

# Elementos Básicos da Linguagem Java

3



Histórico

Estrutura de um programa Java

Sintaxe básica de Java

Reinaldo Gomes  
reinaldo@cefet-al.br

## Definição

- O que é a linguagem Java?

Linguagem de programação **orientada a objetos**, de propósito geral, com suporte nativo à **programação concorrente**, **programação distribuída** e à **programação para web**.

Principal fonte de referência: <http://java.sun.com>

## Histórico



### ■ Um pouco de história...

- Janeiro de 1991: A Sun cria o projeto Green;
- Objetivo do projeto Green: definir uma linguagem portátil para programação de aparelhos eletro-eletrônicos de consumo geral – surge **Oak** ;
- Parceria com a Time Warner é desfeita e o projeto Green é quase que totalmente desativado;
- Em 1993: explosão da Internet e da Web faz os remanescentes do projeto Green direcionarem Oak para aplicações na Internet;
- Em 1993-1994: Oak torna-se Java;

3

## Histórico



### ■ Um pouco de história...

- Maio de 1995: Sun anuncia oficialmente Java na SunWorld Expo '95
- Dezembro de 1995: a Microsoft assinou um acordo de intenções com a Sun para licenciamento da tecnologia Java;
- Fevereiro de 1996: Netscape Communications (Navigator) incorporou Java em seu browser Navigator 2.0: surgem os applets;
- Novembro de 1998: Sun lança plataforma Java 2.

4

## Histórico

### ■ Autores...



5

## Estrutura do código fonte Java

- Escrever sistemas em Java consiste em:
  - Escrever classes que implementem partes específicas da funcionalidade do sistema, de acordo com a modelagem feita para o problema
- A classe é a unidade básica de modularização, compilação e execução
  - **Modularização**: é a menor unidade física de código
  - **Compilação**: compilam-se classes, não arquivos
  - **Execução**: a execução de um sistema Java é a execução de sua classe de aplicação

6

## Estrutura do código fonte Java

- Arquivo: menor unidade compilável de Java
- Estrutura:
  - Zero ou uma diretiva `package`
  - Zero ou mais diretivas `import`
  - Uma ou mais definições de classe (uma é o padrão!)
- Todo comando Java deve estar contido dentro dos métodos das classes
- Todo método deve aparecer dentro de alguma classe.

7

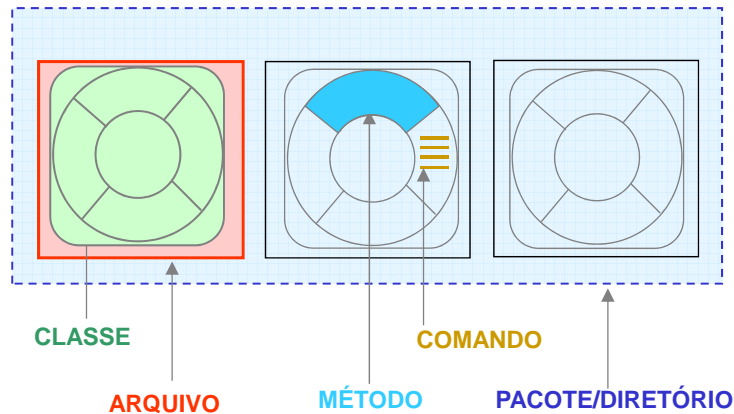
## Estrutura do código fonte Java

- Todo arquivo Java deve conter no máximo uma classe com acesso `public`.
- O nome do arquivo deve ser o mesmo de sua classe `public` com a extensão `.java`.
- É boa prática de programação declarar apenas uma classe por arquivo, mesmo quando estas pertencerem a um mesmo pacote.

