

# Módulo 1

## Introdução à Programação I



## Lição 4

### Fundamentos da programação

**Autor**

Florence Tiu Balagtas

**Equipe**

Joyce Avestro

Florence Balagtas

Rommel Fera

Reginald Hutcherson

Rebecca Ong

John Paul Petines

Sang Shin

Raghavan Srinivas

Matthew Thompson

**Necessidades para os Exercícios****Sistemas Operacionais Suportados**

**NetBeans IDE 5.5** para os seguintes sistemas operacionais:

- Microsoft Windows XP Professional SP2 ou superior
- Mac OS X 10.4.5 ou superior
- Red Hat Fedora Core 3
- Solaris™ 10 Operating System (SPARC® e x86/x64 Platform Edition)

**NetBeans Enterprise Pack**, poderá ser executado nas seguintes plataformas:

- Microsoft Windows 2000 Professional SP4
- Solaris™ 8 OS (SPARC e x86/x64 Platform Edition) e Solaris 9 OS (SPARC e x86/x64 Platform Edition)
- Várias outras distribuições Linux

**Configuração Mínima de Hardware**

**Nota:** IDE NetBeans com resolução de tela em 1024x768 pixel

Sistema Operacional	Processador	Memória	HD Livre
Microsoft Windows	500 MHz Intel Pentium III workstation ou equivalente	512 MB	850 MB
Linux	500 MHz Intel Pentium III workstation ou equivalente	512 MB	450 MB
Solaris OS (SPARC)	UltraSPARC II 450 MHz	512 MB	450 MB
Solaris OS (x86/x64 Platform Edition)	AMD Opteron 100 Série 1.8 GHz	512 MB	450 MB
Mac OS X	PowerPC G4	512 MB	450 MB

**Configuração Recomendada de Hardware**

Sistema Operacional	Processador	Memória	HD Livre
Microsoft Windows	1.4 GHz Intel Pentium III workstation ou equivalente	1 GB	1 GB
Linux	1.4 GHz Intel Pentium III workstation ou equivalente	1 GB	850 MB
Solaris OS (SPARC)	UltraSPARC IIIi 1 GHz	1 GB	850 MB
Solaris OS (x86/x64 Platform Edition)	AMD Opteron 100 Series 1.8 GHz	1 GB	850 MB
Mac OS X	PowerPC G5	1 GB	850 MB

**Requerimentos de Software**

NetBeans Enterprise Pack 5.5 executando sobre Java 2 Platform Standard Edition Development Kit 5.0 ou superior (JDK 5.0, versão 1.5.0\_01 ou superior), contemplando a Java Runtime Environment, ferramentas de desenvolvimento para compilar, depurar, e executar aplicações escritas em linguagem Java. Sun Java System Application Server Platform Edition 9.

- Para **Solaris, Windows, e Linux**, os arquivos da JDK podem ser obtidos para sua plataforma em <http://java.sun.com/j2se/1.5.0/download.html>
- Para **Mac OS X**, Java 2 Platform Standard Edition (J2SE) 5.0 Release 4, pode ser obtida diretamente da Apple's Developer Connection, no endereço: <http://developer.apple.com/java> (é necessário registrar o download da JDK).

Para mais informações:

