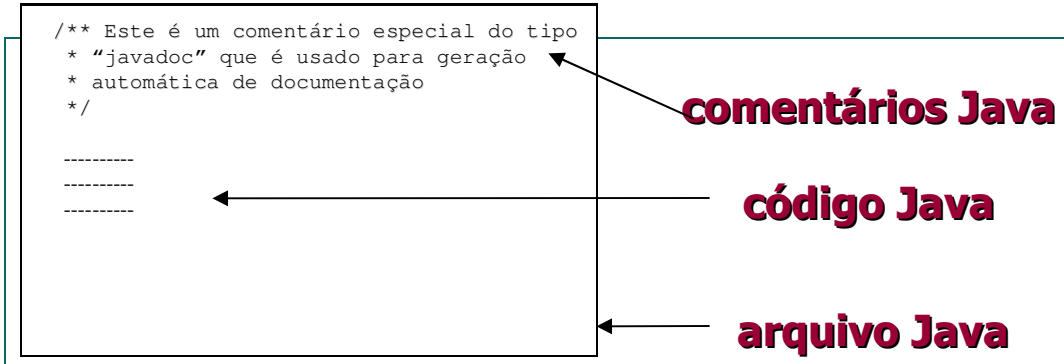
Sintaxe Básica de Java



- // Este é um comentário de uma única linha

- /* Este comentário pode ocupar várias linhas sem problemas */

- /** Este é um comentário especial do tipo

  * "javadoc" que é usado para geração
  * automática de documentação
  */

Identificadores

- Identificam elementos de um programa Java
  - métodos, atributos, rótulos, …
- Regras para identificadores
  - Devem iniciar por uma letra, um “sublinhado” (_) ou o símbolo do dólar ($). Caracteres subseqüentes podem ser letras, dígitos, sublinhados ou $.
  - *São “Case sensitive”*:
    - Maiúsculas são diferenciadas de minúsculas

Identificadores

- Identificadores válidos
  - soma
  - temp01
  - _numClientes
  - $fortuna
  - nomeLongoDeVariavel
- Identificadores inválidos
  - 102dalmatas
  - 123
  - #x

