Programação Orientada por Objectos com Java

Ademar Aguiar
www.fe.up.pt/~aaguiar
ademar.aguiar@fe.up.pt

Universidade do Porto
Faculdade de Engenharia
FEUP

Introdução ao Java

Objectivos

Ser capaz de:

- Identificar os elementos principais do Java
- Descrever a Java Virtual Machine (JVM)
- Comparar a utilização do Java para a construção de *applets* e de aplicações
- Identificar os componentes principais do Java Development Kit (JDK)
- Descrever as opções de instalação do Java (deployment)

O que é o Java?

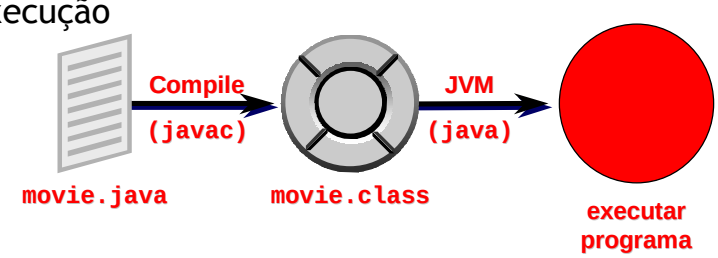
- Concebido pela Sun para a electrónica de consumo, mas rapidamente alcançou a WWW
- Uma linguagem orientada por objectos e uma biblioteca de classes
- Utiliza uma máquina virtual para a execução de programas

Vantagens Principais do Java

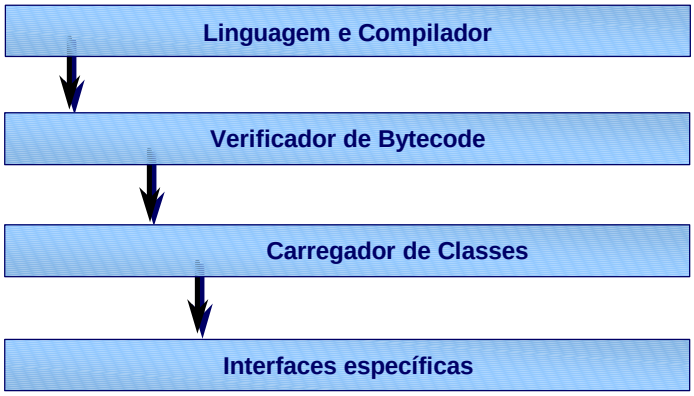
- Orientado por objectos
- Interpretado e independente da plataforma
- Dinâmico e distribuído
- Multithreaded
- Robustez e segurança

Independente da Plataforma

- O código Java é armazenada num ficheiro .java
- Um programa .java é compilada para ficheiros .class
- Bytecodes são interpretados em tempo de execução



Ambiente de Segurança do Java



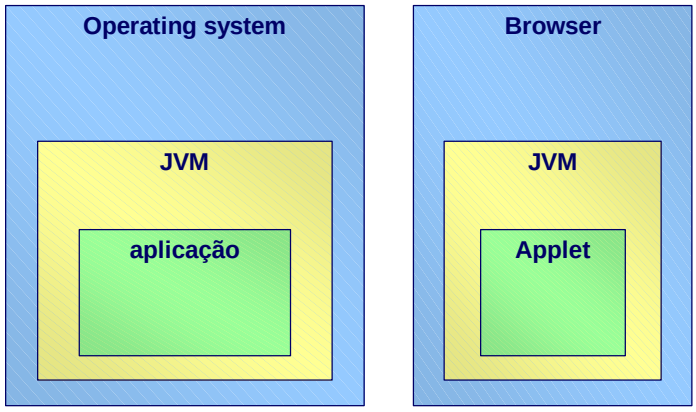
Applets Java

- A forma mais comum de utilização do Java, inicialmente
- Vocacionada para utilização em páginas HTML
- Pode incluir conteúdos activos (forms, audio, imagens, vídeo)
- Aparece num *browser* e pode comunicar com o servidor

