



Programação OO

1ª parte

Prof. Douglas Oliveira

douglas.oliveira@infnet.br

- **Procedural**: baseado em rotinas/procedimentos
 - ✓ Exemplos: Pascal, C, Fortran
- **Funcional**: baseado em funções
 - ✓ Exemplos: Lisp, Haskell
- **Lógico**: baseado na lógica dos predicados (fatos/regras)
 - ✓ Exemplo : Prolog
- **Orientado a Objetos**: baseado em objetos e classes
 - ✓ Exemplos: SmallTalk, C++, **Java**, C#



Características da Linguagem Java

- ❑ Desenvolvida na década de 90 por James Gosling da Sun Microsystems.
- ❑ Orientada a Objetos com uma grande diversidade de bibliotecas de classes disponível.
- ❑ Independente de plataforma:
 - ✓ *write once, run everywhere*
- ❑ Segurança
 - ✓ Mecanismos para sistemas livres de vírus e criptografia.

Características da Linguagem Java

□ Simplicidade

- ✓ Sintaxe dos comandos básicos segue o padrão do C.
- ✓ Sintaxe relacionada à OO bem mais simples que o C++.

□ Internacionalização

- ✓ UNICODE: padrão que permite manipular textos de qualquer sistema de escrita.

□ Robustez

- ✓ Tratamento de exceções
- ✓ JVM impede que uma aplicação mal comportada paralise o sistema.

Características da Linguagem Java

□ Distribuída e multitarefa

- ✓ Os programas podem utilizar recursos da rede com a mesma facilidade que acessam arquivos locais.
- ✓ Trabalha com diversos protocolos (TCP/IP, HTTP, FTP,...)
- ✓ Execução simultânea de múltiplas *threads*.

□ Desempenho

- ✓ Nas primeiras versões, Java era bem mais lenta que as linguagens compiladas puras (C, C++).
- ✓ Hoje, os problemas de desempenho são resolvidos com compilação *just-in-time*.



□ Applications

- ✓ Semelhantes aos programas tradicionais (desktop, cliente/servidor).

□ Applets

- ✓ Programa que pode ser incluído em uma página HTML.
- ✓ Programa é carregado de um servidor e executado localmente dentro de um browser.
- ✓ É uma tecnologia antiga que caiu em desuso.

□ Servlets

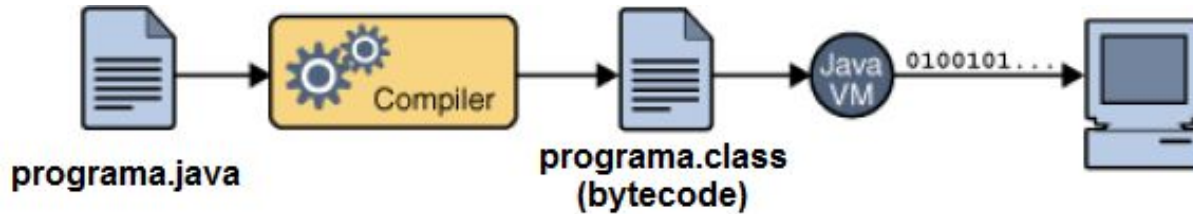
- ✓ Programas que executam em um servidor web e são acessados via browser.
- ✓ São a base de outras tecnologias como JSP, Struts, JSF, etc.

- O termo **plataforma** normalmente é usada para designar um conjunto:

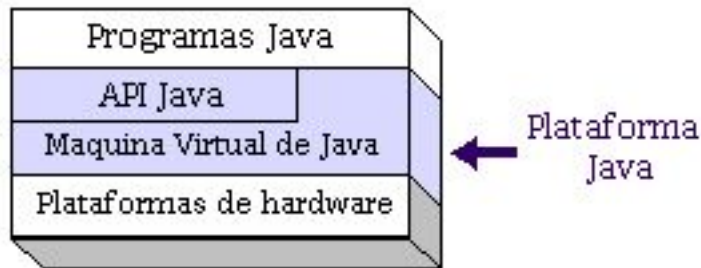
Hardware + Sistema Operacional

- A **plataforma Java** é definida apenas em **software** e possui dois componentes:
 - ✓ Máquina Virtual Java (JVM – Java Virtual Machine)
 - ✓ Conjunto de bibliotecas que disponibilizam funções comuns (API Java)

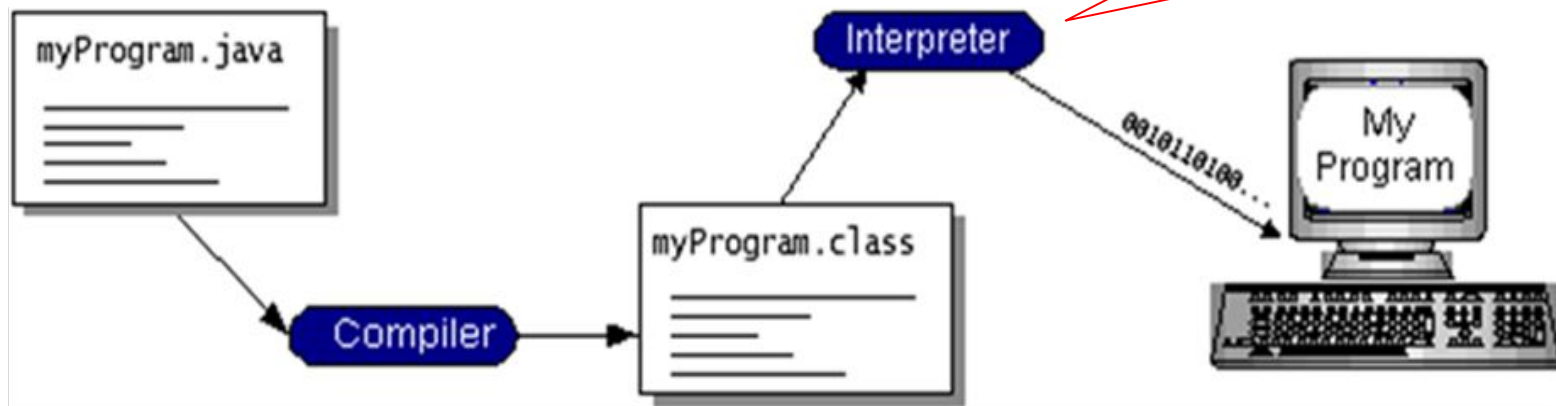
- Diferentemente das linguagens convencionais, que são compiladas para **código nativo**, a linguagem Java é compilada para **bytecode** (gerando .class ou .jar) que é executado por uma máquina virtual Java (JVM – *Java Virtual Machine*).