<http://www.netbeans.org/community/releases/55/relnotes.html>

## Colaboradores que auxiliaram no processo de tradução e revisão

Alexandre Mori	Hugo Leonardo Malheiros Ferreira	Mauro Regis de Sousa Lima
Alexis da Rocha Silva	Ivan Nascimento Fonseca	Namor de Sá e Silva
Aline Sabbatini da Silva Alves	Jacqueline Susann Barbosa	Néres Chaves Rebouças
Allan Wojcik da Silva	Jader de Carvalho Belarmino	Nolyanne Peixoto Brasil Vieira
André Luiz Moreira	João Aurélio Telles da Rocha	Paulo Afonso Corrêa
Andro Márcio Correa Louredo	João Paulo Cirino Silva de Novais	Paulo José Lemos Costa
Antonie de Assis Lima	João Vianney Barrozo Costa	Paulo Oliveira Sampaio Reis
Antonio Jose R. Alves Ramos	José Augusto Martins Nieviadonski	Pedro Antonio Pereira Miranda
Aurélio Soares Neto	José Leonardo Borges de Melo	Pedro Henrique Pereira de Andrade
Bruno da Silva Bonfim	José Ricardo Carneiro	Renato Alves Félix
Bruno dos Santos Miranda	Kleberth Bezerra G. dos Santos	Renato Barbosa da Silva
Bruno Ferreira Rodrigues	Lafaiete de Sá Guimarães	Reydersson Magela dos Reis
Carlos Alberto Vitorino de Almeida	Leandro Silva de Moraes	Ricardo Ferreira Rodrigues
Carlos Alexandre de Sene	Leonardo Leopoldo do Nascimento	Ricardo Ulrich Bomfim
Carlos André Noronha de Sousa	Leonardo Pereira dos Santos	Robson de Oliveira Cunha
Carlos Eduardo Veras Neves	Leonardo Rangel de Melo Filardi	Rodrigo Pereira Machado
Cleber Ferreira de Sousa	Lucas Mauricio Castro e Martins	Rodrigo Rosa Miranda Corrêa
Cleyton Artur Soares Urani	Luciana Rocha de Oliveira	Rodrigo Vaez
Cristiano Borges Ferreira	Luís Carlos André	Ronie Dotzlaw
Cristiano de Siqueira Pires	Luís Octávio Jorge V. Lima	Rosely Moreira de Jesus
Derlon Vandri Aliendres	Luiz Fernandes de Oliveira Junior	Seire Pareja
Fabiano Eduardo de Oliveira	Luiz Victor de Andrade Lima	Sergio Pomerancblum
Fábio Bombonato	Manoel Cotts de Queiroz	Silvio Sznifer
Fernando Antonio Mota Trinta	Marcello Sandi Pinheiro	Suzana da Costa Oliveira
Flávio Alves Gomes	Marcelo Ortolan Pazzetto	Tásio Vasconcelos da Silveira
Francisco das Chagas	Marco Aurélio Martins Bessa	Thiago Magela Rodrigues Dias
Francisco Marcio da Silva	Marcos Vinicius de Toledo	Tiago Gimenez Ribeiro
Gilson Moreno Costa	Maria Carolina Ferreira da Silva	Vanderlei Carvalho Rodrigues Pinto
Givailson de Souza Neves	Massimiliano Girolodi	Vanessa dos Santos Almeida
Gustavo Henrique Castellano	Mauricio Azevedo Gamarra	Vastí Mendes da Silva Rocha
Hebert Julio Gonçalves de Paula	Mauricio da Silva Marinho	Wagner Eliezer Roncoletta
Heraldo Conceição Domingues	Mauro Cardoso Mortoni	

## Auxiliadores especiais

Revisão Geral do texto para os seguintes Países:

- **Brasil** – Tiago Flach
- **Guiné Bissau** – Alfredo Cá, Bunene Sisse e Buon Olossato Quebi – ONG Asas de Socorro

## Coordenação do DFJUG

- **Daniel deOliveira** – JUGLeader responsável pelos acordos de parcerias
- **Luci Campos** - Idealizadora do DFJUG responsável pelo apoio social
- **Fernando Anselmo** - Coordenador responsável pelo processo de tradução e revisão, disponibilização dos materiais e inserção de novos módulos
- **Regina Mariani** - Coordenadora responsável pela parte jurídica
- **Rodrigo Nunes** - Coordenador responsável pela parte multimídia
- **Sérgio Gomes Veloso** - Coordenador responsável pelo ambiente JEDI™ (Moodle)

## Agradecimento Especial

**John Paul Petines** – Criador da Iniciativa JEDI™

**Rommel Feria** – Criador da Iniciativa JEDI™

# 1. Objetivos

Nesta lição discutiremos as partes básicas de um programa em Java. Começaremos explicando as partes do programa Hello.java mostrado na última lição. Discutiremos ao longo desta lição também dicas e convenções para se escrever programas de fácil entendimento.

Ao final desta lição, o estudante será capaz de:

- Identificar e entender as partes básicas de um programa escrito em Java.
- Diferenciar, em um programa, o que são: os tipos primitivos de dados, variáveis, identificadores e operadores.
- Desenvolver, em Java, um programa usando os conceitos compreendidos nesta lição.

