Tipos, literais, operadores e controle de fluxo

Helder da Rocha
www.argonavis.com.br

Operadores e controle de fluxo da execução

- operadores
  - aritméticos, lógicos, binários, booleanos, de deslocamento, de concatenação, de conversão, ...
- conversão de tipos
  - promoção
  - coerção
- estruturas de controle de execução
  - if-else,
  - for, while, do-while
  - break, continue, rótulos
  - switch (case)

Operadores

- *Um operador produz um novo valor a partir de um ou mais argumentos*
- *Os operadores em Java são praticamente os mesmos encontrados em outras linguagens*
  - +, -, /, *, =, ==, <, >, >=, &&, *etc.*
- *A maior parte dos operadores só trabalha com valores de tipos primitivos.*
- *Exceções:*
  - + e += *são usados na concatenação de strings*
  - !=, = e == *são usados também com objetos (embora não funcionem da mesma forma quanto aos valores armazenados nos objetos)*

Precedência

- A precedência determina em que ordem as operações em uma expressão serão realizadas.
  - Por exemplo, operações de multiplicação são realizadas antes de operações de soma:
  - `int x = 2 + 2 * 3 – 9 / 3;  // 2 + 6 – 3 = 5`
- Parênteses podem ser usados para sobrepor a precedência
  - `int x = (2 + 2) * (3 – 9) / 3;  // 4 * (– 6) / 3 = – 8`
- A maior parte das expressões de mesma precedência é calculada da esquerda para a direita
  - `int y = 13 + 2 + 4 + 6; // (((13 + 2) + 4) + 6)`
  - há exceções. Por exemplo, atribuição.

Atribuição

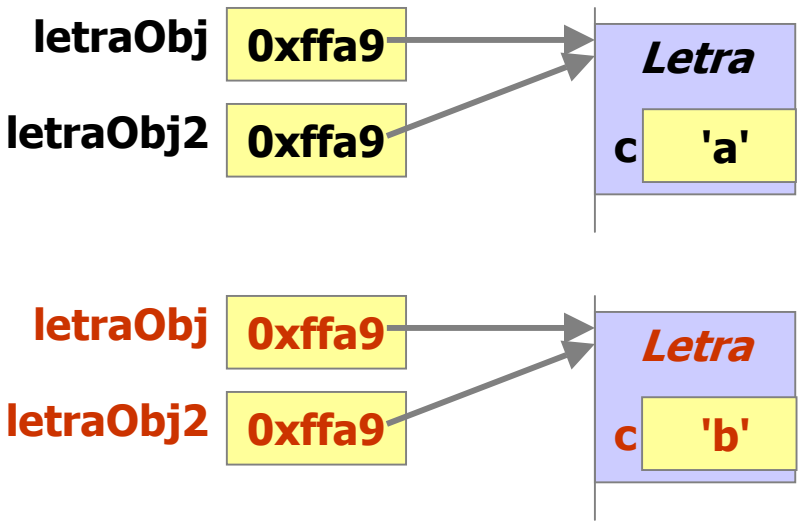
- A atribuição é realizada com o operador '='
  - '=' serve apenas para atribuição – não pode ser usado em comparações (que usa '==')!
  - copia o valor da variável ou constante do lado direito para a variável do lado esquerdo.
    - `x = 13;  // copia a constante inteira 13 para x`
    - `y = x; // copia o valor contido em x para y`
- A atribuição copia valores
  - o valor armazenado em uma variável de tipo primitivo é o valor do número, caractere ou literal booleana (true ou false)
  - O valor armazenado em uma variável de tipo de classe (referência para objeto) é o ponteiro para o objeto ou null.
    - conseqüentemente, copiar referências por atribuição não copia objetos mas apenas cria novas referências para o mesmo objeto!