- Com a JVM existe uma **camada extra** entre o **Sistema Operacional** (Windows, Linux, Mac OS, Android, iOS, etc.) e a **aplicação**.
- Essa camada permite que a aplicação seja executada em diversas plataformas sem a necessidade de alterar a aplicação (**portabilidade**).



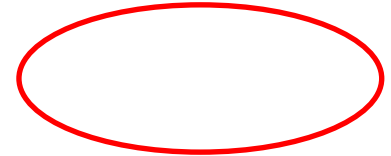
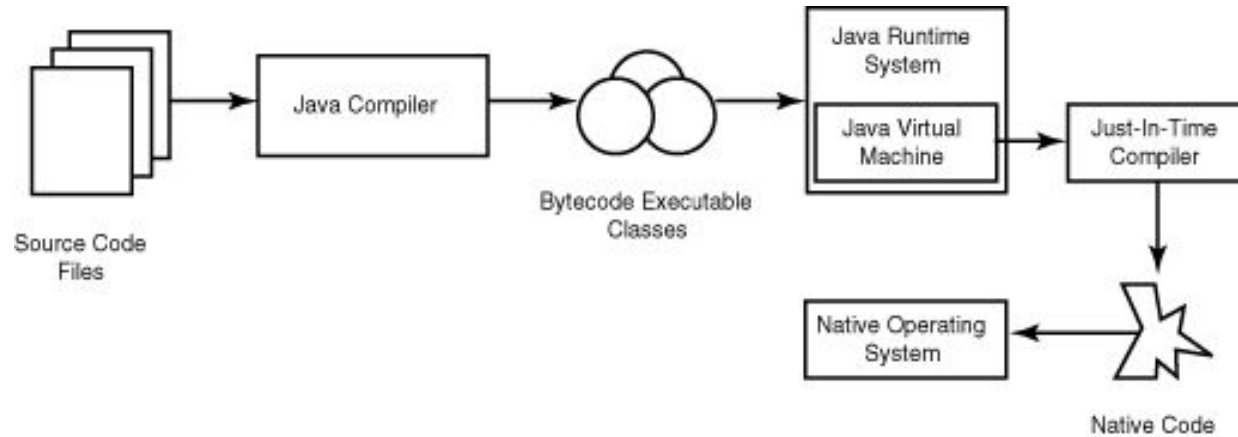
Modelo inicial



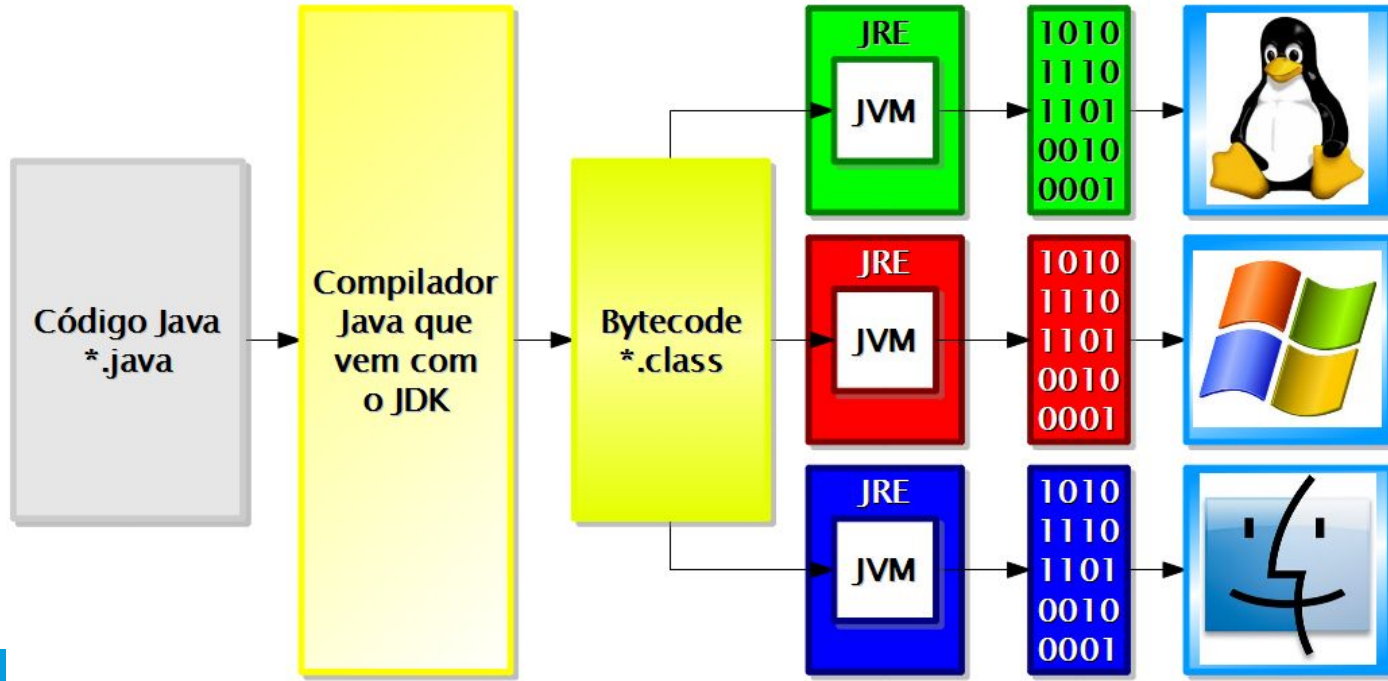
bytecode
(independente de plataforma)

- ❑ O modelo inicial tinha um **desempenho muito baixo** por causa da interpretação comando a comando do *bytecode*.
- ❑ Nos últimos anos, a interpretação do *bytecode* foi substituída por outra compilação, que transforma o *bytecode* em código binário nativo.
- ❑ Essa segunda compilação ocorre quando o programa é executado (compilador JIT , isto é, *just-in-time*) e nem o programador nem o usuário precisam se preocupar com ela.
- ❑ Com essa segunda compilação, o programa é executado a partir do código binário nativo, o que leva a um desempenho muito próximo das linguagens puramente compiladas como o C/C++.