Aplicações Java

- Instalação no lado do cliente:
  - JVM corre em aplicações autónomas
  - Não necessita de carregar classes pela rede
- Instalação do lado do servidor:
  - Pode servir múltiplos clientes a partir de uma mesma origem
  - Encaixa bem com modelos multi-camada para computação na Internet

JVM - Java Virtual Machine



Como funciona a JVM

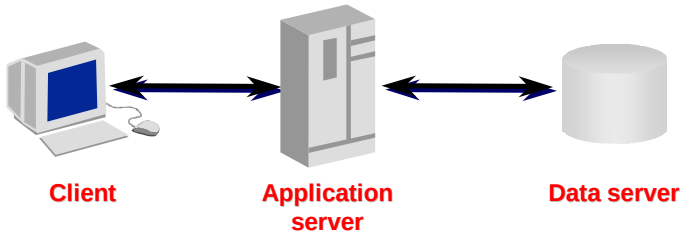
- O "JVM class loader" carrega todas as classes necessárias.
- O "JVM verifier" verifica os bytecodes ilegais.
- O gestor de memória da JVM liberta memória de volta ao sistema operativo.

Compiladores Just-in-Time (JIT)

- Melhoram a performance
- São úteis se os mesmos *bytecodes* forem executados repetidas vezes
- Traduz bytecodes para instruções nativas
- Optimizam código repetitivo, tais como ciclos

Java e Computação na Internet

- A computação na Internet podem consistir em três diferentes camadas:



- Java pode ser usada em todas estas camadas.

Resumo

- O código Java é compilado em bytecodes independentes da plataforma.
- Os *bytecodes* são interpretados por uma JVM.
- As *applets* correm num browser no cliente.
- As aplicações Java são executadas de forma autónoma tanto no cliente como no servidor.

Conceitos Básicos do Java

Objectivos

Ser capaz de:

- Identificar os elementos principais do Java
- Identificar a síntaxe básica do Java
- Descrever ficheiros .java e .class

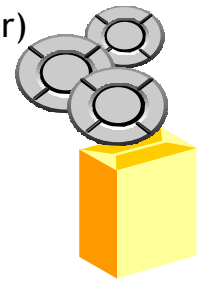
Tópicos

- Componentes Java
- Convenções
- Classes, objectos e métodos
- Utilização de Javadoc
- Compilar e executar programas Java

JDK - Java Development Kit

O JDK da Sun fornece:

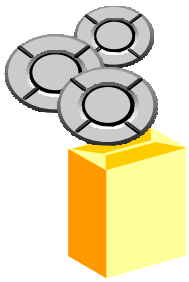
- Compilador (javac)
- Visualizador de *applets* (appletviewer)
- Interpretador de *bytecode* (java)
- Gerador de documentação (javadoc)



JDK - Java Development Kit

O JDK da Sun fornece pacotes standard para:

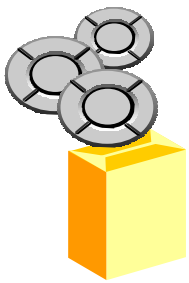
- linguagem
- Sistema de janelas
- Controlo de *Applets*
- Entrada/Saída
- Comunicação em Rede



JDK - Java Development Kit

O JDK da Sun fornece suporte de documentação para:

- Comentários
  - Implementação
  - Documentação
- Gerador de Documentação



Convenções de Nomes

As convenções incluem:

- Nomes de ficheiros
  - Customer.java, RentalItem.java
- Nomes de Classes
  - Customer, RentalItem, InventoryItem
- Nomes de Métodos
  - getCustomerName(), setRentalItemPrice()

Convenções de Nomes...