Passagem de valores via atribuição

- *Variáveis de tipos primitivos*


```
(...)
char letraPri = 'a';
char letraPri2 = letraPrj;
letraPri = 'b';
(...)
```

- *Referências de objetos*




```
public class Letra {
   public char c;
}
```

```
(...)
Letra letraObj = new Letra();
letraObj.c = 'a';
Letra letraObj2 = letraObj;
letraObj2.c = 'b';
(...)
```

Operadores matemáticos

- + *adição*
- - *subtração*
- * *multiplicação*
- / *divisão*
- % *módulo (resto)*

- *Operadores unários*
  - *-n e +n (ex: -23) (em uma expressão: 13 + -12)*
  - *melhor usar parênteses: 13 + (-12)*
- *Atribuição com operação*
  - +=, -=, *=, /=, %=
  - *x = x + 1 equivale a x += 1*

Incremento e decremento

- Exemplo
  - int a = 10;
  - int b = 5;
- Incrementa ou decrementa antes de usar a variável
  - int x = ++a; // a contém 11, x contém 11
  - int y = --b; // b contém 4, y contém 4
  - A atribuição foi feita **DEPOIS**!
- Incrementa ou decrementa depois de usar a variável
  - int x = a++; // a contém 11, x contém 10
  - int y = b--; // b contém 4, y contém 5
  - A atribuição foi feita **ANTES**!

Operadores relacionais

- == igual
- != diferente
- < menor
- <= menor ou igual
- > maior
- >= maior ou igual

- Sempre produzem um resultado booleano
  - true ou false
  - comparam os valores de duas variáveis ou de uma variável e uma constante
  - comparam as referências de objetos (apenas == e !=)

Operadores lógicos

- && E (and)
- || Ou (or)
- ! Negação (not)

- Produzem sempre um valor booleano
  - true ou false
  - argumentos precisam ser valores booleanos ou expressões com resultado booleano
  - Por exemplo: (3 > x) && !(y <= 10)
- Expressão será realizada até que o resultado possa ser determinado de forma não ambígua
  - "short-circuit"
  - exemplo: (false && <qualquer coisa>)
    - a expressão <qualquer coisa> não será calculada

Operadores orientados a bit

- & *and*
- | *or*
- ^ *xor (ou exclusivo)*
- ~ *not*

- *Para operações em baixo nível (bit por bit)*
  - *operam com inteiros e resultados são números inteiros*
  - *se argumentos forem booleanos, resultado será igual ao obtido com operadores booleanos, mas sem 'curto-circuito'*
  - *suportam atribuição conjunta:* &=, |= *ou* ^=

Operadores de deslocamento

- << deslocamento de bit à esquerda (multiplicação por dois)
- >> deslocamento de bit à direita (divisão truncada por dois com sinal)
- >>> deslocamento à direita sem considerar sinal (acrescenta zeros)
- Para operações em baixo nível (bit a bit)
  - operam sobre inteiros e inteiros longos
  - tipos menores (short e byte) são convertidos a int antes de realizar operação
  - podem ser combinados com atribuição: <<=, >>= ou >>>=

Operador ternário (if-else)

- *Retorna um valor ou outro dependendo do resultado de uma expressão booleana*

```
variavel = expressão ? valor, se true
                     : valor, se false;
```

  - *ex:*

```
int x = (y != 0) ? 50 : 500;
String titulo = (sexo == 'f')
                ? "Sra." : "Sr.";
```

- *Use com cuidado*
  - *Pode levar a código difícil de entender*

Operador de concatenação

- *Em uma operação usando "+" com dois operandos, se um deles for String, o outro será convertido para String e ambos serão concatenados*
- *A operação de concatenação, assim como a de adição, ocorre da direita para a esquerda*
  - *String s = 1 + 2 + 3 + "=" + 4 + 5 + 6;*
  - *Resultado: s contém a String "6=456"*