Modelo atual



□ Modelo de compilação e execução:



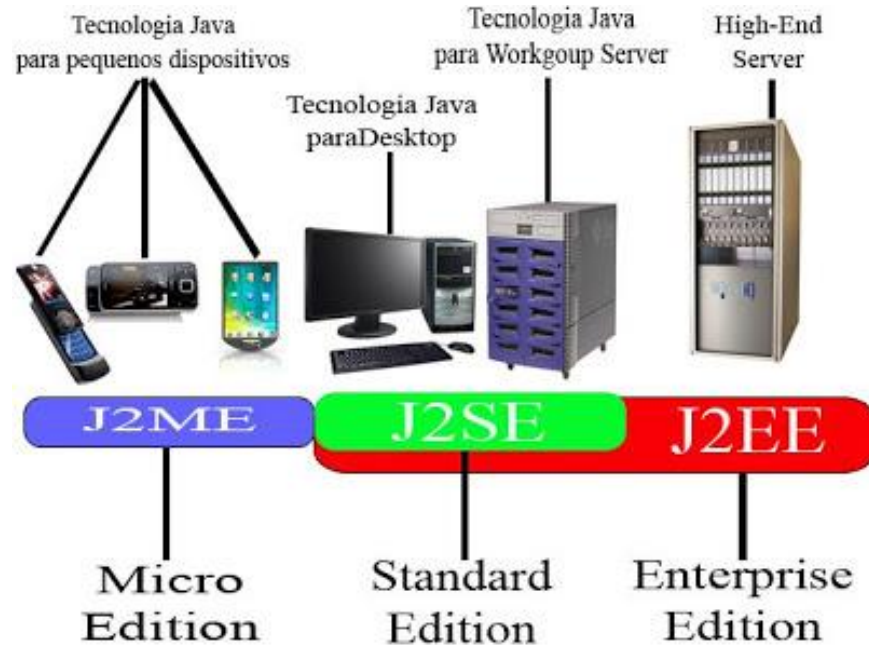
□ Java Development Kit (JDK):

- ✓ Coleção de programas para, dentre outras tarefas, **compilar** e **executar** aplicações Java.
- ✓ Exemplo:
 - javac (compilador Java)
 - javadoc (utilitário para documentação)
 - java (máquina virtual)
 - etc.



- Java Runtime Environment (JRE):
 - ✓ Kit com todos os programas necessários para **executar** aplicações Java.
 - ✓ Faz parte do JDK.
 - ✓ Pode ser instalado separadamente.

- A plataforma Java é composta por 3 plataformas:



- A plataforma Java é composta por 3 plataformas:
- ✓ **J2SE ou Java SE** (*Java Standard Edition*): é a base da plataforma Java e inclui o ambiente de execução e as bibliotecas comuns.
- ✓ **J2EE ou Java EE** (*Java Enterprise Edition*): versão voltada para o desenvolvimento de aplicações distribuídas, multicamadas e aplicações web.
- ✓ **J2ME ou Java ME** (*Java Micro Edition*): versão voltada para o desenvolvimento de aplicações móveis ou embarcadas.



□ IDEs (*Integrated Development Environment*) para Java:

- ✓ Netbeans
- ✓ Eclipse
- ✓ IntelliJ
- ✓ BlueJ
- ✓ JCreator
- ✓ e várias outras.

- ❑ Todo programa Java deve ter um método **main** que determina o início da execução do programa.
- ❑ Entretanto, por ser uma linguagem Orientada a Objetos, os métodos devem vir, obrigatoriamente, dentro de uma classe.
- ❑ Assim, um programa mínimo em Java deve ter uma **classe** e, dentro dela, um método chamado **main**.

```
/* Arquivo: AloMundo.java */  
public class AloMundo {  
    public static void main(String [] args)  
    {  
        System.out.println("Alo mundo!");  
    } // fim do método main  
  
} // fim da classe AloMundo
```

/* Arquivo: AloMundo.java */

```
public class AloMundo {  
    public static void main(String [] args)  
    {  
        System.out.println("Alo mundo!");  
    } // fim do método main  
} // fim da classe AloMundo
```

Os comentários seguem o mesmo padrão do C/C++:

/* e */ para comentários multilinhas

// para comentários em uma linha

```
/* Arquivo: AloMundo.java */  
public class AloMundo {  
    public static void main(String [] args)  
    {  
        System.out.println("Alo mundo!");  
    } // fim do método main  
} // fim da classe AloMundo
```

Todo código Java deve ficar dentro de uma classe.

Para definir uma classe usamos a palavra **class** seguida do **nome da classe**.

Os delimitadores **{** e **}** definem onde a classe inicia e onde ela termina.

Primeiro Programa

```
/* Arquivo: AloMundo.java */  
public class AloMundo {  
    public static void main(String [] args)  
    {  
        System.out.println("Alo mundo!");  
    } // fim do método main  
  
} // fim da classe AloMundo
```

Repare que, em um determinado arquivo, devemos declarar somente **uma** classe.

O nome do arquivo deve ter o **mesmo nome** da classe com a extensão **.java**.

Primeiro Programa

```
/* Arquivo: AloMundo.java */
```

```
public class AloMundo {
```

```
    public static void main(String [] args)
```

```
{
```

```
    System.out.println("Alo mundo!");
```

```
} // fim do método main
```

```
} // fim da classe AloMundo
```

Dentro de uma classe podemos definir **vários** métodos.

Devemos definir, um e somente um, método **main**, que indica **por onde a execução se iniciará**.


```
/* Arquivo: AloMundo.java */  
public class AloMundo {  
    public static void main(String [] args)  
    {  
        System.out.println("Alo mundo!");  
    } // fim do método main  
  
} // fim da classe AloMundo
```

Ao longo do curso vamos entender o significado das palavras **public** e **static**.

Primeiro Programa

```
/* Arquivo: AloMundo.java */  
public class AloMundo {  
    public static void main(String [] args)  
    {  
        System.out.println("Alo mundo!");  
    } // fim do método main  
  
} // fim da classe AloMundo
```

Da mesma forma como fazemos nas funções em C/C++, antes do nome do método definimos o seu tipo de retorno.
O tipo de retorno do método **main** é sempre **void**, ou seja, não retorna nada.

```
/* Arquivo: AloMundo.java */  
public class AloMundo {  
    public static void main(String [] args)  
    {  
        System.out.println("Alo mundo!");  
    } // fim do método main  
  
} // fim da classe AloMundo
```

O parâmetro **args** indica os argumentos do programa.