## 2. Entendendo meu primeiro programa em Java

Tentaremos compreender este primeiro programa.

```
public class Hello
{
    /**
     * Meu primeiro programa em Java
     */
    public static void main(String[] args) {
        // Exibir a mensagem "Hello world" na tela
        System.out.println("Hello world!");
    }
}
```

Esta primeira linha do código:

```
public class Hello
```

Indica o nome da classe, que neste caso é **Hello**. Em Java, todo e qualquer código deverá ser escrito dentro da declaração de uma classe. Fazemos isso usando a palavra-chave **class**. Além disso, a classe usa um identificador de acesso **public**, indicando que a classe é acessível para outras classes de diferentes pacotes (pacotes são coleções de classes). Trataremos de pacotes e identificadores de acesso mais tarde, ainda neste módulo.

A próxima linha contém uma chave {. Indica o início de um bloco de instruções. Neste programa posicionamos a chave na linha após a declaração da classe, entretanto, poderíamos também colocá-la na mesma linha em que a declaração foi feita. Então, o código seria escrito da seguinte forma:

```
public class Hello
{
```

ou

```
public class Hello {
```

As próximas 3 linhas indicam um comentário em Java. Um comentário é uma explicação do programador usada na documentação de uma parte do código. Este comentário não é propriamente uma parte do código, é usado apenas para fins de documentação do programa. É uma boa prática de programação adicionar comentários relevantes ao código.

```
/**
 * Meu primeiro programa em Java
 */
```

Um comentário pode ser indicado pelos delimitadores `"/**` e `*/"`. Qualquer coisa entre estes delimitadores é ignorado pelo compilador Java e é tratado como comentário. A próxima linha,

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

que também pode ser escrita da seguinte forma:

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

indica o nome de um método no programa que é o método principal **main**. O método **main** é o ponto de partida para qualquer programa feito em Java. Todo e qualquer programa escrito

em Java, com exceção de Applets, inicia com o método **main**. Certifique-se de que a assinatura do método (conforme descrita acima) está correta.

A linha seguinte é também um comentário em Java,

```
// exibe a mensagem "Hello world" na tela
```

Até agora, já aprendemos duas maneiras de fazer comentários em Java. Uma é posicionar o comentário entre `/*` e `*/`, e a outra é colocar `/**` antes do comentário. A linha de instrução abaixo,

```
System.out.println("Hello world!");
```

escreve o texto "Hello World!" na tela. O comando `System.out.println( )`, escreve na saída padrão do computador o texto situado entre aspas duplas.

As últimas duas linhas, que contêm somente uma chave em cada, simbolizam, respectivamente, o fechamento do método **main** e da **classe**.

***Dicas de programação :***

1. Os programas em Java devem sempre conter a terminação `.java` no nome do arquivo.
2. O nome do arquivo deve sempre ser idêntico ao nome da classe pública. Então, por exemplo, se o nome da classe pública é **Hello** o arquivo deverá ser salvo com o nome: **Hello.java**.
3. Inserir comentários sobre o que a classe ou método realiza, isso facilitará o entendimento de quem posteriormente ler o programa, incluindo o próprio autor.

## 3. Comentários em Java

Comentários são notas escritas pelo programador para fins de documentação. Estas notas não fazem parte do programa e não afetam o fluxo de controle.

Java suporta três tipos de comentários: comentário de linha estilo C++, comentário de bloco estilo C e um comentário estilo Javadoc (utilizado compor a documentação do programa).

### 3.1. Comentário de linha

Comentários com estilo em C++ se iniciam por `//`. Todo e qualquer texto colocado após as `//` é ignorado pelo compilador e tratado como comentário. Por exemplo:

```
// Este é um comentário estilo C++ ou comentário de linha
```

### 3.2. Comentário de bloco

Comentários com estilo em C, também chamados de comentários multi-linhas, se iniciam com `/*` e terminam com `*/`. Todo o texto posto entre os dois delimitadores é tratado como comentário. Diferente do comentário estilo C++, este pode se expandir para várias linhas. Por exemplo:

```
/*
 * Este é um exemplo de comentário
 * estilo C ou um comentário de bloco
 *
 */
```

### 3.3. Comentário estilo Javadoc

Este comentário é utilizado na geração da documentação em **HTML** dos programas escritos em Java. Para se criar um comentário em estilo **Javadoc** deve se iniciar o comentário com `/**` e terminá-lo com `*/`. Assim como os comentários estilo C, este também pode conter várias linhas. Este comentário também pode conter certas tags que dão mais informações à documentação. Por exemplo:

```
/**
 * Este é um exemplo de um comentário especial usado para \n
 * gerar uma documentação em HTML. Este utiliza tags como:
 * @author Florence Balagtas
 * @version 1.2
 */
```

Este tipo de comentário deve ser utilizado antes da assinatura da classe:

```
public class Hello {
```

Para documentar o objetivo do programa ou antes da assinatura de métodos:

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

Para documentar a utilidade de um determinado método.