Palavras reservadas

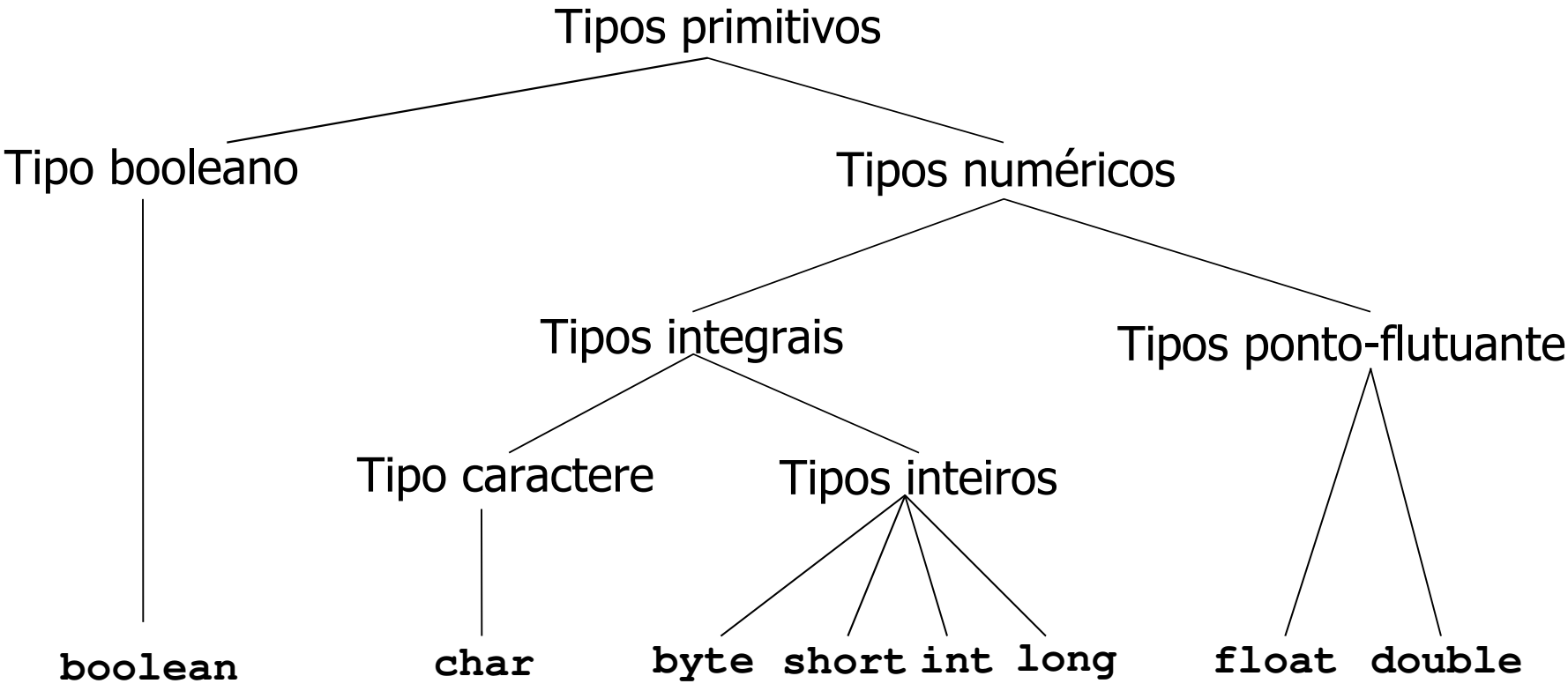
```
abstract
assert
boolean
break
byte
case
catch
char
class
const
continue
default
do
double
else
enum
extends
false
final
finally
float
for
goto
if
implements
import
instanceof
int
interface
long
native
new
null
package
private
protected
public
return
short
static
strictfp
super
switch
synchronized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

Não podem ser usadas como identificador!!!

Tipos Primitivos

boolean	true ou false
char	caractere (16 bits Unicode)
byte	inteiro (8 bits)
short	inteiro (16 bits)
int	inteiro (32 bits)
long	inteiro (64 bits)
float	ponto flutuante (32 bits)
double	ponto flutuante (64 bits)

Hierarquia dos Tipos



Declaração de Variáveis

```
int LIMITE_MAXIMO;

double saldo = 100.5;

float saldo = 100.5f;

int quantidade, idade;
```

Padrão de codificação. Não é restrição da sintaxe de Java

- Padrão para nome de variáveis:
  - começam com letras minúsculas;
  - em caso de palavras compostas a primeira letra da palavra seguinte é maiúscula.

```
int quantidadeMaxima;
```

Operadores

Tipos de operadores

- Aritméticos
- Concatenação
- Comparação
- Lógicos
- Atribuição
- Unários
- Condicional (ternário)

Operadores aritméticos

+ - * / %

- O operador / é também utilizado para calcular divisões inteiras
- O operador % calcula o resto de uma divisão inteira

1/2 => 0

16%5 =>1

1%2 => 1

16/5 => 3

Operador de concatenação

- **+** (aplicado a Strings)

```
String nomeCompleto = nome + sobrenome;
```

A concatenação também faz uma conversão implícita para String

```
mensagem = "Este é o cliente número " + x;
```

```
System.out.println("Total: " + total);
```

Operadores de Comparação

- Operadores de comparação
  - >
  - <
  - >=
  - <=
  - ==
  - !=

Operadores lógicos

- Operadores booleanos
  - short-circuit
    - && (E lógico)
    - || (OU lógico)

Operadores lógicos

- Operadores booleanos
  - bitwise
    - & (E lógico ou bit-a-bit)
    - | (OU lógico ou bit-a-bit)
    - ^ (OU-EXCLUSIVO bit-a-bit)

Atribuição

- Atribuição
  - =
  - +=,-=,*=,/=