Os argumentos são os dados passados na linha de execução do programa (linha de comando)

```
/* Arquivo: AloMundo.java */  
public class AloMundo {  
    public static void main(String [] args)  
    {  
        System.out.println("Alo mundo!");  
    } // fim do método main  
  
} // fim da classe AloMundo
```

Assim como o C/C++ tem suas **funções** organizadas em um conjunto de **bibliotecas**, o Java possui uma série de **métodos** organizados em **classes**.

Nesse caso temos a classe **System**, que fornece, dentre outras coisas, os arquivos de entrada e saída padrão: **in** e **out**.

```
/* Arquivo: AloMundo.java */  
public class AloMundo {  
    public static void main(String [] args)  
    {  
        System.out.println("Alo mundo!");  
    } // fim do método main  
  
} // fim da classe AloMundo
```

Aqui temos:

System é uma **classe**;

out é um **objeto** que pertence à classe **System** e que representa a saída padrão;

println é um **método** do objeto **out**.

```
/* Arquivo: AloMundo.java */  
public class AloMundo {  
    public static void main(String [] args)  
    {  
        System.out.println("Alo mundo!");  
    } // fim do método main  
  
} // fim da classe AloMundo
```

Para acessarmos os elementos dentro de uma classe ou objeto usamos o operador **ponto**.

```
/* Arquivo: AloMundo.java */  
public class AloMundo {  
    public static void main(String [] args)  
    {  
        System.out.println("Alo mundo!");  
    } // fim do método main  
  
} // fim da classe AloMundo
```

O método **println** simplesmente imprime a string passada como parâmetro na saída padrão e pula uma linha.

Repare que, assim como no C/C++, todo comando deve terminar com ;

- ❑ Java é classificada como uma linguagem **fortemente tipada**.
- ❑ Isso quer dizer que **todos** os dados básicos manipulados por um programa Java estão associados a um determinado tipo.
- ❑ Quando realizamos qualquer operação com diversos dados diferentes, o Java usa o tipo de cada dado para verificar se a operação é válida.
- ❑ Qualquer incompatibilidade entre os tipos de dados, o compilador Java acusa como um erro.



□ Java possui 8 tipos primitivos de dados:

Tipo	Bits	Valor mínimo	Valor máximo
boolean	1	true ou false	
char	16	até 65.536 caracteres	
byte	8	-128	127
short	16	-32.768	32.767
int	32	-2.147.483.648	2.147.483.647
long	64	-9.223.372.036.854.775.808	9.223.372.036.854.775.807
float	32	$\sim -1.4e-45$	$\sim 3.4e38$
double	64	$\sim -4.9e-324$	$\sim 1.7e308$

☐ Constantes Inteiras

- ✓ Compostas de dígitos de 0 a 9
- ✓ Podem ser iniciadas com sinal negativo (-)

✓ Exemplos:

19 -> int

-3 -> int

1265L -> long (terminado com **L** maiúsculo)

-9876L -> long (terminado com **L** maiúsculo)

□ Constantes de Ponto Flutuante

✓ Devem conter **sempre** o ponto (para diferenciar das constantes inteiras).

✓ Exemplos:

0.234 -> double

-125.65 -> double

4.93f -> float (terminado com **f** minúsculo)

-12.765f -> float (terminado com **f** minúsculo)



□ Constantes Caracter

✓ É uma letra ou símbolo entre **aspas simples**.

✓ Exemplos:

'a'

'.'

'*'



□ Constantes Booleanas

- ✓ São representados pelas palavras `true` ou `false`.

□ Constantes String

- ✓ Consiste de uma sequência de zero ou mais caracteres entre **aspas duplas**.
- ✓ Exemplos:

```
"Oba !"
```

```
"Rio de Janeiro"
```

```
"A resposta é: "
```

```
"a"
```

```
"Ela gritou \"Socorro !!!\""
```

- ❑ Identificadores são usados para **nomear** os elementos do nosso programa: variáveis, constantes, classes, métodos, atributos, etc.
- ❑ Os identificadores devem começar sempre com uma letra, \$, ou underscore (_).
- ❑ Após o primeiro caracter, são permitidos letras, \$, _ , ou dígitos.
- ❑ Não há limite de tamanho para o identificador.
- ❑ Identificadores em Java são *case-sensitive*, ou seja, **pessoa**, **PESSOA** e **Pessoa** são identificadores diferentes.
- ❑ Apesar de ser possível, **não é recomendável** o uso de caracteres de acentuação nos identificadores (ç, á, é, í, ó, ú, â, ê, ô, à, ã, õ).

- Algumas palavras reservadas não são permitidas como identificadores:

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

- A declaração de variáveis tem a mesma sintaxe do C/C++:

```
int _a;  
double $c;  
char c1, c2, c3;  
long nomeBastanteExtensoParaMinhaVariavel;  
int i, j;  
boolean achou;
```

- A declaração e inicialização de variáveis tem a mesma sintaxe do C/C++:

```
double $c = 0.1;  
char c1 = 'A', c2, c3;  
long nomeBastanteExtensoParaMinhaVariavel = 1000L;  
int i = 0, j = 1;  
boolean achou = false;
```

- Variáveis não inicializadas têm valor **indefinido** !
- Se você tentar usar uma variável antes de definir seu valor, o compilador Java **vai acusar um erro**.

- A declaração e inicialização de **constantes** tem a mesma sintaxe da declaração e inicialização de variáveis, acrescidas da palavra **final**:

```
final int MAXIMO = 100;
```

```
final float PI = 3.14159f;
```

```
final char FIM = '$';
```

```
final boolean OK = true;
```

Constantes não podem ter seu valor alterado durante o programa.

Caso haja uma tentativa de alterar um desses valores o compilador Java acusará um erro.