- Standard para variáveis
  - customerName, customerCreditLimit
- Standard para constantes
  - MIN_WIDTH, MAX_NUMBER_OF_ITEMS
- Utilização de caracteres maiúsculos e minúsculos
- Números e caracteres especiais

Definição de Classes

- A definição de classes normalmente inclui:
  - Modificador de acesso: *public*, *private*
  - A palavra-chave *class*
  - Campos das instâncias
  - Constructores
  - Métodos das instâncias
  - Campos da classe
  - Métodos da classe

Definição de Classes...

```
public class Customer {
 // Instance variáveis
 String customerName;
 String customerPostalCode;
 float customerAmountDue;
 …
 // Instance métodos
 float getAmountDue (String cust) {
  …
 }
 …
}
```

Declaração

Variável de Instância

Método da Instância

Definição de Métodos

- Sempre dentro de uma classe
- Especificam:
  - Modificador de acesso
  - Palavra-chave *static*
  - Argumentos
  - Tipo de retorno

```
[access-modifiers] [static] <método-name>
<return-tipo> ([arguments]) <java code block>
```


Definição de Métodos

```
float getAmountDue (String cust) {
 // método variáveis
 int numberOfDays;
 float due;
 float lateCharge = 1.50;
 String customerName;
// método body
 numberOfDays = this.getOverDueDays();
 due = numberOfDays * lateCharge;
 customerName = getCustomerName(cust);
 return due;
}
```

Declaração

Variáveis de método

Instruções de método

Retorno

Regras para Declaração de Variáveis e Constantes

- Devem ser declaradas antes de ser utilizadas
- Uma declaração por linha
- No início de um bloco de código
- O bloco de código define o âmbito
- Inicialização

Regras para Declaração de Variáveis e Constantes

```
float getAmountDue (String cust) {
 float due = 0;
 int numberOfDays = 0;
 float lateFee = 1.50;
 {int tempCount = 1; // new code block
   due = numberOfDays * lateFee;
   tempCount++;
   …
 }                   // end code block
 return due;
}
```

Variáveis de método

Variáveis temporárias

Regras para a criação de blocos de código

- Agrupar todas as declarações de classe.
- Agrupar todas as declarações de métodos.
- Agrupar outros segmentos de código relacionado entre si.

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


Regras para a criação de instruções

- As instruções terminam sempre com um ponto-e-vírgula (;)
- Instruções compostas são definidas dentro de chavetas { }.
- Utilizar chavetas para instruções de controlo.

Compilar e Executar uma Aplicação Java

- Para compilar um ficheiro .java:

```
prompt> javac SayHello.java
… compiler output …
```

- Para executar um ficheiro .class:

```
prompt> java SayHello
Hello world
prompt>
```

- Atenção às maiúsculas e minúsculas!

Resumo

- O JDK fornece as ferramentas Java essenciais.
- O JDK fornece um conjunto valioso de classes e métodos pré-definidos.
- Os programas Java são constituídos por classes, objectos, e métodos.
- A adopção de normas de programação facilita a leitura e reutilização de código.

Exemplos Práticos

- Explorar um programa fonte em Java
- Inspeccionar classes, métodos e variáveis
- Compilar e executar uma aplicação

Tipos de Dados e Operadores

Objectivos

Ser capaz de:

- Descrever os tipos de dados primitivos
- Declarar e inicializar variáveis primitivas
- Utilizar operadores para manipular o valor de uma variável primitiva

Tópicos

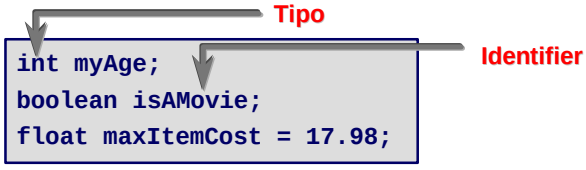
- O Java oferece primitivas para os tipos de dados básicos.
- As primitivas são a fundação para armazenar e utilizar informação.
- Declarar e inicializar primitivas é a bse da construção de tipos definidos pelo utilizador.