```
x = 0;
```

```
x += 1;
```
⟺
```
x = x + 1;
```

```
a = b = c = -1;
```

```
y -= k;
```
⟺
```
y = y - k;
```

```
y -= X + 5;
```
⟺
```
y = y - (x + 5);
```

Unários

- **++, --**

```
y = ++x     y = --x

y = x++     y = x--
```

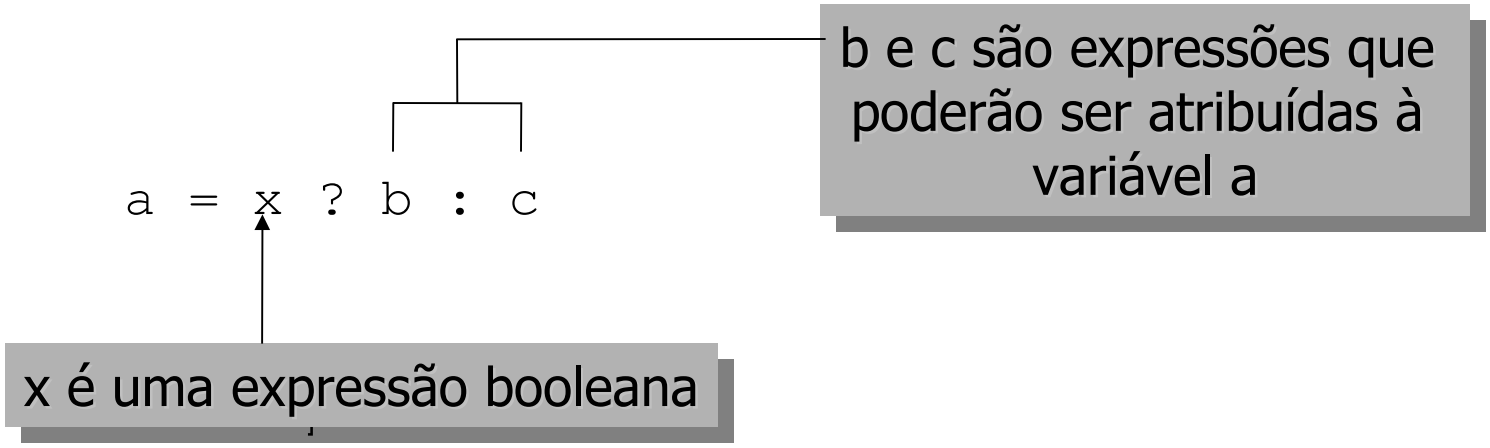
Usar esses operadores com cuidado!

- **-,!**

```
y = -x      y = !x
```

Condicional

- Conhecido também como operador ternário
  - ?:



Uso dos Operadores

- Ordem de avaliação dos operadores

- Associatividade

Ordem de avaliação dos operadores

- Ordem de precedência (maior para menor):
  - expr++, expr--
  - ++expr, --expr, +expr, -expr
  - (tipo) expr
  - *, /, %
  - +, -
  - <, > >=,<=
  - ==, !=
  - &&
  - ||
  - =, +=, -=, *=, /=

Associatividade

- Quando os operadores possuem a mesma precedência, avalia-se primeiro o operador mais a esquerda
  - Exemplo: `a + b + c` equivale a `(a + b) + c`
- (exceção) Todos os operadores binários de Java são associativos a esquerda, exceto a atribuição
  - Exemplo: `a = b = c` equivale a `a = (b = c)`
- Precedência e associatividade podem ser redefinidas através de parênteses
  - Exemplo: `a*(b+c), a + (b + c)`

Conversão de Tipos

Introdução

- Conversões entre tipos, e *Casts,* acontecem freqüentemente quando programamos em Java