## 4. Instruções e Blocos em Java

Uma instrução é composta de uma ou mais linhas terminadas por ponto-e-vírgula. Um exemplo de uma simples instrução pode ser:

```
System.out.println("Hello world");
```

Um bloco é formado por uma ou mais instruções agrupadas entre chaves indicando que formam uma só unidade. Blocos podem ser organizados em estruturas aninhadas indefinidamente. Qualquer quantidade de espaços em branco é permitida. Um exemplo de bloco pode ser:

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

### ***Dicas de programação:***

1. Na criação de blocos, a chave que indica o início pode ser colocada ao final da linha anterior ao bloco, como no exemplo:

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

ou na próxima linha, como em:

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

2. É uma boa prática de programação organizar as instruções que serão colocadas após o início de um bloco, como por exemplo:

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



## 5. Identificadores em Java

Identificadores são representações de nomes de variáveis, métodos, classes, etc. Exemplos de identificadores podem ser: Hello, main, System, out.

O compilador Java difere as letras maiúsculas de minúsculas (case-sensitive). Isto significa que o identificador **Hello** não é o mesmo que **hello**. Os identificadores em Java devem começar com uma letra, um underscore "\_", ou um sinal de cifrão "\$". As letras podem estar tanto em maiúsculo quanto em minúsculo. Os caracteres subseqüentes podem usar números de 0 a 9.

Os identificadores não podem ter nomes iguais às palavras-chave ou palavras reservadas do Java, como: class, public, void, int, etc. Discutiremos mais sobre estas palavras mais tarde.

### Dicas de programação:

1. Para nomes de classes, a primeira letra deve ser maiúscula. Nomes de métodos ou variáveis devem começar com letra minúscula. Por exemplo:

```
ExemploDeNomeDeUmaClasse  
exemploDeNomeDeUmMetodo
```

2. No caso de identificadores com mais de uma palavra, a primeira letra de cada palavra, com exceção da primeira, deve vir em maiúsculo. Por exemplo:

```
charArray - fileNumber - className
```

3. Evite o uso de underscores no início de um identificador. Por exemplo:

```
_NomeDeClasse.
```

## 6. Palavras-chave em Java

Palavras-chave são identificadores que, em Java, foram pré-definidas para propósitos específicos. Não se pode usar esses identificadores como nomes de variáveis, métodos, classes, etc. A seguir, temos a lista com as palavras-chave em Java.

abstract	continue	for	new	switch
assert ***	default	goto *	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum ****	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp **	volatile
const *	float	native	super	while
* not used				
** added in 1.2				
*** added in 1.4				
**** added in 5.0				

*Figura 1: Palavras-Chave em Java*

Ao longo dos tópicos seguintes, iremos abordar os significados destas palavras-chave e como são usadas nos programas em Java.

**Nota:** **true**, **false** e **null** não são palavras-chave, porém, são palavras-reservadas, e, da mesma maneira, não é permitido seu uso na atribuição a nomes de variáveis, métodos ou classes.

## 7. Tipos de Dados em Java

Java possui 4 tipos de dados. Estes tipos de dados são divididos em: **boolean**, **character**, **integer** e **float-point**.

### 7.1. Boolean

Um dado **boolean** poderá assumir somente dois valores: **true** ou **false**.

### 7.2. Character

Os **characters** são representações da tabela **Unicode**. Um símbolo da tabela **Unicode** é um valor de 16 bits que pode substituir os símbolos da tabela **ASCII** (que possuem 8 bits). A tabela **Unicode** permite a inserção de símbolos especiais ou de outros idiomas. Para representar um caractere usam-se aspas simples. Por exemplo, a letra "a" será representada como 'a'. Para representar caracteres especiais, usa-se a "\" seguido pelo código do caractere especial. Por exemplo, '\n' para o caractere de nova linha, '\"' representar o **character** ' (aspas simples) e '\u0061' representação **Unicode** do símbolo 'a'.