Tópicos

- Os operadores manipulam dados e objectos.
- Aceitam um ou mais argumentos e produzem um valor.
- Java oferece 44 operadores diferentes.
- Alguns operadores alteram o valor do operando.

Variáveis

- Uma variável é a unidade básica de armazenamento.
- As variáveis devem ser declaradas explicitamente.
- Cada variável tem um tipo, um identificador, e um âmbito.
- As variáveis podem ser inicializadas.



Nomes de Variáveis

- Os nomes das variáveis devem começar por uma letra do alfabeto, um *underscore*, ou um $.
- Os outros caracteres podem incluir digitos.
- Deve-se utilizar nomes elucidativos para as variáveis; por exemplo,
  customerFirstName, ageNextBirthday.

```
a            item_Cost
itemCost     _itemCost
item$Cost    itemCost2
```

```
item#Cost    item-Cost
item*Cost    abstract
2itemCost
```


Palavras Reservadas

```
boolean
byte
char
double
float
int
long
short
void
```

```
abstract
final
native
private
protected
public
static
synchronized
transient
volatile
```

```
false
null
true
```

```
break
case
catch
continue
default
do
else
finally
for
if
return
switch
throw
try
while
```

```
class
extends
implements
interface
throws
```

```
import
package
```

```
instanceof
new
super
this
```

Tipos de Variáveis

- Oito tipos de dados primitivos:
  - Seis tipos numéricos
  - Tipo *char*, para caracteres
  - Tipo Booleano, para valores verdadeiro ou falso
- Tipos definidos pelo utilizador
  - Classes
  - Interfaces
  - Arrays

Tipos de Dados Primitivos

Integer	Floating Point	Character	True False
byte short int long	float double	char	boolean
1,2,3,42 07 0xff	3.0 .3337 4.022E23	'a' '\141' '\u0061' '\n'	true false

Declaração de Variáveis

- A forma básica de declaração de uma variável:

tipo *identifier* [= valor]

```
public static void main(String[] args) {
    int itemsRented;
    float itemCost;
    int i, j, k;
    double interestRate;
}
```

- As variáveis podem ser inicializadas quando declaradas.

Declaração de Variáveis

- As variáveis locais estão contidas apenas num método ou bloco de código.
- As variáveis locais devem ser inicializadas antes de ser usadas.

```
class Rental {
  private int instVar;     // instance variável
  public void addItem() {
    float itemCost = 3.50; // local variável
    int numOfDays  = 3;    // local variável
  }
}
```

Literais Números

Literais Inteiros

```
0  1  42  -23795          (decimal)
02  077  0123             (octal)
0x0  0x2a  0X1FF          (hex)
365L  077L  0x1000L       (long)
```

Literais Floating-point

```
1.0  4.2  .47
1.22e19  4.61E-9
6.2f  6.21F
```


Literais não-Números

Literais Booleanos

```
true  false
```

Literais Caracter

```
'a'  '\n'  '\t'  '\077'
'\u006F'
```

Literais String

```
"Hello, world\n"
```

Exercício: Declaração de variáveis

- Encontrar os erros no código abaixo e corrigi-los.

```
1  byte sizeof = 200;
2  short mom = 43;
3  short hello mom;
4  int big = sizeof * sizeof * sizeof;
5  long bigger = big + big + big     // ouch
6  double old = 78.0;
7  double new = 0.1;
8  boolean consequence = true;
9  boolean max = big > bigger;
10 char maine = "New England state";
11 char ming = 'd';
```


Operadores

Cinco tipos de operadores:

- Atribuição
- Aritméticos
- Manipulação de bits
- Relacionais
- Booleanos

Operador de Atribuição

- A expressão da direita é atribuída à variável da esquerda:

```
int var1 = 0, var2 = 0;
var1 = 50;        // var1 now equals 50
var2 = var1 + 10; // var2 now equals 60
```

- A expressão da direita é sempre avaliada antes da atribuição.
- As atribuições podem ser agrupadas:

```
var1 = var2 = var3 = 50;
```


Operadores Aritméticos

- Realizam operações aritméticas básicas
- Operam sobre variáveis e literais numericos

```
int a, b, c, d;
a = 2 + 2;    // addition
b = a * 3;    // multiplication
c = b - 2;    // subtraction
d = b / 2;    // division
e = b % 2;   // returns the remainder of division
```

Operadores Aritméticos...

