruções terminam com ponto-e-vírgula (;).

- Um *bloco* está sempre entre chaves - { e } e constitui uma instrução composta. Dois blocos aninhados são mostrados no exemplo:
  - Instruções compostas para a declaração de classe **{ }**
  - Instruções que abrangem a declaração main method **{ }**

- Os *espaços em branco* são permitidos entre os elementos do código-fonte, sem qualquer restrição. Os espaços em branco incluem espaços, tabulações e novas linhas. Usados para melhorar a aparência visual e a legibilidade do código-fonte
- Java é ***case-sensitive***, ou seja, maiúsculas são diferentes de minúsculas. Exemplo: TRUE != true

Tipos primitivos e seus valores

- Inteiros
  - **byte**: 8 bits, -128 a 127
  - **short**: 16 bits, -32768 a 32767
  - **int**: 32 bits, -2147483648 a 2147483647
  - **long**: 64 bits, ... (200L - literal 200 long)
- Reais
  - **float**: 32 bits (1f - literal 1 float; 1e+9f)
  - **double**: 64 bits (1d - literal 1 double; 47e-341d)
- Caracter
  - **char** (Unicode caracter): 16 bits (‘a’ - literal)
- Lógicos
  - **boolean** (1 bit). Valores literais: { true, false }

Literais

- Literais inteiros (*int*)
  - Decimais: 1 , 2, 45 ,...
  - Octais: 07, 010
  - Hexadecimais: 0xff (255)
- Literais de ponto flutuante – decimais com fração (*double*)
  - 2.0 , 3.14159 , ...
  - 314159e-05 , 314159E-05
- Literais booleanos (*boolean*)
  - *true* e *false*
- Literais de caracteres (*char*)
  - Valores de 16 bits que podem ser manipulados como inteiros
  - Par de apóstrofos (‘ ‘) ‘a’
  - Seqüência de escape (\ ) ‘ \141’ octal = ‘ \u0061’ hex = ‘a’

A classe String

- Representa qualquer seqüência de caracteres
- Valores: “exemplo de valor literal string”

```
String nome = “João da Silva”;
nome.toUpperCase();  // mensagem para objeto nome.
```

- Operador:

```
+ (concatenação)
nome = nome + “sauro”;
```

- Observações
  - Strings são objetos, não são tipos primitivos, porém Java oferece facilidades de manipulação.
  - Strings são objetos que não mudam de valor

Principais métodos da classe String

- boolean equals(String s)
  - retorna true se a string é igual a ***s***
- boolean equalsIgnoreCase(String s)
  - retorna true se a string é igual a ***s*** independente de maiúsculo/minúsculo
- int length()
  - retorna o tamanho da string
- int indexOf(String s)
  - procura a ***s*** na string e retorna a posição, retorna -1 se não achou
- char charAt(int i)
  - retorna o caracter na posição ***i*** da string (começa de 0)

Literais

- Literais de string
  - Cria um objeto para cada literal de string
  - Texto arbitrário entre aspas (“ “)
  - “Hello World”
  - “duas\nlinhas”

Seqüência de escape	**Descrição**
\ddd	Caracter octal
\uxxxx	Caracter UNICODE hexadecimal (xxxx)
\’	Apóstrofo
\”	Aspas
\\	Barra Invertida
\r	Retorno de carro
\n	Nova linha (*linefeed*)
\f	Nova página (*formfeed*)
\t	Tab
\b	Backspace

Conversão de tipos

- Tipos primitivos (**cast**)
  - char a;
  - int i = (int)a;
  - int x = (int)10L;

- String em número
  - String piStr = "3.14159";
  - Float pi = Float.valueOf(piStr);
  - float pi = Float.parseFloat(piStr);

Operadores

- Operadores para números e char
  - relacionais
    - `< > <= >= == !=` (resultado booleano)
  - aritméticos
    - `+ - * / %` (resultado numérico)
    - `++ --` (incremento/decremento)
    - `& (and) ^ (xor) | (or)` (para bits)
- Operadores lógicos
  - booleanos
    - `== !=` (resultado lógico)
    - `&&(and) ||(or) !(not)` (resultado lógico)
- Atribuição
    - `= += -=`

Expressões

- Expressões avaliam (computam) o valor de uma sequência de variáveis, valores, operadores e chamada de métodos
- Exemplos
  - 2
  - 2 * 4
  - a == 3
  - 5 + 2 * 3 - a
  - 2 * Math.sqrt(9)
- Precedência
  - Explícita, utilizando parênteses
  - Implícita: multiplicativo, aditivo, igualdade, &&, ||

Variáveis

- Declaração
  - *tipo nome* [= *expressão*]
  - exemplos: `int a;  char c = ‘a’;  int d=3, e, f=5;`
  - pode ser feita em qualquer lugar do programa
- Nome
  - Identificador válido em Java, diferente de palavras reservadas.
  - Válido = qualquer seqüência descritiva de caracteres de letras, números, caracteres de sublinhado e símbolo de cifrão. Não pode começar com número.
  - Por convenção, variáveis iniciam com letra minúscula
- Escopo
  - Classe (variável membro)
  - Método
- Tempo de vida
  - mesmo que escopo
- Visibilidade
  - método, classe, super-classe (nesta ordem)

Controle de fluxo - Alternativa

if (*expressão*) *comando*
[**else** *comando*]

switch (*expressão*) {
  case *valor1*: *comando*; [**break**];
  case *valor2*: *comando*; [**break**];
  ....
  [**default:** *comando*;]
}

- Exemplos

```
if (resposta == OK) {
   // comandos
} else {
   // comandos
}

int mes;
switch (mes) {
case 1: System.out.println(“Jan”);
   break;
case 2: System.out.println(“Fev”);
   break;
   ....
}
```

Controle de fluxo - Repetição

```
while (expressão)
    comando

for (ini; cond; fim)
  comando

do {
  comandos
} while (expressão);
```

- **break** - sai do loop corrente
- **continue** - volta para o teste do loop

- Exemplos

```
int i = 3;
while (i <= 10) {
  i++;
  System.out.println(i);
}

for (int i=1; i < 10; i++)
  System.out.println(i);
```

Exemplo: números pares

```
class Exemplo1 {

  public void demo() {
     for (int i=0; i<100; i++) {
          if ((i % 2) == 0)  {
               System.out.println(i);
          }
     }
  }
}
```