### 7.3. Integer

Os dados do tipo **integer** vêm em diferentes formatos: **decimal** (base 10), **hexadecimal** (base 16), e **octal** (base 8). Ao usar estes diferentes tipos de dados **Integer** nos programas é necessário seguir algumas notações preestabelecidas. Para dados decimais não existe notação, basta escrever o número. Os números hexadecimais deverão ser precedidos por "0x" ou "0X". E os octais deverão ser precedidos por "0".

Por exemplo, considere o número **12**. Sua representação em decimal é apenas o número **12**, sem nenhuma notação adicional, enquanto que sua representação em hexadecimal é **0xC** (pois o número 12 em hexadecimal é representado pela letra C), e em octal ele é representado por **014**. O valor padrão para tipos de dados **Integer** é o tipo **int**. Um **int** é um valor, com sinal, de 32 bits.

Em alguns casos pode ser necessário forçar o dado **Integer** a ser do tipo **long**, para fazer isso basta colocar a letra "l" (L minúscula) ou "L" após o número. Um dado do tipo **long** é um valor com sinal de 64 bits. Falaremos mais sobre os tipos de dados mais tarde.

### 7.4. Float-Point

Os tipos de **float-point** representam dados **Integer** com parte fracionária. Um exemplo é o número 3.1415 (lembrando que o padrão inglês utiliza o "." como divisor da parte fracionária). Esses tipos podem ser expressos em notação científica ou padrão. Um exemplo de notação padrão é o número 583.45 enquanto que em notação científica ele seria representado por 5.8345e2. O valor padrão para um dado ponto-flutuante é o **double**, que é um valor de 64 bits. Se for necessário usar um número com uma precisão menor (32 bits) usa-se o **float**, que é finalizado pela letra "f" ou "F" acrescida ao número em questão, por exemplo, 583.45f.

## 8. Tipos de Dados Primitivos

A linguagem Java possui 8 tipos de dados primitivos. Eles são divididos nas seguintes representações:

Representação	Tipo de Dado	Dado Primitivo
lógico	Boolean	boolean
inteiro	Integer e Character	char, byte, short e int
inteiro longo	Integer	long
número fracionário	Float-point	float e double

Tabela 1: Representações dos dados primitivos

### 8.1. Lógico

O tipo **boolean** pode representar dois estados: **true** (verdadeiro) ou **false** (falso). Um exemplo é:

```
boolean resultado = true;
```

No exemplo demonstrado acima, é declarado um atributo chamado **resultado** do tipo **boolean** e atribuído a este o valor verdadeiro.

### 8.2. Inteiro

Os inteiros em Java podem ser representados em 5 formas, como já foi visto, e estas são: decimal, octal, hexadecimal, ASCII e **Unicode**. Como por exemplo:

```
2          // valor 2 em decimal
077        // 0 indica que ele está representado em octal.
0xBACC     // 0x indica que ele está representado em hexadecimal.
'a'        // representação ASCII
'\u0061'   // representação Unicode
```

O dado do tipo **char** é um inteiro especial, sendo exclusivamente positivo e representa um único **Unicode**. Ele deve ser, obrigatoriamente, colocado entre aspas simples ("). Sua representação como inteiro pode ser confusa para o iniciante, entretanto, o tempo e a prática farão com que se acostume com este tipo. Por exemplo:

```
char c = 97;    // representa o símbolo 'a'
byte b = 'a';   // em inteiro representa o número 97
```

É possível definir para qualquer inteiro nas formas mostradas. O que difere o tipo **char** dos demais inteiros é que a sua saída sempre será mostrada como um valor ASCII. Enquanto que os inteiros serão sempre mostrados por números decimais. Por exemplo:

```
char c = 97;
byte b = 'a';
System.out.println("char = " + c + " - byte = " + b);
```

Resultará:

```
char = a - byte = 97
```

Um cuidado deve ser tomado quanto aos inteiros: qualquer operação efetuada entre eles terá sempre como resultado um tipo **int**. Por exemplo:

```
byte b1 = 1;
byte b2 = 2;
byte resultado = b1 + b2;
```

Esta instrução final causará um erro de compilação, devendo ser modificada para:

```
int resultado = b1 + b2;
```