Exercício

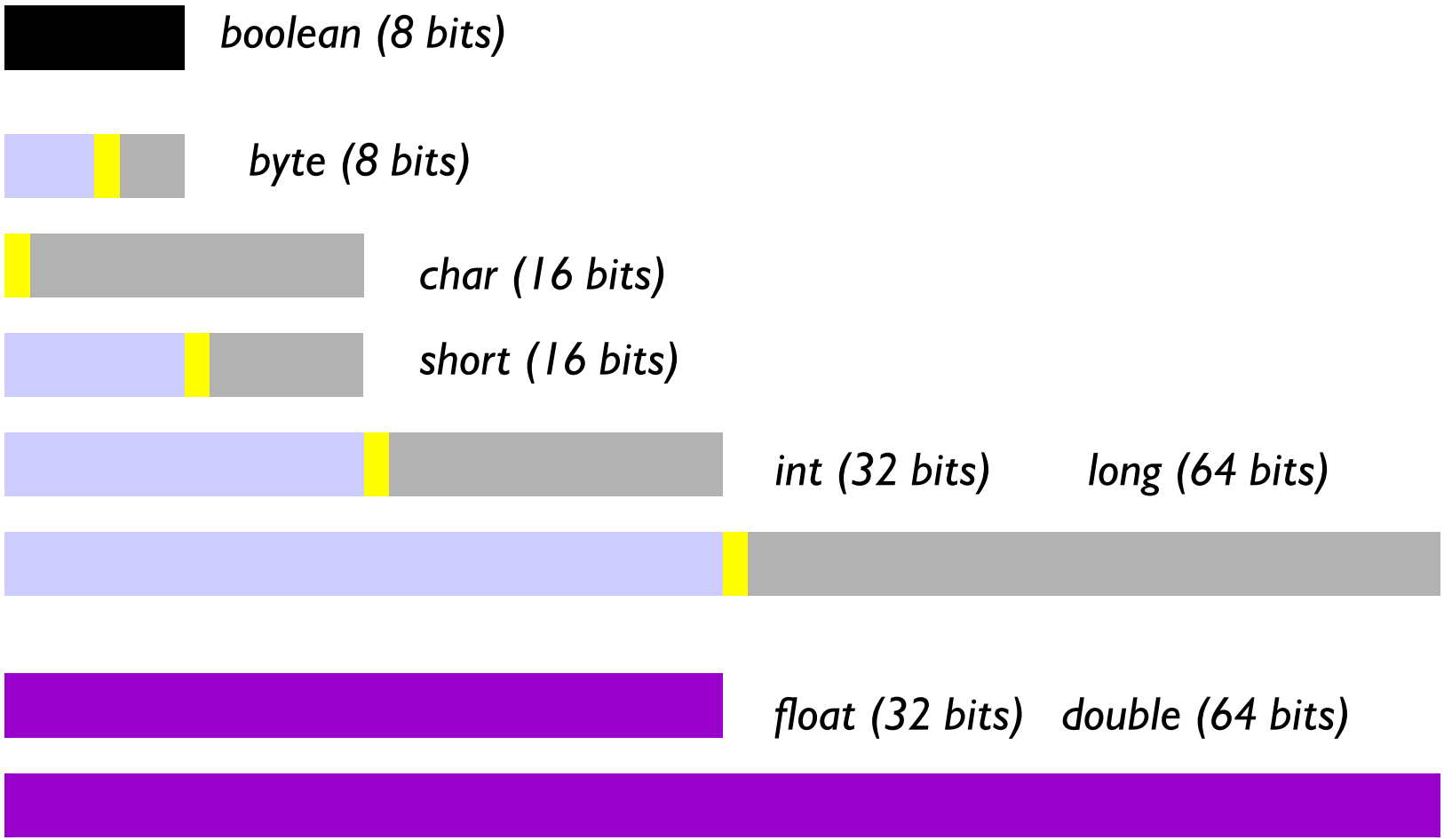
- *1. Crie uma classe TesteOperadores*
- *2. Declare variáveis do tipo int, double e String (duas de cada)*
- *3. Imprima as variáveis e resultados de operações que incluam*
  - *a-d) soma, subtração, divisão, multiplicação*
  - *e-f) incremento e decremento*
  - *g) operador ternário*
  - *h) operadores condicionais*

instanceof

- *instanceof é um operador usado para comparar uma referência com uma classe*
  - *a expressão será true se a referência for do tipo de uma classe ou subclasse testada e false, caso contrário*
  - *Sintaxe: referência instanceof Classe*
- *Exemplo:*