□ Apesar de não ser obrigatório, Java possui algumas convenções de nomenclatura que os programadores seguem:

1. Nomes de variáveis, atributos e métodos: utilizar o formato *camelCase* (primeira letra minúscula e primeira letra das demais palavras em maiúscula)

```
nomeCliente
```

```
matriculaAluno
```

```
cpf
```

```
dataNascimento
```

1. Nomes de constantes: devem ser definidas em caixa alta e usar o *underscore* como separador:

```
LIMITE_SUPERIOR
```

```
MAX
```

- Apesar de não ser obrigatório, Java possui algumas convenções de nomenclatura que os programadores seguem:
- 3. Nomes de classes: utilizar o formato *PascalCase* (todas as primeiras letras das palavras em maiúsculo).

AloMundo

Cliente

Disciplina

SistemaAcademico

AlunoBolsista

UnidadeEnsino

- Os operadores aritméticos previstos são:

Operador	Ação	Tipos
+	Soma	Inteiro e ponto flutuante
-	Subtração	Inteiro e ponto flutuante
*	Multiplicação	Inteiro e ponto flutuante
/	Divisão	Inteiro e ponto flutuante
%	Resto da divisão	Inteiro
++	Incremento	Inteiro e ponto flutuante
--	Decremento	Inteiro e ponto flutuante

- Ordem de precedência dos operadores aritméticos:

Operadores	Associação
++ --	Direita para esquerda
* / %	Esquerda para direita
+ -	Esquerda para direita

- Para alterar essa ordem devemos usar parênteses nas expressões.

```
float a, b, i = 10, j = 30, k = 40;  
a = i + j / k;      // a = 10.75  
b = (i + j) / k;    // b = 1
```

❑ Importante:

- ✓ O operador `/` pode ser aplicado tanto a números **inteiros** quanto a números de **ponto flutuante**.
- ✓ Quando todos os argumentos desse operador são inteiros então o resultado será um número inteiro, ou seja, a parte decimal é **desprezada**.

```
int i = 6, j = 3, k = 4;
```

```
i / j      resulta em 2
```

```
i / k      resulta em 1 e não em 1.5
```

- ✓ Para resolver esse problema será usado o operador de *casting*, que será visto mais adiante.

- Em Java o operador de atribuição é responsável por colocar o resultado da expressão à direita na variável à esquerda:

`variavel = expressão`

- Podemos encadear várias atribuições a partir de uma única expressão:

`variavel1 = variavel2 = variavel3 = ... = expressão`

- Exemplos:

```
int i, j, k;
```

```
double max, min;
```

```
i = j = k = 1;    // Todas as variaveis = 1
```

```
max = min = 0.0;  // Todas as variaveis = 0.0
```

Interface com o Usuário

- O desenvolvimento da interface com o usuário em Java depende muito do **tipo de aplicação** a ser desenvolvida:
 - ✓ Para aplicações desktop, as interfaces podem ser construídas com as bibliotecas AWT ou Swing do próprio Java.
 - ✓ Para as aplicações Web, as interfaces podem usar HTML/JavaScript/CSS ou um conjunto de componentes específicos de acordo com a tecnologia adotada. Por exemplo: componentes RichFaces para desenvolvimento usando a tecnologia Java Server Faces (JSF).
 - ✓ Para aplicações móveis, normalmente são adotados componentes específicos dependendo do SO. Por exemplo: para o Android existem componentes específicos desenvolvidos pela Google.
- Cada uma dessas tecnologias requer um tempo de aprendizado. Por isso, nesse curso vamos adotar somente a entrada/saída padrão via console (teclado e monitor em modo texto).

□ Classe System

- ✓ Pertence à biblioteca padrão do Java chamada `java.lang`.
- ✓ A classe System define, dentre outras coisas, os arquivos de entrada e saída padrão.
- ✓ O arquivo `out` representa a saída de vídeo.
- ✓ O arquivo `in` representa a entrada via teclado.
- ✓ Note que `in` e `out` são objetos e, por isso, podemos acessar seus métodos.
- ✓ Para realizar a saída de dados via console (vídeo) usamos os métodos `print`, `println` ou `printf` do objeto `out`.

□ Classe Scanner

- ✓ É usada para realizar a entrada de dados.
- ✓ Pertence à uma biblioteca do Java chamada `java.util`.
- ✓ Por não estar definido em uma biblioteca padrão, para usar a classe Scanner precisamos informar onde essa ela se encontra. Isso é feito através do comando `import`:

```
import java.util.Scanner;
```

□ Classe Scanner

- Existem vários métodos na classe Scanner para fazer a entrada de dados:

- ler um int: `nextInt()`

- ler um double: `nextDouble()`

- ler um float: `nextFloat()`

- ler um character: `nextLine().charAt(0)`

- ler um long: `nextLong()`

- ler uma string: `nextLine()`

- Vamos ver o uso da classe Scanner com um exemplo.



```
/* Arquivo: Leitura .java */
```

```
import java.util.Scanner;
```

```
public class Leitura {
```

```
    public static void main(String [] args) {
```

```
        int idade;
```

```
        Scanner teclado = new Scanner(System.in);
```

```
        System.out.println("Qual a sua idade ? ");
```

```
        idade = teclado.nextInt();
```

```
        System.out.printf("Idade = %d\n", idade);
```

```
    } // fim do metodo main
```

```
} // fim da classe
```

Importa a classe **Scanner** para avisar ao compilador Java onde ela está definida. Nesse caso ela está definida no pacote **java.util**.



Entrada/Saída via Console

```
/* Arquivo: Leitura .java */  
import java.util.Scanner;  
public class Leitura {  
    public static void main(String [] args) {  
        int idade;  
        Scanner teclado = new Scanner(System.in);  
        System.out.println("Qual a sua idade ? ");  
        idade = teclado.nextInt();  
        System.out.printf("Idade = %d\n", idade);  
    } // fim do metodo main  
} // fim da classe
```

Cria um objeto da classe **Scanner**.
O operador **new** é usado para criar esse objeto.

A variável **teclado** passa a referenciar esse objeto.

Ao criar o objeto da classe Scanner, informamos de onde serão lidos os dados: **System.in**



Entrada/Saída via Console

```
/* Arquivo: Leitura .java */  
import java.util.Scanner;  
public class Leitura {  
    public static void main(String [] args) {  
        int idade;  
        Scanner teclado = new Scanner(System.in);  
        System.out.println("Qual a sua idade ? ");  
        idade = teclado.nextInt();  
        System.out.printf("Idade = %d\n", idade);  
    } // fim do metodo main  
} // fim da classe
```

System.in é um objeto que referencia a entrada padrão, que no nosso caso é o **teclado**.