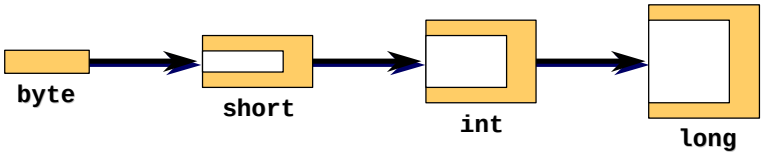
- A maioria das operações resultam num int ou long:

```
byte b1 = 1, b2 = 2, b3;
b3 = b1 + b2;     // error: result is an int
                  // b3 is byte
```

- Valores byte, char, e short são promovidos a int antes da operação.
- Se algum argumento for long, o outro é promovido a long, e o resultado é long.

Conversões e Casts

- O Java converte automaticamente valores de um tipo numérico para outro tipo maior.



- O Java não faz automaticamente o "downcast."

Incrementar e Decrementar

- O operador ++ incrementa 1 unidade:

```
int var1 = 3;
var1++;          // var1 now equals 4
```

- O operador ++ pode ser usado de duas maneiras:

```
int var1 = 3, var2 = 0;
var2 = ++var1;   // Prefix:  Increment var1 first,
                 //          then assign to var2.
var2 = var1++;   // Postfix: Assign to var2 first,
                 //          then increment var1.
```

- O operador -- decrementa 1 unidade.

Comparações

- Operadores relacionais e de igualdade:

>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to
==	equal to
!=	not equal to

```
int var1 = 7, var2 = 13;
boolean res = true;
res = (var1 == var2);     // res now equals false
res = (var2 > var1);      // res now equals true
```

Operadores Lógicos

- Os resultados de expressões Booleanas podem ser combinados usando operadores lógicos:

&& & || | ^ !	e (with / without short-circuit evaluation) or (with / without short-circuit evaluation) exclusive or not

```
int var0 = 0, var1 = 1, var2 = 2;
boolean res = true;
res = (var2 > var1) & (var0 == 3);   // now false
res = !res;                          // now true
```


Atribuição Composta

- O operador de atribuição pode ser combinado com qualquer operador binário convencional:

```
double total=0, num = 1;
double percentage = .50;
…
total  = total + num;    // total is now  1
total += num;            // total is now  2
total -= num;            // total is now  1
total *= percentage;     // total is now .5
```

Precedência de Operadores

Order	Operadores	Comments	Assoc.
1	++ -- + - ~ !(tipo)	Unary operadores	R
2	* / %	Multiply, divide, remainder	L
3	+ - +	Add, subtract, add string	L
4	<< >> >>>	Shift (>>> is zero-fill shift)	L
5	< > <= >= instanceof	Relational, tipo compare	L
6	== !=	Equality	L
7	&	Bit/logical e	L
8	^	Bit/logical exclusive OR	L
9	|	Bit/logical inclusive OR	L
10	&&	Logical e	L
11	||	Logical OR	L
12	?:	Conditional operador	R
13	= op=	Assignment operadores	R

Precedências

- A precedência de um operador determina a ordem pela qual os operadores são executados:

```
int var1 = 0;
var1 = 2 + 3 * 4;    // var1 now equals 14
```

- Operadores com a mesma precedência são executados da esquerda para a direita (ver nota):

```
int var1 = 0;
var1 = 12 - 6 + 3;   // var1 now equals 9
```

- Os parentesis permitem alterar a ordem definida.