8

## Estrutura do código fonte Java

- Um sistema em Java:



9

## Estrutura do código fonte Java

- Esqueleto de um arquivo com uma classe:

```
[package NomeDoPacote]
{[import * ou Classe]}

public class NomeDaClasse
    [extends Superclasse] {

    <corpo da classe>

}
```

[ ] indicam opcionalidade  
{ } indicam repetição

10

## Estrutura de uma aplicação

### ■ Em Java:

- Desenvolver uma aplicação consiste em escrever *classes de objetos*
- As classes podem representar *objetos reais* do *domínio da aplicação* ou *estruturas abstratas*
- Uma *aplicação em execução* consiste num *conjunto de objetos*, criados a partir das classes, *trocando mensagens* entre si
- Saber como fatorar a solução do problema em classes requer experiência e prática


11

## Estrutura de um código fonte Java

### ■ Exemplo de um pequeno sistema em Java:

- Escrever um sistema que leia nome, matrícula e as 3 notas de vários alunos de uma turma e informa suas respectivas médias

### ■ Quais as entidades do domínio do problema?

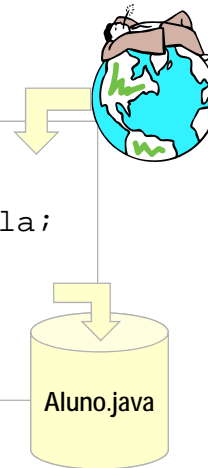
- |             |  |             |
|-------------|--|-------------|
| ▪ Aluno     |  | ▪ Aluno     |
| ▪ Matrícula |  | • Matrícula |
| ▪ Nome      |  | • Nome      |
| ▪ Notas     |  | • Notas     |
| ▪ Turma     |  |             |
- 

12

## Estrutura de um código fonte Java

### ■ Exemplo de uma classe Aluno:

```
public class Aluno {  
    private String nome;  
    private String matricula;  
    private double nota1;  
    private double nota2;  
    private double nota3;  
}
```



13

## Estrutura de um código fonte Java

### ■ Adicionando comportamento à classe:

```
public class Aluno {  
    private String nome;  
    private String matricula;  
    private double nota1;  
    private double nota2;  
    private double nota3;  
  
    public void setNome(String novo){  
        nome = novo;  
    }  
  
    public String getNome() {  
        return nome;  
    }  
}
```

métodos de acesso

Um par de métodos getXxx()  
e setXxx() para cada propriedade

14

## Estrutura de um código fonte Java

### ■ Adicionando comportamento à classe:

```
public class Aluno {  
    private String nome;  
    private String matricula;  
    private double nota1;  
    private double nota2;  
    private double nota3;  
  
    //métodos anteriores...  
    public double getMedia() {  
        return (nota1+nota2+nota3) / 3.0;  
    }  
}
```

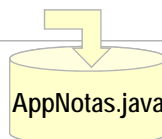
método que informa a média do aluno

15

## Estrutura de um código fonte Java

### ■ Exemplo de uma classe de aplicação:

```
public class AppNotas {  
    public static void main (String[] args) {  
        Aluno a = new Aluno();  
  
        a.setNome("Joao");  
        a.setMatricula("021013123");  
        a.setNota1(7.0);  
        a.setNota2(8.0);  
        a.setNota3(9.0);  
        System.out.println("Nome = " + a.getNome());  
        System.out.println("Media = " + a.getMedia());  
    }  
}
```



16



## Estrutura de um código fonte Java

### ■ Compilando e executando:

```
W:\projetos> javac Aluno.java
W:\projetos> javac AppNotas.java
W:\projetos> java AppNota
Nome = Joao
Media = 8.0
Y:\projetos> _
```

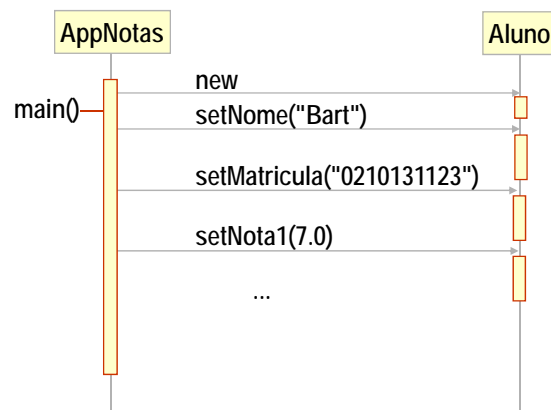
### ■ Compilador Java

- Produz arquivos **.class** chamados *bytecodes*

17

## Estrutura de um código fonte Java

### ■ Colaboração entre os objetos

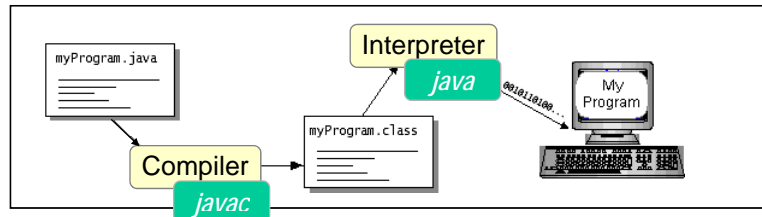


18

## A portabilidade de Java

### ■ Portabilidade

- Uso de uma linguagem de máquina virtual (*bytecode*)



- Não permite tamanhos diferentes para tipos fundamentais de dados em máquinas diferentes.