Entrada/Saída via Console

```
/* Arquivo: Leitura .java */  
import java.util.Scanner;  
public class Leitura {  
    public static void main(String [] args) {  
        int idade;  
        Scanner teclado = new Scanner(System.in);  
        System.out.println("Qual a sua idade ? ");  
        idade = teclado.nextInt();  
        System.out.printf("Idade = %d\n", idade);  
    } // fim do metodo main  
} // fim da classe
```

Utilizamos o método **nextInt** para ler um número inteiro da entrada padrão (teclado). O número lido é armazenado na variável **idade**.

Entrada/Saída via Console

```
/* Arquivo: Leitura .java */  
import java.util.Scanner;  
public class Leitura {  
    public static void main(String [] args) {  
        int idade;  
        Scanner teclado = new Scanner(System.in);  
        System.out.println("Qual a sua idade ? ");  
        idade = teclado.nextInt();  
        System.out.printf("Idade = %d\n", idade);  
    } // fim do metodo main  
} // fim da classe
```

O método **printf()** do Java usa os mesmos formatadores da função **printf()** do C/C++.

- ❑ Exercício 1: Leia duas variáveis inteiras e imprima a soma, subtração, multiplicação e divisão entre elas.
- ❑ Exercício 2: Altere o tipo das duas variáveis do exercício 1 para ponto flutuante.

```
import java.util.Scanner;

public class Ex01 {
    public static void main(String [] args)
    {
        Scanner teclado = new Scanner(System.in);

        /* Coloque o resto do codigo aqui ! */

    }
}
```

- Exercício 3: Leia o salário e o percentual de aumento. Em seguida, aplique o percentual de aumento sobre o salário e imprima o novo salário.

- Exercício 4: Leia o raio de um círculo. Em seguida imprima o perímetro ($2\pi R$) e a área (πR^2) do círculo com esse raio.

- Os operadores `++` e `--` são **muito** utilizados em Java.
- Os operadores `++` e `--` podem ser prefixados ou pósfixados. Assim:

`x++` usa o valor de `x` e depois incrementa `x`

`++x` incrementa `x` e depois usa o valor já incrementado

- Qual o valor final de `i`, `j` e `k` nas expressões abaixo?

```
int i, j, k;
```

```
i = 10;
```

```
j = i++;
```

```
k = ++j;
```

Incremento/Decremento

- Os operadores `++` e `--` são **muito** utilizados em Java.
- Os operadores `++` e `--` podem ser prefixados ou pósfixados. Assim:
 - `x++` usa o valor de `x` e **depois incrementa** `x`
 - `++x` incrementa `x` e **depois usa** o valor já incrementado
- Qual o valor final de `i`, `j` e `k` nas expressões abaixo?

```
int i, j, k;
```

```
i = 10;
```

```
j = i++;
```

```
k = ++j;
```

Esse trecho pode ser reescrito como:

```
i = 10;
```

```
j = i;
```

```
i++;
```

```
++j;
```

```
k = j;
```

Assim, `i`, `j` e `k` terão valor 11.



- Em programação é comum encontrar expressões como:

```
quantidade = quantidade + 10;
```

```
salario = salario - salario * percentual / 100;
```

- Em programação é comum encontrar expressões como:

```
quantidade = quantidade + 10;
```

quantidade quantidade + 10;

variável variável expressão

```
salario = salario - salario * percentual / 100;
```

salario salario - salario * percentual / 100;

variável variável expressão

- Em Java, é muito comum combinar os operadores **aritméticos** com o operador de **atribuição**.
- A sintaxe da atribuição aritmética é:

Expressão normal	Expressão
<code>var = var + expressão</code>	<code>var += expressão</code>
<code>var = var - expressão</code>	<code>var -= expressão</code>
<code>var = var * expressão</code>	<code>var *= expressão</code>
<code>var = var / expressão</code>	<code>var /= expressão</code>
<code>var = var % expressão</code>	<code>var %= expressão</code>



□ Exemplos:

```
quantidade = quantidade + 10;
```

```
quantidade += 10;
```

```
salario = salario - salario * percentual / 100;
```

```
salario -= salario * percentual / 100;
```

- Quando misturamos vários tipos em uma expressão, o Java tenta sempre converter os tipos com valores **menos significativos** para tipos **mais significativos** para não haver perda de dados durante o processamento.
- Essa conversão se dá na seguinte ordem:
`byte → short → int → long → float → double`
- Exemplos:
 - ✓ Se uma expressão envolve tipos byte e int, os valores das variáveis do tipo byte serão convertidos para int antes de avaliar a expressão.
 - ✓ Se uma expressão envolve os tipos int, float e double, os valores das variáveis int e float serão convertidos para double antes da avaliação.

- Uma expressão só pode ser atribuída a uma variável se o tipo dessa expressão for **igual ou menos significativo** que o tipo da variável. Caso contrário será gerado um **erro de compilação**.
- Assim, a atribuição deve ser feita obedecendo à seguinte ordem :
byte → short → int → long → float → double
- Exemplos:
 - ✓ Um expressão do tipo int pode ser armazenada em uma variável do tipo int ou long.
 - ✓ Um expressão do tipo float pode ser armazenada em uma variável do tipo float ou double.
 - ✓ Um expressão do tipo long pode ser armazenada em uma variável do tipo long, float ou double

- Para forçarmos a conversão de um tipo para outro usamos o operador de *casting*. Existem duas sintaxes:

(tipo) variável converte a *variável* para o *tipo*

(tipo) (expressão) converte o *resultado da expressão*
para o *tipo*

- Exemplos:

```
int i = 6, j = 3, k = 4;
```

```
(float) i / j converte i para 6.0 e o resultado é 2.0
```

```
(float) (i) / k converte i para 6.0 e o resultado é 1.5
```

```
(float) (i / k) converte 1 para 1.0, ou seja só faz o cast  
depois da divisão
```

□ Qual o resultado das expressões abaixo?

`5 * 4 / 6 + 7`

`5 * 4 / (6 + 7)`

`5 * 4.0 / 6 + 7`

`5 * 4 % 6 + 7`

`5 * 4 / (float) 6 + 7`

`(4 / 3) + (3.0 * 5)`

`(4 / 3.0) + (3 * 5)`

`(int) (4 / 3.0) + (3 * 5)`

□ Qual o resultado das expressões abaixo?