### 8.3. Inteiro Longo

Os inteiros têm por padrão o valor representado pelo tipo primitivo **int**. Pode-se representá-los como **long** adicionando, ao final do número, um "l" ou "L". Os tipos de dados inteiros assumem valores nas seguintes faixas:

<i>Tamanho em memória</i>	<i>Dado primitivo</i>	<i>Faixa</i>
8 bits	byte	$-2^7$ até $2^7-1$
16 bits	char	0 até $2^{16}-1$
16 bits	short	$-2^{15}$ até $2^{15}-1$
32 bits	int	$-2^{31}$ até $2^{31}-1$
64 bits	long	$-2^{63}$ até $2^{63}-1$

Tabela 2: Tipos e faixa de valores dos **Inteiros** e **Inteiro Longo**

#### **Dicas de programação:**

1. Para declarar um número como sendo um long é preferível usar "L" maiúsculo, pois, se este estiver em minúsculo, pode ser difícil distingui-lo do dígito 1.

### 8.4. Número Fracionário

Os dados do tipo ponto-flutuante possuem o valor double como padrão. Os números flutuantes possuem um ponto decimal ou um dos seguintes caracteres:

```
E ou e // expoente
F ou f // float
D ou d // double
```

São exemplos,

```
3.14           // tipo double
6.02E23        // double com expoente
2.718F         // float
123.4E+306D     // double
```

No exemplo acima, o número 23 após o E é implicitamente positivo. É equivalente a 6.02E

+23. Os dados de tipo ponto-flutuante podem assumir valores dentro das seguintes faixas:

<b>Tamanho em memória</b>	<b>Dado primitivo</b>	<b>Faixa</b>
32 bits	float	$-10^{38}$ até $10^{38}-1$
64 bits	double	$-10^{308}$ até $10^{308}-1$

Tabela 3: Tipos e faixa de valores dos **Número Fracionários**

## 9. Variáveis

Uma **variável** é um espaço na memória usado para armazenar o estado de um objeto.

Uma variável deve ter um **nome** e um **tipo**. O **tipo da variável** indica o tipo de dado que ela pode conter. O **nome das variáveis** deve seguir as mesmas regras de nomenclatura que os identificadores.

### 9.1. Declarando e inicializando Variáveis

A seguir, vemos como é feita a declaração de uma variável:

```
<tipo do dado> <nome> [= valor inicial];
```

**nota:** os valores colocados entre < > são obrigatórios, enquanto que os valores contidos entre [ ] são opcionais.

Aqui vemos um exemplo de programa que declara e inicializa algumas variáveis:

```
public class VariableSamples {
    public static void main( String[] args ){
        // declara uma variável com nome result e tipo boolean
        boolean result;

        // declara uma variável com nome option e tipo char
        char option;
        // atribui o símbolo C para a variável
        option = 'C';

        // declara uma variável com nome grade e tipo double
        // e a inicializa com o valor 0.0
        double grade = 0.0;
    }
}
```

#### **Dicas de programação:**

1. É sempre preferível que se inicialize uma variável assim que ela for declarada.
2. Use nomes com significado para suas variáveis. Se usar uma variável para armazenar a nota de um aluno, declare-a com o nome 'nota' e não simplesmente com uma letra aleatória 'x'.
3. É preferível declarar uma variável por linha, do que várias na mesma linha. Por exemplo:

```
int variavel1;
int variavel2;
```

E não:

```
int variavel1, variavel2;
```

### 9.2. Exibindo o valor de uma Variável

Para exibirmos em qualquer dispositivo de saída o valor de uma variável, fazemos uso dos seguintes comandos:

```
System.out.println()  
System.out.print()
```

Aqui está um simples programa como exemplo:

```
public class OutputVariable {  
    public static void main( String[] args ){  
        int value = 10;  
        char x;  
        x = 'A';  
  
        System.out.println(value);  
        System.out.println("The value of x = " + x );  
    }  
}
```

A saída deste programa será a seguinte:

```
10  
The value of x = A
```

### 9.3. System.out.println( ) e System.out.print( )

Qual é a diferença entre os comandos **System.out.println( )** e o **System.out.print( )**? O primeiro faz iniciar uma nova linha após ser exibido seu conteúdo, enquanto que o segundo não.