```
double d1 = 10.0d;
System.out.println("Soma: " + (d1 + 10));
...
byte b = (byte) 32.0d;
...
```

Conversões ocorrem

- Entre tipos primitivos
  - Atribuição
  - Passagem de parâmetros
  - Promoções aritméticas

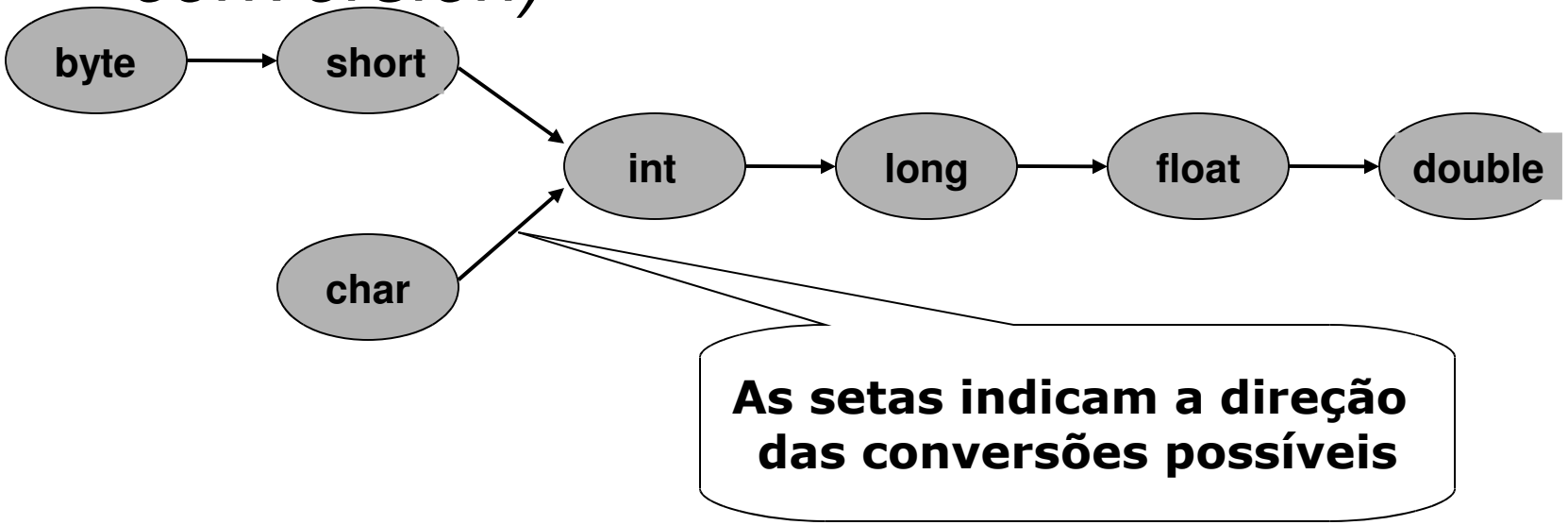
- Entre objetos
  - Atribuição
  - Passagem de parâmetros

Conversões entre tipos primitivos

- Widening conversion
  - Conversão para um tipo de maior capacidade
- Narrowing conversion
  - Conversão para um tipo de menor capacidade
  - Pode haver perda de informação
- Essas conversões podem ser
  - Implícitas
  - Explícitas

Atribuição e passagem de parâmetros

- É sempre possível quando a conversão ocorre de um tipo "menor" para um tipo "maior" *(widening conversion)*



Promoção aritmética

- Acontece quando valores de tipos diferentes são operandos de uma expressão aritmética
- Operadores binários:
  - O tipo do operando de menor tamanho (bits) é promovido para o tipo do operando de maior tamanho
  - Os tipos *short*, *char* e *byte* são sempre convertidos para o tipo *int* em operações envolvendo apenas esses tipos

Promoção aritmética

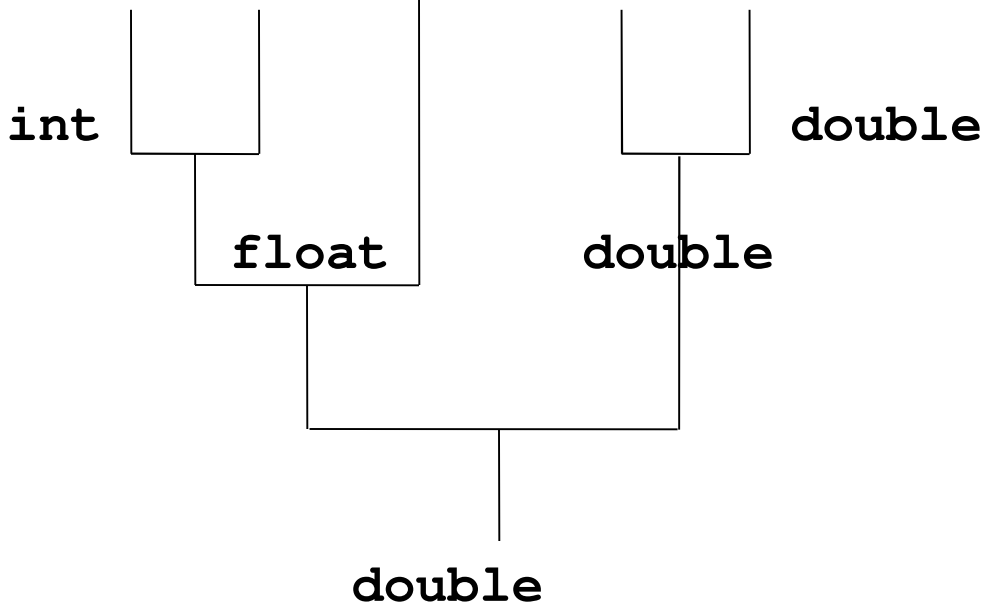
- Operadores unários:
  - Os tipos *short*, *char* e *byte* são sempre convertidos para o tipo *int* (exceto quando usados ++ e --)
  - Demais tipos são convertidos de acordo com o maior tipo sendo utilizado na mesma expressão