`5 * 4 / 6 + 7` `10`

`5 * 4 / (6 + 7)` `1`

`5 * 4.0 / 6 + 7` `10.33333`

`5 * 4 % 6 + 7` `9`

`5 * 4 / (float) 6 + 7` `10.33333`

`(4 / 3) + (3.0 * 5)` `16.0`

`(4 / 3.0) + (3 * 5)` `16.33333`

`(int) (4 / 3.0) + (3 * 5)` `16`

- Os operadores relacionais previstos em Java são:

Operador	Significado
>	Maior que
<	Menor que
>=	Maior ou igual a
<=	Menor ou igual a
==	Igual a
!=	Diferente de

- A sintaxe das operações relacionais é:

`expressão_aritmética_1 op_relacional expressão_aritmética_2`

- Os operadores aritméticos tem **precedência** sobre os operadores relacionais. Assim, se colocarmos esses dois tipos de operadores em uma mesma expressão, os operadores aritméticos serão avaliados primeiro e, em seguida, os relacionais.

$$1 + 3 \geq 3 + 6$$

$$(1 + 3) \geq (3 + 6)$$

$$5.0 / 3 \leq 10 / (4 + 1)$$

$$(5.0 / 3) \leq (10 / (4 + 1))$$

- Os operadores lógicos previstos em Java são:

Operador		Significado
&&	E (AND)	
	OU (OR)	
!	NEGAÇÃO (NOT)	

- A sintaxe das operações relacionais é:

`expressão_relacional_1 op_lógico expressão_relacional_2`

- O operador **?:** é usado em expressões condicionais, ou seja, uma expressão que podem ter um ou outro valor dependendo de uma condição. Sua sintaxe é:

`condição ? expressão_1 : expressão_2`

- A avaliação é feita da seguinte forma:

se `condição` for verdadeira então

retorna o resultado da `expressão_1`

senão

retorna o resultado da `expressão_2`

Exemplo:

```
int menor, maior, i, j;
```

```
if (i < j)
```

```
    menor = i;
```

```
else
```

```
    menor = j;
```



```
menor = i < j ? i : j;
```

□ Qual o valor de **i** em cada expressão?

```
int i = 1, j = 2, k = 3;
```

```
i = i > k ? i : k;
```

```
i = i > 0 ? j : k;
```

```
i = j > i ? ++k : --k;
```

```
i = k == i && k != j ? i + j : i - j;
```

□ Qual o valor de **i** em cada expressão?

```
int i = 1, j = 2, k = 3;
```

```
i = i > k ? i : k; 3
```

```
i = i > 0 ? j : k; 2
```

```
i = j > i ? ++k : --k; 4
```

```
i = k == i && k != j ? i + j : i - j; -1
```

- ❑ Exercício 5: Leia uma variável **t** com um tempo qualquer em segundos e imprima esse valor em hora, minuto e segundo.
- ❑ Exercício 6: Leia a distância percorrida por um carro, o tempo gasto e a quantidade de gasolina consumida. Em seguida, imprima a velocidade média (KM/h) e o consumo de combustível (Km/l).
- ❑ Exercício 7: Leia uma variável **n** inteira. Em seguida, imprima uma mensagem informando se o número **n** é par ou ímpar.
- ❑ Exercício 8: Leia duas variáveis com a quantidade de kilowatts consumidos em uma casa e o valor do kilowatt. Em seguida, calcule o valor a ser pago, concedendo um desconto de 10% caso o consumo seja menor que 150Kw.



□ Bloco de Comandos

- ✓ Cria um bloco que agrupa **declaração de variáveis** e **comandos**.
- ✓ Variáveis declaradas dentro de um bloco são visíveis **apenas nesse bloco**.
- ✓ Podemos aninhar blocos, ou seja, declarar um bloco dentro de outro.
- ✓ Se houver duas variáveis com o mesmo nome declaradas em um bloco externo e um bloco interno, a variável do bloco interno irá "esconder" a variável do bloco externo.
- ✓ Sintaxe:

```
{  
  /* declaracao e comandos */  
}
```


□ O que será impresso nesses dois casos ?

```
public static void  
main(String[]args)  
{  
    int i = 1;  
    {  
        int j;  
  
        i = 2;  
        j = 10;  
        i += j;  
    }  
    i++;  
    System.out.println(i);  
}
```

```
public static void main(String[]args)  
{  
    int i = 1;  
    {  
        int i, j;  
  
        i = 2;  
        j = 10;  
        i += j;  
    }  
    i++;  
    System.out.println(i);  
}
```

❑ O que será impresso nesses dois casos ?

```
public static void
main(String[] args)
{
    int i = 1;
    {
        int j;
        i = 2;
        j = 10;
        i += j;
    }
    i++;
    System.out.println(i);
}
```

Como só existe uma variável *i* nesse código, ela será usada no bloco interno e externo. Logo,

```
public static void main(String[] args)
{
    int i = 1;
    {
        int i, j;
        i = 2;
        j = 10;
        i += j;
    }
    i++;
    System.out.println(i);
}
```

A variável *i* do bloco mais interno esconde a variável *i* do bloco externo. Logo, será

- ❑ **Comando if...else:** Comando de seleção que permite analisar uma expressão lógica e desviar o fluxo de execução.

✓ **Sintaxes:**

<pre>if (expressao_logica) comando;</pre>	<pre>if (expressao_logica) bloco_de_comandos</pre>
<pre>if (expressao_logica) comando1;</pre>	<pre>if (expressao_logica) bloco_de_comandos1</pre>
<pre>else comando2;</pre>	<pre>else bloco_de_comandos2</pre>
<pre>if (expressao_logica1) comando1;</pre>	<pre>if (expressao_logica) bloco_de_comandos1</pre>
<pre>else if (expressao_logica2) comando2;</pre>	<pre>else if (expressao_logica) bloco_de_comandos2</pre>
<pre>else comando3;</pre>	<pre>else bloco_de_comandos3</pre>

- Exercício 9: Ler um número real x e imprimi-lo arredondando seu valor para mais ou para menos. Se a parte decimal for menor que 0.5 arredonda para menos. Se for maior ou igual a 0.5 arredonda para mais.
- Exercício 10: Ler um número inteiro n e mais dois números (inferior e superior) que formam um intervalo. Ao final, imprima uma mensagem informando se n está antes, dentro ou depois do intervalo.
- Exercício 11: Ler um caractere op representando uma operação aritmética (+, -, *, /) e em seguida dois números reais a e b . Imprimir a expressão matemática junto com o seu resultado no formato: $a\ op\ b = \text{resultado}$