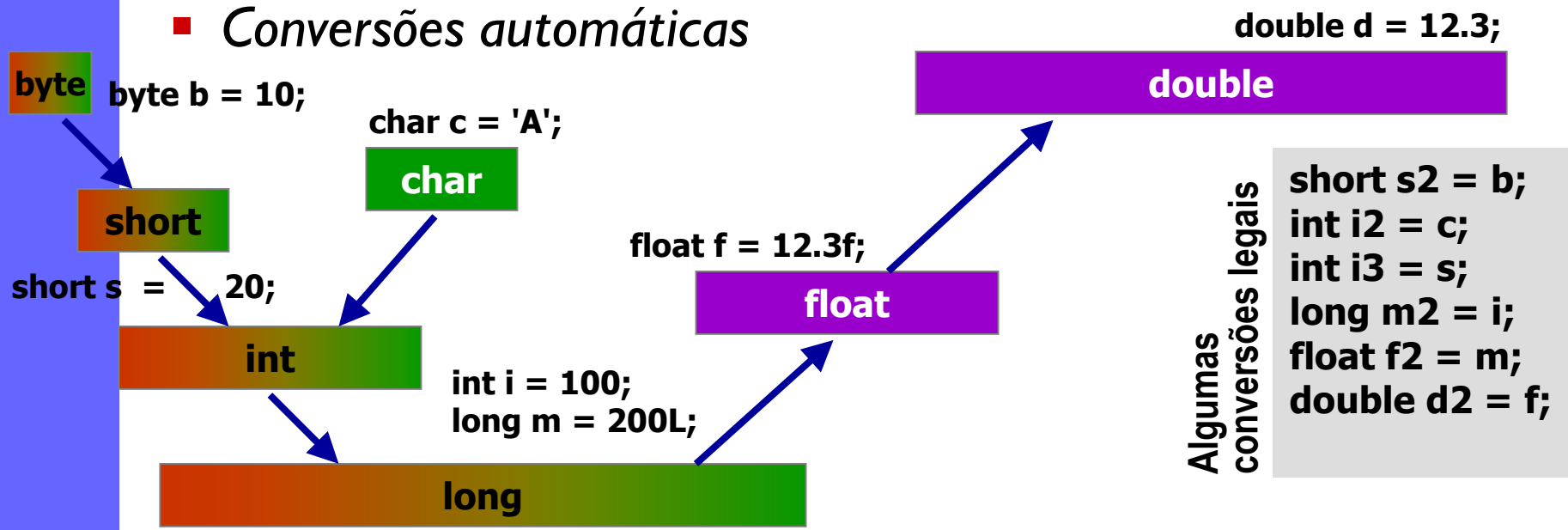
```
if (obj instanceof Point) {
  System.out.println("Descendente de Point");
}
```

Tipos de dados



Conversão de tipos primitivos

- *Java converterá um tipo de dados em outro sempre que isto for apropriado*
  - *as conversões ocorrerão automaticamente quando houver garantia de não haver perda de informação*
    - *tipos menores em tipos maiores*
    - *tipos de menor precisão em tipos de maior precisão*
    - *inteiros em ponto-flutuante*
- *Conversões automáticas*



Conversão de referências

- *Pode-se atribuir uma referência A a uma outra referência B de um tipo diferente, desde que*
  - *B seja uma superclasse (direta ou indireta) de A*
    - *Qualquer referência pode ser atribuída a uma referência da classe Object*
  - *B seja uma interface implementada por A*
    - *mais detalhes sobre interfaces em aulas futuras*

```
class Carro extends Veiculo {...}

class Veiculo implements Dirigivel {...}

class Porsche extends Carro {...}
```

Algumas conversões legais

```
Carro c = new Carro();
Veiculo v = new Carro();
Object o = new Carro();
Dirigivel d = new Carro();
Carro p = new Porsche();
```

Operadores de coerção

- *Na coerção (cast), o programador assume os riscos da conversão de dados*
  - *No tipo byte cabem inteiros até 127*
  - *No tipo short cabem inteiros até 65535*
  - *Não há risco de perda de informação na atribuição a seguir*
    ```
    short s = 100; byte b = s;
    ```
  - *(pois 100 cabe em byte) mas o compilador acusará erro porque short não pode ser atribuído a byte.*
  - *Solução* ***operador de coerção (cast)***
    ```
    byte b = (byte) s;
    ```
  - *O programador atesta, entre parênteses, que o conteúdo de s cabe em byte.*
  - *O operador de coerção tem maior precedência que os outros operadores!*

Promoção

- *Qualquer operação com dois ou mais operandos de tipos diferentes sofrerá promoção, isto é, conversão automática ao tipo mais abrangente, que pode ser*
  - *o maior ou mais preciso tipo da expressão (até double)*
  - *o tipo int (para tipos menores que int)*
  - *o tipo String (no caso de concatenações) - (na verdade isto não é uma promoção)*
- *Exemplos*
  - `String s = 13 - 9 * 16 + "4" + 9 + 2; // "-131492"`
  - `double d = 12 + 9L + 12.3; // tudo é promovido p/ double`
  - `byte b = 9; byte c = 10; byte d = 12; byte x = (byte) (b + c + d);`

a partir daqui só o sinal '+' é permitido!

cast é essencial aqui! Observe os parênteses!

promovido para int!