```
short s = 9;
int i = 10;
float f = 11.1f;
double d = 12.2d;
if (-s * i >= f / d) {
    ...
}
...
```

Exemplo de promoção aritmética

```
byte b = 1;
int i = 1;
float f = 5.2f;
double d = 7.5;
```

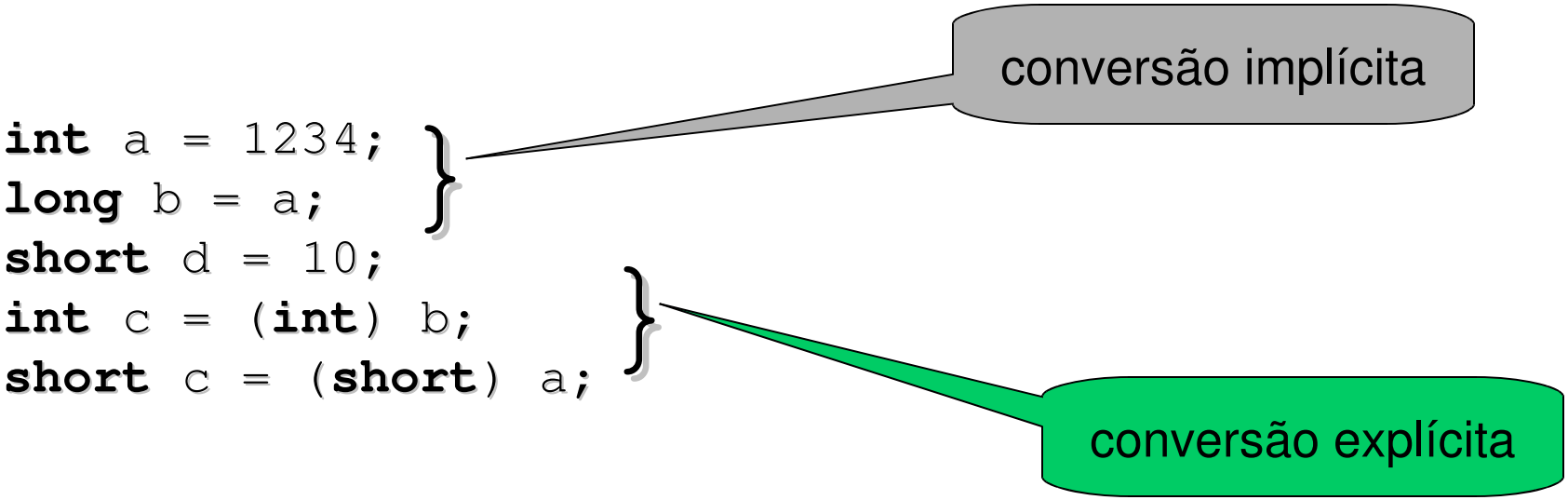
```
((b + i) * f) / (d + i)
```



O operador “cast”

- Usado para conversões
- Sintaxe

  (<*tipo*>) <*expressão*>



Cast entre tipos primitivos

- Casts podem ser realizados entre quaisquer tipos primitivos, exceto boolean