19

## A portabilidade de Java

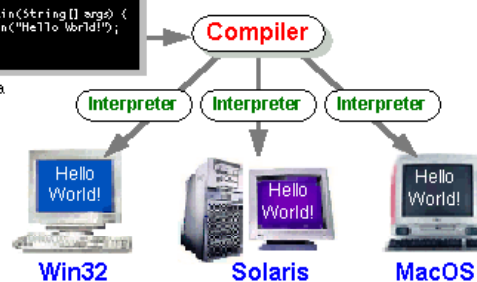
### ■ Portabilidade

#### Java Program

```
class HelloWorldApp {  
    public static void main(String[] args) {  
        System.out.println("Hello World!");  
    }  
}
```

HelloWorldApp.java

*Write once, run anywhere!*



- Java pode ser executada em qualquer máquina que possua o interpretador Java portado para ela;

20

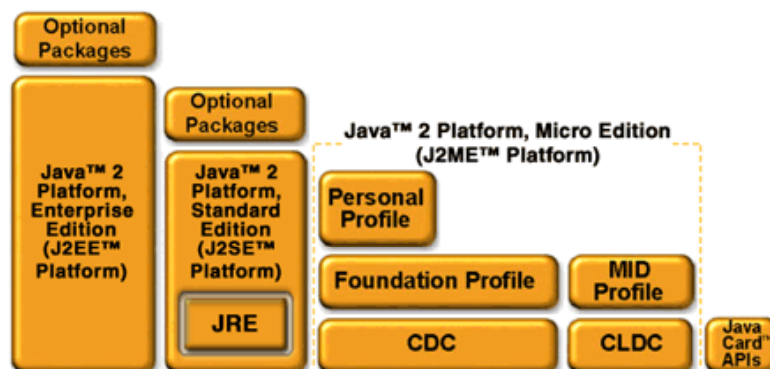
## Plataformas Java

- Java atualmente é distribuída em 3 plataformas:
  - **Java 2 Enterprise Edition (J2EE)**: para desenvolvimento de aplicações corporativas distribuídas
  - ➔ ■ **Java 2 Standard Edition (J2SE)**: para desenvolvimento de aplicações *desktop* comuns  
<http://java.sun.com/j2se/downloads/index.html>
  - **Java 2 Micro Edition (J2ME)**: para desenvolvimento de aplicações embarcadas em equipamentos eletrônicos de baixo consumo (PDAs, celulares, etc.)

21

## Plataformas Java

- Plataformas para diferentes requisitos:



22

## Plataformas Java

### ■ Pequeno Glossário de Java

- **API**
  - Application Program Interface ou interface para programação de aplicações (as bibliotecas de Java! É a alma de Java!)
- **JDK**
  - Nome antigo do SDK (obsoleto, mas ainda muito usado!)
- **SDK** ou **J2SDK**
  - Software Development Kit: nome do kit da Sun com ferramentas e APIs da linguagem para desenvolvimento
  - Ferramentas do kit: javac, java, jar, javadoc, jdb, etc.
- **JRE**
  - Java Runtime Environment: É um pedaço do JDK consistindo apenas do interpretador e algumas poucas ferramentas. É para quem usa e não programa.