Concatenação de String's

- O operador + cria e concatena strings:

```
String name = "Jane ";
String lastName = "Hathaway";
String fullName;
name = name + lastName;    // name is now
                           //"Jane Hathaway"
                           //    OR
name += lastName;          // same result
fullName = name;
```


Resumo

- O Java tem oito tipos de dados primitivos.
- Uma variável deve ser declarada antes de ser usada.
- O Java dispõe de um bom conjunto de operadores.
- Casting explícitos podem ser necessários se utilizar tipos de dados menores do que int.
- Os operadores + e += podem ser usados para criar e concatenar strings.

Exercício Prático

- Declarar e inicializar variáveis
- Utilizar vários operadores para calcular novos valores
- Mostrar os resultados no écran

Instruções de Controlo de Fluxo

Objectivos

Ser capaz de:

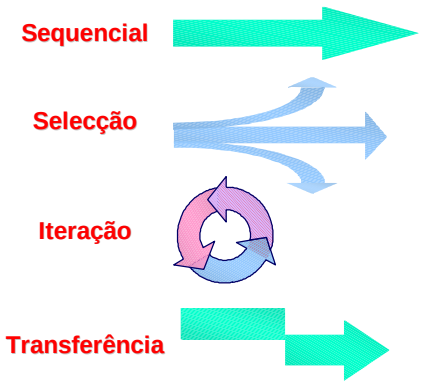
- Utilizar construções para tomar decisões
- Realizar ciclos de operações

Tópicos

- O código por defeito executa sequencialmente.
- Código mais complexo exige uma execução condicional.
- Existem instruções que necessitam de ser executadas repetidamente.
- O Java dispõe de mecanismos de controlo standard.

Tipos Básicos de Controlo

- Controlo de fluxo pode ser categorizado em quatro tipos:



Controlo de Fluxo em Java

- Agrupar instruções utilizando chavetas para formar uma instrução composta, i.e. um bloco.
- Cada bloco é executado como uma única instrução dentro da estrutura de controlo de fluxo.

```
{
  boolean finished = true;
  System.out.println("i = " + i);
  i++;
}
```

if ... else

Forma geral:

```
if ( boolean_expr )
    statement1;
[else]
    statement2;
```

Exemplos:

```
if (i % 2 == 0)
   System.out.println("Even");
else
   System.out.println("Odd");
```

```
…
if (i % 2 == 0) {
    System.out.println(i);
    System.out.println(" is even");
}
```


if...if...if...else if...else

```
if (speed >= 25)
    if (speed > 65)
        System.out.println("Speed over 65");
    else
        System.out.println("Speed over 25");
else
    System.out.println("Speed under 25");
```

```
if (speed > 65)
  System.out.println("Speed over 65");
else if (speed >= 25)
        System.out.println("Speed over 25");
     else
        System.out.println("Speed under 25");
```

Operador Condicional (? :)

- (boolean_expr ? expr1 : expr2)

```
boolean_expr ? expr1 : expr2
```

- É uma alternativa útil ao if…else:
- Se boolean_expr=true, o resultado é expr1, senão o resultado é expr2:

```
int val1 = 120, val2 = 0;
int highest;
highest = (val1 > val2) ? 100 : 200;
System.out.println("Highest value is " + highest);
```


Exercício: Descubra os Erros!