```
double d = 10.0d;
int i = (int) d;
```

Casts envolvendo o tipo boolean não são permitidos!

Estruturas de Controle

Estruturas de Controle

- if
- if-else
- if-else-if
- switch-case
- while
- do-while
- for

if

- Declaração condicional mais simples em Java

```
if (expressão booleana){
    comando
}
```

```
if (nomeUsuario == null){
nomeUsuario = "indefinido";
}
```

```
if (vendas >= meta){
        desempenho = "Satisfatório";
        bonus = 100;
}
```

if-else

```
if (expressão booleana){
    comando1
}else{
    comando2
}
```

```
if (media < 5){
    resultado = "Reprovado";
}else {
    resultado = "Aprovado";
}
```

```
if (vendas >= meta){
      desempenho = "Satisfatório";
      bonus = 100+ 0.01*(vendas-meta);
}else {
      desempenho = "Não Satisfatório";
      bonus = 0;
}
```

if-else-if

```
if (expressão booleana){
    bloco de comandos
}else if (expressão booleana){
    bloco de comandos
}else {
    bloco de comandos
}
```

```
if (vendas >= 2*meta){
        desempenho = “Excelente”;
        bonus = 1000;
} else if (vendas >= 1.5*meta){
        desempenho = “Boa”;
        bonus = 500;
} else if (vendas >= meta){
        desempenho = “Satisfatório”;
        bonus = 100;
} else {
        System.out.println(“Você está em apuros”);
}
```

switch-case

```
switch(<expressão inteira>) {
    case 1:
        // Bloco de código 1
           break;
    case 2:
        // Bloco de código 2
           break;
    case 3:
        // Bloco de código 3
           break;
    default:
           // Bloco de código
}
```

Tipo da expressão deve ser byte, char, short ou int

Executado somente quando todos os outros cases falham

switch-case

```
boolean analisarResposta(char resposta)
{
   switch(resposta) {
      case 's':
      case 'S': return true;
      case 'n':
      case 'N': return false;
      default:
          System.out.println(
              "Resposta inválida!");
          return false;
   }
}
```

return também pode ser usado para sair do **case**

while

```
while (expressão booleana){
      comando
}
```

Teste é feito no **início**

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

```
x = 10;
while (x < 10)
    x = x + 1;
```

pode ser executado
0 vezes!

```
while (true)
System.out.println("Casa Forte");
```

loop infinito

do-while

```
do{
   comando
}while(expressão booleana);
```

Teste é feito no **final**

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

comandos são executados pelo menos uma vez

```
String resposta;
do {
  resposta = "Resposta Incorreta!";
}
while (ehInvalida(resposta));
```

for

```
for (inicialização; condição;
   incremento){
```

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

```
void tabuada() {
  int x,y;
  for (x=1, y=1; x<=10; x++, y++) {
      System.out.print(x + " X " + y + " = ");
      System.out.println(x*y);
  }
}
```

break

- Usado para terminar a execução de um bloco **for**, **while**, **do** ou **switch**

```
int procurar(String nome) {
  int indice = -1;
  for (int i = 0; i < 50; i++) {
    if (i < MAXIMO ) {
       indice = i;
       break;
    }
  }
  return indice;
}
```

continue

- Termina a execução da iteração atual do loop e volta ao começo do loop.

```
for (int k = 0; k < 10; k++) {
    if (k == 5) {
        continue;
    }
    System.out.println("k = " + k);
}
```

return

- Termina a execução de um método e retorna a chamada ao invocador
- É obrigatório quando o tipo de retorno do método não é `void`
- O resultado da avaliação de *`expressão`* deve poder ser atribuído ao tipo de retorno do método

```
int somar (int x, int y) {
   return x + y;
}
```

o tipo de retorno deve ser compatível com o tipo de retorno do método

métodos sem retorno também podem usar **return**

```
void atualizarDados() {
   ...
   if (dadosAtualizados)
     return;
   ...
}
```