Exercícios

- *1. Crie uma classe TesteOperadores2*
- *2. Crie duas variáveis de cada um dos tipos*
  - *byte, int, short, char, double, float*
- *3. Escreva 10 expressões de soma usando as variáveis e atribuíndo os resultados a variáveis de tipo*
  - *byte, int, char, double*
- *4. Use coerção (cast) quando necessário*
- *5. Imprima os resultados*

Controle de execução

- O controle do fluxo da execução em Java utiliza os mesmos comandos existentes em outras linguagens
  - repetição: for, while, do-while
  - seleção: if-else, switch-case
  - desvios (somente em estruturas de repetição): continue, break, rótulos
- Não existe comando goto
  - goto, porém, é palavra-reservada.

true e false

- *Todas as expressões condicionais usadas nas estruturas for, if-else, while e do-while são expressões booleanas*
  - *O resultado das expressões deve ser sempre true ou false*
  - *Não há conversões automáticas envolvendo booleanos em Java (evita erros de programação comuns em C/C++)*

Código errado.
Não compila em Java

```
int x = 10;
if (x = 5) {
   ...
}
```

código aceito em C/C++ (mas provavelmente errado) x, com valor 5, converte-se em 'true'.

Código correto.
x == 5 é expressão com resultado true ou false

```
int x = 10;
if (x == 5) {
   ...
}
```

if-else

- Sintaxe

```
if (expressão booleana)
   instrução_simples;

if (expressão booleana) {
   instruções
}
```

```
if (expressão booleana) {
   instruções
} else if (expressão booleana) {
   instruções
} else {
   instruções
}
```

- Exemplo

```
if ( ano < 0) {
   System.out.println("Não é um ano!");
} else if ( ano%4==0 && ano%400==0 && ano%100!=0 ) {
   System.out.println("É bissexto!");
} else {
   System.out.println("Não é bissexto!");
}
```

return

- *A palavra-chave return tem duas finalidades*
  - *especifica o que um método irá retornar (se o método não tiver sido declarado com tipo de retorno void)*
  - *causa o retorno imediato à linha de controle imediatamente posterior à chamada do método*
- *Exemplos de sintaxe:*

```
boolean método() {
    if (condição) {
        instrução;
        return true;
    }
    resto do método
}
```

```
void método() {
    if (condição) {
        instrução;
        return;
    }
    mais coisas...
}
```

Este exemplo funciona como um if com else:

while e do-while

- Sintaxe

```
while (expresão booleana )
{
   instruções;
}
```

```
do
{
   instruções;
} while (expresão booleana );
```

- Exemplos

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

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

```
while ( true ) {
   if (obj.z == 0) {
     break;
   }
}
```

loop infinito!

for

- Sintaxe

```
for ( inicialização ;
      expressões booleanas;
      passo da repetição )
{
   instruções;
}
```

```
for ( inicialização ;
      expressões booleanas;
      passo da repetição )

  instrução_simples;
```