23

## Elementos Básicos da Linguagem

- ### ■ Elementos da linguagem, em ordem crescente de complexidade:
- Caracteres
  - Tipos de dados, valores literais, identificadores
  - Operadores e expressões
  - Comandos
  - Métodos
  - Classes
  - Pacotes

24

## Caracteres

- Uso do padrão Unicode: conjunto de caracteres de 16 bits
  - 65536 caracteres distintos!
- Caracteres Unicode são gravados em arquivos usando um formato chamado UTF-8
- Inserção de caracteres Unicode no código fonte:  
`\uxxxx`
  - `\u3c00` =  $\pi$

25

## Comentários

- Três tipos de comentários:
  - `// comentário de linha`
  - `/*  
comentário de bloco  
*/`
  - `/**  
Classe <b>Cliente</b>  
Comentário de bloco para o <i>javadoc</i>  
@author Nome do autor  
*/`

26

## Identificadores

- Identificadores nomeiam variáveis (membro, locais, parâmetros, etc), métodos e classes
- Apenas os caracteres especiais "\_" e "\$" são permitidos, além de letras e números unicode
- Devem iniciar por uma letra ou pelos símbolos \_ ou \$ (evite-os, pois o compilador os utiliza)
  - MAIÚSCULAS ≠ minúsculas

27

## Identificadores (estilo a ser adotado)

- Classes:
  - Primeira letra de cada palavra maiúscula  
Ex: MinhaClasse, Pilha
- Membros de uma classe e variáveis locais:
  - Primeira letra minúscula  
Ex: idade, marcaModelo, getCor(), getSalarioFinal(), i, cont, somaTotal
- Constantes de classe (static final):
  - Todas as letras maiúsculas  
Ex: PI, MAX\_ALUNOS, MEDIA

28

## Palavras reservadas

abstract	do	if	package	synchronized
boolean	double	implements	private	this
break	else	import	protected	throw
byte	extends	instanceof	public	throws
case	false	int	return	transient
catch	final	interface	short	true
char	finally	long	static	try
class	float	native	strictfp	void
<del>const</del>	for	new	super	volatile
<del>continue</del>	<del>goto</del>	null	switch	while
default	<del>assert</del>			

29

## Tipos de Dados em Java

- Java possui duas categorias de tipos de dados:
- **Tipos primitivos**: possuem representações definidas em termos de bytes
  - As variáveis destes tipos guardam valores dentro da faixa definida pelo tipo
- **Tipos referenciáveis**: contêm uma referência para os dados na memória, cujas representações são definidas pela classe do objeto
  - As variáveis destes tipos contêm um ponteiro p/ objeto

30

## Tipos primitivos

<i>Tipo</i>	<i>Default</i>	<i>Tamanho</i>	<i>Domínio</i>
<b>boolean</b>	false	1bit	NA
<b>char</b>	\u0000	16bits	\u0000 a \uFFFF
<b>byte</b>	0	8bits	-128 a 127
<b>short</b>	0	16bits	-32768 a 32767
<b>int</b>	0	32bits	-2147483648 a 2147483647
<b>long</b>	0	64bits	-9223372036854775808 a 9223372036854775807
<b>float</b>	0.0f	32bits	$\pm 1.4E-45$ a $\pm 3.4028235E+38$
<b>double</b>	0.0	64bits	$\pm 4.9E-324$ a $\pm 1.797693134862E+308$

31

## Tipos primitivos

### ■ Tipo **boolean**

- Dois valores: **true** e **false**

### ■ Tipo **char**

- Valores delimitados por apóstrofos: **char c = 'c'**
- Uso de literais unicode: **char c = \u0041**
- Caracteres especiais:
  - \b** (*backspace*)      **\t** (tabulação)
  - \n** (nova linha)      **\r** (return)
  - \'** (apóstrofos)      **\"** (aspas)

32



## Tipos primitivos

### ■ Tipos inteiros (**byte**, **short**, **int**, **long**)

- Literais na base decimal:
  - 0, 1, 123, -23456
- Literais nas bases hexadecimal e octal:
  - 0xcafe //51966 em hexa
  - 0377 //255 em octal
- Inteiros(32 bits) x inteiros longos (64bits)
  - 1234
  - 1234L //long
  - 0xF34L //long em hexa

33

## Tipos primitivos

### ■ Tipos de ponto flutuante (**float** e **double**)

- Literais:
  - 0.0, 1.0, .01, -3.5
- Notação científica:
  - 1.2345E02 //1.2345 x 10<sup>2</sup>
  - 1e-6 //1 x 10<sup>-6</sup>
- Literais ponto-flutuante são sempre **double**!
  - 12.34 //double
  - 12.34f //float
  - 6.02e23F //float