Considere as seguintes instruções:

```
System.out.print("Hello ");  
System.out.print("world!");
```

Essas instruções apresentarão a seguinte saída:

```
Hello world!
```

Considere as seguintes:

```
System.out.println("Hello ");  
System.out.println("world!");
```

Estas apresentarão a seguinte saída:

```
Hello  
world!
```

### 9.4. Referência de Variáveis e Valor das Variáveis

Iremos diferenciar os dois tipos de variáveis suportados pelo Java. Estes podem ser de **referência** ou de **valor**.

**As variáveis de "valor", ou primitivas,** são aquelas que armazenam dados no exato espaço de memória onde a variável está.



**As variáveis de referência** são aquelas que armazenam o endereço de memória onde o dado está armazenado. Ao declarar uma variável de certa classe (variável de classe), se declara uma variável de referência a um objeto daquela classe.

Por exemplo, vamos supor que se tenha estas duas variáveis do tipo **int** e da classe **String**.

```
int num = 10;  
String nome = "Hello";
```

Suponha que o quadro abaixo represente a memória do computador, com seus endereços de memória, o nome das variáveis e os tipos de dados que ele pode suportar.

Endereço de memória	Nome da variável	Dado
1001	num	10
:		:
1563	nome	Endereço (2000)
:		:
:		:
2000		"Hello"

A variável (do tipo int) **num** o dado é o atual valor contido por ela e, a referência da variável (do tipo string) **nome** somente é armazenado o endereço de memória que contém o valor da variável.

## 10. Operadores

Em Java temos diferentes tipos de operadores. Existem **operadores aritméticos**, **operadores relacionais**, **operadores lógicos** e **operadores condicionais**. Estes operadores obedecem a uma ordem de precedência para que o compilador saiba qual operação executar primeiro, no caso de uma sentença possuir grande variedade destes.

### 10.1. Operadores Aritméticos

Aqui temos a lista dos operadores aritméticos que podem ser utilizados na criação de expressões matemáticas:

<b>Operador</b>	<b>Uso</b>	<b>Descrição</b>
*	op1 * op2	Multiplica op1 por op2
/	op1 / op2	Divide op1 por op2
%	op1 % op2	Resto da divisão de op1 por op2
-	op1 - op2	Subtrai op2 de op1
+	op1 + op2	Soma op1 e op2

*Tabela 4: Operadores aritméticos e suas funções*

Aqui temos um programa que exemplifica o uso destes operadores:

```
public class ArithmeticDemo {
    public static void main(String[] args) {
        // alguns números
        int i = 37;
        int j = 42;
        double x = 27.475;
        double y = 7.22;
        System.out.println("Variables values...");
        System.out.println("    i = " + i);
        System.out.println("    j = " + j);
        System.out.println("    x = " + x);
        System.out.println("    y = " + y);

        // adição dos números
        System.out.println("Adding...");
        System.out.println("    i + j = " + (i + j));
        System.out.println("    x + y = " + (x + y));

        // subtração dos números
        System.out.println("Subtracting...");
        System.out.println("    i - j = " + (i - j));
        System.out.println("    x - y = " + (x - y));

        // multiplicação dos números
        System.out.println("Multiplying...");
        System.out.println("    i * j = " + (i * j));
        System.out.println("    x * y = " + (x * y));
    }
}
```

```
// divisão dos números
System.out.println("Dividing...");
System.out.println("    i / j = " + (i / j));
System.out.println("    x / y = " + (x / y));

// resto da divisão
System.out.println("Computing the remainder...");
System.out.println("    i % j = " + (i % j));
System.out.println("    x % y = " + (x % y));

// misturando operações
System.out.println("Mixing types...");
System.out.println("    j + y = " + (j + y));
System.out.println("    i * x = " + (i * x));
    }
}
```

e como saída, temos:

```
Variables values...
    i = 37
    j = 42
    x = 27.475
    y = 7.22
Adding...
    i + j = 79
    x + y = 34.695
Subtracting...
    i - j = -5
    x - y = 20.255
Multiplying...
    i * j = 1554
    x * y = 198.37
Dividing...
    i / j = 0
    x / y = 3.8054
Computing the remainder...
    i % j = 37
    x % y = 5.815
Mixing types...
    j + y = 49.22
    i * x = 1016.58
```

**Nota:** Quando um número de tipo inteiro e um outro de número fracionário são usados numa única operação, o resultado será dado pela variável de maior tipo, no caso, valor de número fracionário. O número inteiro é implicitamente convertido para o número fracionário antes da operação ter início.