- Exemplos

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

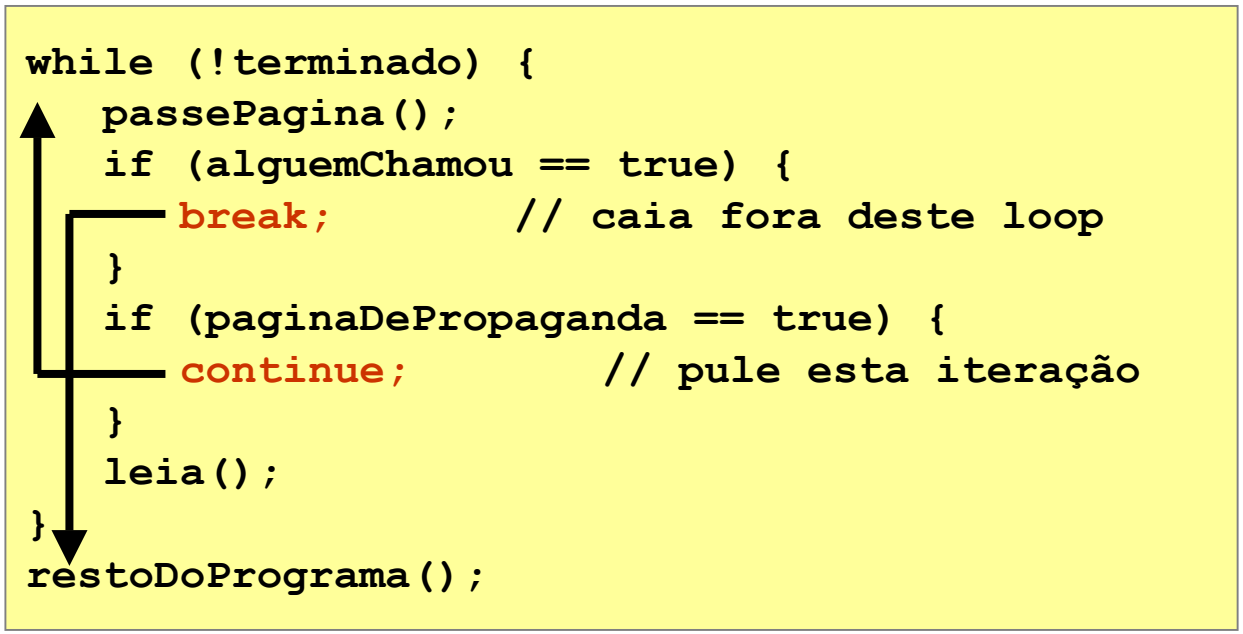
```
for ( int x = 0, int y = 25;
      x < 10 && (y % 2 == 0);
      x++, y = y - 1 ) {
   System.out.println (x + y);
}
```

```
for ( ; ; ) {
  if (obj.z == 0) {
    break;
  }
}
```

loop infinito!

break e continue

- **break**: interrompe a execução do bloco de repetição.
  - Continua com a próxima instrução, logo após o bloco.
- **continue**: interrompe a execução da iteração
  - Testa a condição e reinicia o bloco com a próxima iteração.



break e continue com rótulos

- ***break*** *e* ***continue*** *sempre atuam sobre o bloco de repetição onde são chamados*
- *Em blocos de repetição contidos em outros blocos, pode-se usar* ***rótulos*** *para fazer break e continue atuarem em blocos externos*
- *Os rótulos só podem ser usados antes de do, while e for*
- *As chamadas só podem ocorrer dentro de blocos de repetição. Exemplo:*

```
revista: while (!terminado) {
   for (int i = 10; i < 100; i += 10) {
      passePagina();
      if (textoChato) {
         break revista;
      }
   }
   maisInstrucoes();
}
restoDoPrograma();
```

break sem rótulo quebraria aqui!

- Sintaxe:

```
ident: do {...}
       ou
ident: while () {...}
       ou
ident: for () { ...}
```

switch (case)

- Sintaxe

qualquer expressão que resulte em valor inteiro (incl. char)

```
switch(seletor_inteiro) {
  case valor_inteiro_1 :
    instruções;
    break;
  case valor_inteiro_2 :
    instruções;
    break;
   ...
  default:
    instruções;
}
```

uma constante inteira (inclui char)

- Exemplo

char letra;

```
switch(letra) {
  case 'A' :
    System.out.println("A");
    break;
  case 'B' :
    System.out.println("B");
    break;
   ...
  default:
    System.out.println("?");
}
```

Exercícios

- *1. Escreva um programa que leia um número da linha de comando e imprima o quadrado de todos os números entre 1 e o número passado.*
  - *Para converter de String para int, use:*
    int numero = Integer.parseInt("10");
- *2. Use o JOptionPane (veja documentação) e repita o exercício anterior recebendo os dados através da janela de entrada de dados*
  - *use JOptionPane.showInputDialog(string) de javax.swing para ler entrada de dados*
  - *use JOptionPane.showMessageDialog(null, msg) para exibir a saída de dados*