34

## Conversão de tipos

- Java permite conversões entre inteiros (**byte**, **short**, **int**, **long**), caractere (**char**) e pontos flutuantes (**float** e **double**)
- O tipo **boolean** é o único primitivo que não é convertido em nenhum outro
- Tipos de conversão
  - Conversões ampliadoras (*widening conversions*)
  - Conversões redutoras (*narrowing conversions*)

35

## Conversões ampliadoras

- Um valor de um tipo é convertido para um tipo maior, isto é, um representado por mais bits
  - São realizadas automaticamente
- Exemplos:


```
int i = 20;  
float f = i;    //20 convertido para float
```

```
int i = 'A';    //65 convertido para int  
Math.sqrt(4);  //4 convertido para double
```

36

## Conversões redutoras

- Um valor é convertido para um tipo com menos bits
- **Não** são realizadas automaticamente
- O programador deve confirmar a redução com um *cast*, do contrário o código não é compilado
- Exemplos:



```
int i = 13;  
byte b = i;           //Erro de compilação!  
b = (byte) i;         //Ok!  
i = (int) 32.601;     //i recebe 32
```

37

## Operadores

- Operadores *aritméticos*
  - Resultam num valor numérico (inteiro ou ponto flutuante)
- Operadores de *incremento* e *decremento*
  - Resultam num valor numérico (inteiro ou ponto flutuante)
- Operadores *relacionais*
  - Resultam num valor booleano (**true** ou **false**)
- Operadores *lógicos*
  - Produzem um valor booleano

38

## Operadores

- Operadores de *atribuição*
  - Executam uma operação seguida de uma atribuição
- Operador *condicional* (*?:*)
  - Condiciona seu valor de retorno a uma expressão lógica
- Operador *instanceof*: verifica se um objeto é instância de uma classe
  - Resulta num valor boolean
- Operador *new*: instancia uma classe
  - Produz um objeto da classe

39

## Operadores aritméticos

- Símbolos:
  - Adição: +
  - Subtração: -
  - Multiplicação: \*
  - Divisão: /
  - Resto da divisão inteira: %
- Exemplos:

```
c = 7 % 3;           //c receberá 1
x = 5 + 10 / 2;      //x receberá 10
```

40

## Operadores incremento/decremento

### ■ Símbolos:

- Incremento: ++
- Decremento: --
- O comportamento deste operador depende da posição relativa ao operando

### ■ Exemplos:

```
c = 10;  
c++; //c←11
```

```
y=1; x=0; z=0;  
x = y++; //x←1 e y←2  
z = ++x; //z←2 e x←2
```

41

## Operadores relacionais

### ■ Símbolos:

- Maior que, maior ou igual a: >, >=
- Menor que, menor ou igual a: <, <=
- Igual a: ==
- Diferente de: !=

### ■ Exemplos:

```
if ( c == 10 ) ...  
  
return x != null;
```

42

## Operadores lógicos ou booleanos

### ■ Símbolos:

- AND: `&&` ou `&`
- OR: `||` ou `|`
- NOT: `!`
- XOR: `^`

### ■ Exemplos:

```
(c != 0) && (a > (x/c))    //ok
(c != 0) & (a > (x/c))     //erro se c==0
!(a > b)
```

43

## Operadores de atribuição

### ■ Símbolos:

- Aritméticos+atribuição: `+=`, `-=`, `*=`, `/=` e `%=`
- Equivalem a uma soma entre o operando da direita e o operando da esquerda seguida por uma atribuição
- Forma geral: `var op= valor`  $\leftrightarrow$  `var = var op valor`

### ■ Exemplos:

```
c += 5;           //c = c + 5
a -= b;           //a = a - b
f %= (c+a);       //f = f % (c+a)
e += 1;           // e++ ou e = e + 1
```

44

## Operador condicional

### ■ Símbolo:

- Operador ternário: **?:**
- Equivale a um comando de decisão que resulta num valor
- Sintaxe: *expr\_booleana* **?** *expressaoV* **:** *expressaoF*

### ■ Exemplos:

```
j = (a>b)?a:b;  
a = (a != null) ? a : "<vazio>";  
v = r>0 ? 10 : 10.5;    //erro!  
a = (x<y)&&(z<y) ? y : ((x<z) ? z : x);
```