1
```
int x = 3, y = 5;
if (x >= 0)
  if (y < x)
    System.out.println("y is less than x");
else
  System.out.println("x is negative");
```

2
```
int x = 7;
if (x = 0)
  System.out.println("x is zero");
```

3
```
int x = 15, y = 24;
if ( x % 2 == 0  &&  y % 2 == 0 );
  System.out.println("x and y are even");
```

switch...case

- É útil para seleccionar um entre vários valores inteiros alternativos

```
switch ( integer_expr ) {

    case constant_expr1:
        statement1;
        break;
    case constant_expr2:
        statement2;
        break;
    [default:
        statement3;
        break;]
}
```


switch...case

- As etiquetas de case devem ser constantes.
- Utilizar break para saltar fora do switch.
- Dar sempre uma alternativa default.

```
switch (choice) {
  case 37:
     System.out.println("Coffee?");
     break;

  case 45:
     System.out.println("Tea?");
     break;

  default:
     System.out.println("???");
     break;
}
```

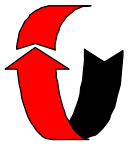
Ciclos

- Em Java existem três tipos de ciclos:
  - while
  - do…while
  - for
- Todos os ciclos têm quatro partes:
  - Inicialização
  - Iteração
  - Corpo
  - Terminação

while...

- O while é o mais simples de todos os ciclos:
- Exemplo:

```
while ( boolean_expr )
    statement;
```

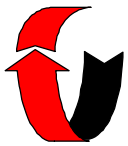


```
int i = 0;
while (i < 10) {
   System.out.println("i = " + i);
   i++;
}
```

do…while

- Os ciclos do…while têm o teste no fim do ciclo:
- Exemplo:

```
do
    statement;
while ( termination );
```



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


for...

- Os ciclos for são os mais comuns:

```
for ( initialization; termination; iteration )
    statement;
```

- Exemplo:

```
for (i = 0; i < 10; i++)
    System.out.println(i);
```

- Qual o ciclo *while* equivalente?

for...

- Podem ser declaradas variáveis na parte de inicialização do ciclo for:

```
for (int i = 0; i < 10; i++)
    System.out.println("i = " + i);
```

- As partes de inicialização e iteração podem consistir de uma lista de expressões separadas por vírgulas:

```
for (int i = 0, j = 10; i < j; i++, j--) {
    System.out.println("i = " + i);
    System.out.println("j = " + j);
}
```


Exercício: Descubra os Erros!

1

```
int x = 10;
while (x > 0);
  System.out.println(x--);
System.out.println("We have lift off!");
```

2

```
int x = 10;
while (x > 0)
  System.out.println("x is " + x);
  x--;
```

3

```
int sum = 0;
for (; i < 10; sum += i++);
System.out.println("Sum is " + sum);
```

break

- Interrompe um ciclo ou uma instrução switch:
- Transfere o controlo para a primeira instrução depois do corpo do ciclo ou instrução switch
- Pode simplificar o código

```
…
while (age <= 65) {
    balance = (balance+payment) * (1 + interest));
    if (balance >= 250000)
        break;
    age++;
}
…
```


continue

- Apenas pode ser usado em ciclos
- Abandona a iteração em curso e salta para a próxima iteração do ciclo

```
…
for (int year = 2000; year < 2099; year++) {
    if ((year % 100 == 0) && (year % 400 != 0))
        continue;
    if (year % 4 == 0)
        System.out.println(year);
}
…
```

labeled break, continue

- Pode ser usado para saltar fora de ciclos encaixados, ou continuar um ciclo exterior ao ciclo corrente

```
outer_loop:
for (int i = 0; i < 10; i++) {
    for (int j = 0; j < 5; j++) {
        System.out.println(i, j);
        if (i + j > 7)
            break outer_loop;
    }
}
…
```


Resumo

- A intrução if...else é a forma principal de implementar decisões.
- Java também dispõe de instrução switch.
- Java oferece três instruções de ciclos:
  - while
  - do…while
  - for
- A utilização de break e continue deve ser feita criteriosamente.

Exercícios Práticos

- Realizar testes utilizando instruções if…else
- Utilizar o operador condicional ternário ?:
- Utilizar ciclos while e for paa realizar operações iterativas
- Utilizar break para abandonar um ciclo
- Utilizar os operadores &&, ||, e ! Expressões booleanas

Fim