□ Comando while

- ✓ Avalia uma expressão lógica e executa um bloco de comando enquanto ela for verdadeira
- ✓ O bloco é executado ZERO ou mais vezes.
- ✓ Sintaxe:

```
while (expressao_logica)  
    comando;
```

```
while (expressao_logica)  
    bloco_de_comandos
```



Estruturas de Repetição – do..while

□ Comando `do..while`

- ✓ Avalia uma expressão lógica e executa um bloco de comando enquanto ela for verdadeira.
- ✓ O bloco é executado UMA ou mais vezes.
- ✓ Sintaxe:

```
do
    bloco_de_comandos
while (expressao_logica);
```

□ Comando **for**

- ✓ Executa um bloco de comandos enquanto uma expressão booleana for verdadeira.
- ✓ É composto de 3 partes.
- ✓ Sintaxe:

```
for (expr_inicializacao; expressao_logica; expr_incremento)
    comando;
for (expr_inicializacao; expressao_logica; expr_incremento)
    bloco_de_comandos
```

□ A execução do **for** se dá da seguinte forma:

1. Executa a expressão de inicialização
2. Testa a expressão lógica. Se for FALSA termina o for
3. Executa o bloco de comandos
4. Executa a expressão de incremento
5. Volta para o passo 2

```
for (expr_inicializacao; expressao_logica; expr_incremento)  
{  
    comando1;  
    comando2;  
}
```


□ Assim, o comando **for**:

```
for (expr_inicializacao; expressao_logica; expr_incremento)  
{  
    comando1;  
    comando2;  
}
```

é equivalente a:

```
expr_inicializacao;  
while(expressao_logica)  
{  
    comando1;  
    comando2;  
    expr_incremento;  
}
```

- Conforme dito anteriormente, o **for** é composto de 3 partes. Entretanto, **nenhuma** dessas 3 partes é obrigatória.

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

b) `for (int i = 0; i < 10;) {
 System.out.println(i);
 i += 2;
 }`

c) `int i = 1;
for (; i < 50; i *= 2)
 System.out.println(i);`

d) `int i = 20;
for (; i >= 0;) {
 System.out.println(i);
 i--;
}`

e) `for (int i = 0; ; i++)
 System.out.println(i);`

f) `int i = 0;
for (; ; i++)
 System.out.println(i);`

g) `int i = 0;
for (; ;)
 System.out.println(i);`

- Exercício 12: Ler dois números inteiros (a e b) e imprimir os pares no intervalo $a..b$, além da soma e da média desses números. Caso a seja maior que b , imprima os números no intervalo $b..a$.
- Exercício 13: Ler um número de alunos n . Em seguida, ler as notas dos n alunos e imprimir, ao final, a média da turma.



□ Comando **switch**

- ✓ É um comando de seleção semelhante ao if-else, porém ele é mais recomendado quando temos **muitos caminhos** possíveis a partir de uma única condição.
- ✓ A expressão do switch tem que ser, obrigatoriamente, do tipo caracter (char) ou inteiro (byte, short, int ou long).
- ✓ O comando **break** é usado para terminar o switch.

□ Comando **switch**

```
switch(expressão)
{
    case opção1: comando1;
                comando2;
                break;
    case opção2: comando1;
                break;
    case opção3: comando1;
                comando2;
                break;
    default:    comando1;
                break;
}
```

opção1, opção2, opção3,
etc., podem ser **variáveis**
ou **constantes**.

Quando executa um
break o switch termina.

Se a expressão não for igual
a nenhuma opção é
executado o **default**.

□ Importante:

1. Quando o **switch** encontra uma **opção igual ao valor da expressão**, ele executa **todos** os comandos daí em diante até encontrar o comando **break**.
2. O **case** pode ter um comando **vazio**.

```
switch(caracter)
{
    case 'a':
    case 'e':
    case 'i':
    case 'o':
    case 'u':    System.out.println("É uma vogal");
                break;

    case 'x':    System.out.println("Letra X");
    default:    System.out.println("Letra inválida");
                break;
}
```

- O comando **break** força a saída do *loop* mais interno de um comando de repetição (**while**, **do..while** ou **for**) ou em um comando **switch**.
- Exemplo:

```
int n;  
while(...)  
{  
    for (...)  
    {  
        n = n - 1  
        if (n == 0)  
            break;  
        n++;  
    }  
}
```

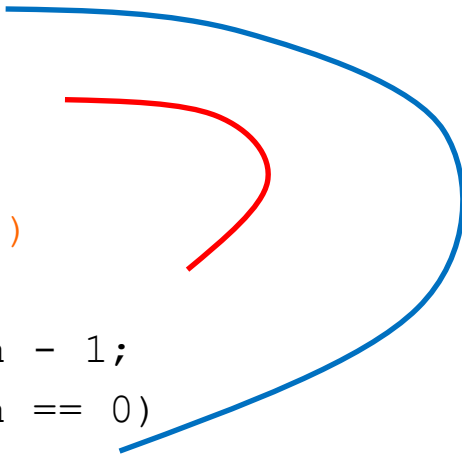
Sai do **for**, que é o comando de repetição mais interno.
Não executa o **n++**

```
System.out.println(n);
```

Estruturas de Controle – continue

- O comando **continue** força o início da próxima interação do *loop* mais interno de um comando de repetição (**while**, **do..while** ou **for**).
- Exemplo:

```
int n;  
while(...)  
{  
    for (...)  
    {  
        n = n - 1;  
        if (n == 0)  
            continue;  
        System.out.print(n);  
    }  
}
```



Para o comando **for** reinicia o loop executando a expressão de incremento e depois testando a expressão lógica.

No **while** e **do..while** reinicia o loop testando a expressão lógica.

- Exercício 14: Ler notas de alunos até que o usuário digite **-1**. Ao final imprimir a quantidade de alunos, a média da turma, a maior nota e a menor nota.

- Exercício 15: Ler caracteres até que o usuário digite '.' (ponto). Ao final imprimir: a quantidade de vogais, a quantidade de dígitos e a quantidade dos demais caracteres.