45

## Operador **instanceof**

### ■ Sintaxe:

- (*objeto* ou *array*) **instanceof** *nome\_da\_classe*
- Retorna **true** se o objeto for instância da classe

### ■ Exemplos:

```
"string" instanceof String    //true  
"" instanceof Object          //true  
null instanceof Object         //false
```

46

## Operador **new**

### ■ Sintaxe:

- **new** *construtor\_da\_classe*
- Cria um objeto da classe especificada no construtor

### ■ Exemplos:

```
c = new Cliente("Bart", "Springfield");  
linguagem = new String("Java");  
venda.adicioneProduto(new Produto());  
faixa = new int[] {1,2,3,4,5};
```

47

## Operadores

### ■ (P)recedência e (A)ssociatividade

P	A	Operador
15	E	. [] ( <i>params</i> ) ++ --
14	D	+ - !
13	D	new ( <i>tipo</i> )
12	E	* / %
11	E	+ - + (concatenação de strings)
10	E	<< >> <<<
9	E	< <= > >= instanceof

48



## Operadores

P	A	Operador
8	E	<code>p == p p != p r == r r != r</code> (p=primitivo, r=referência)
7	E	<code>&amp;</code>
6	E	<code>^</code>
5	E	<code> </code>
4	E	<code>&amp;&amp;</code>
3	E	<code>  </code>
2	D	<code>?:</code>
1	D	<code>= *= /= %= += -= &lt;&lt;= &gt;&gt;= &gt;&gt;&gt;= &amp;= ^=  =</code>

49

## Comandos

- Estruturas de decisão:
  - `if-else` e `switch-case`
- Estruturas de repetição:
  - `for`, `while` e `do-while`
- Tratamento de exceções:
  - `throw` e `try-catch-finally`
- Desvio de fluxo:
  - `break`, `continue` e `return`

50

## Comando simples x bloco de comandos

- Um comando em Java pode ser um *comando simples* ou um *bloco de comandos*
- Um *comando simples* pode ser: uma expressão, um dos comandos anteriores ou uma ativação de método;
- Um *bloco de comandos* é um conjunto de um ou mais comandos simples delimitados por "{" e "}" e separados entre si por ";".

51

## Comando if-else

- Sintaxe: `if ( expr_booleana )`  
*comando simples ou bloco;*  
`else`  
*comando simples ou bloco;*

- Exemplo:

```
if (a>b)
    maior = a;
else
    maior = b;
```

```
if (n != 0){
    c += n;
    n = 0;
} else
    c++;
```

52

## Comando `switch-case`

- Sintaxe: `switch ( var_inteira ) {`  
    `[case val: comando ou bloco; break;]`<sub>1-n</sub>  
    `[default: comando ou bloco; break;]`<sub>0-1</sub>  
}

- Exemplo:

(char | byte | short | int)

```
switch (mes) {  
    case 1: nome = "Janeiro"; break;  
    case 2: nome = "Fevereiro"; break;  
    case 3: nome = "Março"; break;  
    case 4: nome = "Abril"; break;  
    default: nome = "Mês inválido"; break;  
}
```

53

## Comando `for`

- Sintaxe: `for ( inic; teste; inc )`  
    `comando simples ou bloco;`
- Onde:
  - **inic**: inicialização da variável contadora. A variável também pode ser declarada aqui.
  - **teste**: expressão booleana que determina a parada ou continuação do laço
  - **inc**: incremento da variável contadora
  - **Interessante** → todas as três cláusulas são opcionais!

54

## Comando `for`

### ■ Exemplos:

```
int i;
for (i=0 ; i<10 ; i++)
    x += 5;

//outra forma
for (int i=0 ; i<10 ; i++)
    x += a[i];

//laço infinito
for( ;; ) {
    if (x != null) break;
    else { ... }
}
```

`i` só pode ser usada no corpo do laço `for`

55

## Comando `for`

### ■ Exemplos:

```
//mais de uma variável declaradas
for (int i=0, j=10 ; i<10 ; i++, j--)
    soma += i*j;

//não precisa ser uma contagem
for (Node n = cabeca ; n!=null ; n=n.next())
    System.out.println(n);

//laço vazio (e possível erro de compilação)
for (int i=0 ; i<10 ; i++);
```

56

## Comando **while**

- Sintaxe: **while** ( *expressão* )  
*comando ou bloco*;
- Onde:
  - **expressão**: expressão que retorne um booleano
    - Se o valor for **true**, o corpo do laço é executado
    - Se for **false**, o laço é encerrado
  - Pode ser análogo a um **for**, desde que se acrescente uma operação de inicialização antes do comando

57

## Comando **while**

### ■ Exemplos:

```
int c=0;
while (c < 10) {
    System.out.println(c);
    c++;
}

//equivale a
for (int c=0; c<10 ;c++)
    System.out.println(c);
```

c só pode ser usada no corpo do laço for

58

## Comando **do-while**

- Sintaxe: **do**

*comando* ou *bloco*;  
**while** (*expressão*);

- Onde:

- *expressão*: expressão que retorne um booleano
- O laço **do-while** é executado pelo menos uma vez
- Só pára se a expressão for **false** (cuidado para não confundi-lo como repeat-until do Pascal)

59

## Comando **do-while**

- Exemplos:

```
//equivalente ao laço while do exemplo  
int c=0;  
do {  
    System.out.println(c);  
    c++;  
} while (c < 10);
```

60

## Comando **break**

- Sintaxe: **break**;
- Objetivo:
  - Faz a execução saltar para o final de um dos comandos: **while**, **do-while**, **for** ou **switch-case**
  - No comando **switch-case** seu uso é necessário para evitar que a execução de uma cláusula **case** continue na seguinte

61

## Comando **break**

- Exemplo:

```
...  
for (int i=0; i<dados.length ;i++) {  
    if (dados[i] == procurado) {  
        posicao = i;  
        break;  
    }  
} //a execução continua daqui  
...
```

62

## Comando `continue`

- Sintaxe: `continue`;
- Objetivo:
  - Interrompe a iteração atual do laço e inicia a próxima
  - Só pode ser usado dentro de laços **while**, **do-while** e **for**
    - **while**: volta para o início do laço e testa a condição de parada para decidir se entra no laço ou não
    - **do-while**: vai para o final do laço, onde a condição de parada é testada para decidir se entra no laço ou não
    - **for**: volta para o início, executa o incremento e depois o teste

63

## Comando `continue`

- Exemplo:

```
...  
for (int i=0; i<dados.length ;i++) {  
    if (dados[i] == VAZIO)  
        continue;  
    processe(dados[i]);  
}  
...
```

64



## Comando **return**

- Sintaxe: **return**; ou **return** *expressão*;
- Objetivo:
  - Para a execução do método corrente
  - Métodos com tipo de retorno **void** pedem a primeira sintaxe do **return**
  - Métodos que retornam dados pedem a segunda

65

## Comando **return**

- Exemplos:

```
String informeNome() {  
    return this.nome;  
}  
  
public void insereProduto(Produto item) {  
    if (item == null)  
        return;  
    itens.add(item);  
}
```

66

## Aula Prática 2

### ■ Objetivos:

- Implementar classes simples
- Testar classes implementadas
- Executar uma classe de aplicação

67

## Bibliografia

- [1] **Booch**, G. *Object Oriented Design*. 1991.
- [2] **Campione**, M. **Walrath**, K. *The Java Tutorial*.  
Disponível em  
<http://java.sun.com/docs/books/tutorial/index.html>.
- [3] **Flanagan**, D. *Java in a Nutshell*. O'Reilly. 1999.

68

# Elementos Básicos da Linguagem Java

3



Histórico

Estrutura de um programa Java

Sintaxe básica de Java

Reinaldo Gomes  
reinaldo@cefet-al.br

## Operadores

P	A	Operador
8	E	<code>p == p p != p r == r r != r</code> (p=primitivo, r=referência)
7	E	<code>&amp;</code> (bitwise) <code>&amp;</code> (lógico)
6	E	<code>^</code> (bitwise) <code>^</code> (lógico)
5	E	<code> </code> (bitwise) <code> </code> (lógico)
4	E	<code>&amp;&amp;</code>
3	E	<code>  </code>
2	D	<code>?:</code>
1	D	<code>= *= /= %= += -= &lt;&lt;= &gt;&gt;= &gt;&gt;&gt;= &amp;= ^=  =</code>

70