## 11. Operadores de Incremento e Decremento

Além dos operadores aritméticos básicos, Java dá suporte ao operador unário de incremento (++) e ao operador unário de decremento (--). Operadores de incremento ou decremento aumentam ou diminuem em 1 o valor da variável. Por exemplo, a expressão,

```
count = count + 1; // incrementa o valor de count em 1
```

é equivalente a,

```
count++;
```

<b>Operador</b>	<b>Uso</b>	<b>Descrição</b>
++	op++	Incrementa op em 1; Avalia a expressão antes do valor ser acrescido
++	++op	Incrementa op em 1; Incrementa o valor antes da expressão ser avaliada
--	op--	Decrementa op em 1; Avalia a expressão antes do valor ser decrescido
--	--op	Decrementa op em 1; Decrementa op em 1 antes da expressão ser avaliada

*Tabela 5: Operadores de incremento e decremento*

Como visto na tabela acima, os operadores de incremento e decremento podem ser usados tanto antes como após o operando. E sua utilização dependerá disso. Quando usado antes do operando, provoca acréscimo ou decréscimo de seu valor antes da avaliação da expressão em que ele aparece. Por exemplo:

```
int i = 10,
int j = 3;
int k = 0;
k = ++j + i; //resultará em k = 4+10 = 14
```

Quando utilizado depois do operando, provoca, na variável, acréscimo ou decréscimo do seu valor após a avaliação da expressão na qual ele aparece. Por exemplo:

```
int i = 10,
int j = 3;
int k = 0;
k = j++ + i; //resultará em k = 3+10 = 13
```

### Dicas de programação:

1. Mantenha sempre as operações incluindo operadores de incremento ou decremento de forma simples e de fácil compreensão.

## 12. Operadores Relacionais

Os operadores relacionais são usados para comparar dois valores e determinar o relacionamento entre eles. A saída desta avaliação será fornecida com um valor **lógico**: true ou false.

<b>Operador</b>	<b>Uso</b>	<b>Descrição</b>
>	op1 > op2	op1 é maior do que op2
>=	op1 >= op2	op1 é maior ou igual a op2
<	op1 < op2	op1 é menor do que op2
<=	op1 <= op2	op1 é menor ou igual a op2
==	op1 == op2	op1 é igual a op2
!=	op1 != op2	op1 não igual a op2

Tabela 6: Operadores relacionais

O programa a seguir, mostra a utilização destes operadores:

```
public class RelationalDemo {
    public static void main(String[] args) {
        // alguns números
        int i = 37;
        int j = 42;
        int k = 42;
        System.out.println("Variables values...");
        System.out.println("    i = " + i);
        System.out.println("    j = " + j);
        System.out.println("    k = " + k);

        // maior que
        System.out.println("Greater than...");
        System.out.println("    i > j = " + (i > j)); //false
        System.out.println("    j > i = " + (j > i)); //true
        System.out.println("    k > j = " + (k > j)); //false

        // maior ou igual a
        System.out.println("Greater than or equal to...");
        System.out.println("    i >= j = " + (i >= j)); //false
        System.out.println("    j >= i = " + (j >= i)); //true
        System.out.println("    k >= j = " + (k >= j)); //true

        // menor que
        System.out.println("Less than...");
        System.out.println("    i < j = " + (i < j)); //true
        System.out.println("    j < i = " + (j < i)); //false
        System.out.println("    k < j = " + (k < j)); //false

        // menor ou igual a
        System.out.println("Less than or equal to...");
        System.out.println("    i <= j = " + (i <= j)); //true
        System.out.println("    j <= i = " + (j <= i)); //false
    }
}
```

```
        System.out.println("    k <= j = " + (k <= j)); //true

        // igual a
        System.out.println("Equal to...");
        System.out.println("    i == j = " + (i == j)); //false
        System.out.println("    k == j = " + (k == j)); //true

        // diferente
        System.out.println("Not equal to...");
        System.out.println("    i != j = " + (i != j)); //true
        System.out.println("    k != j = " + (k != j)); //false
    }
}
```

A seguir temos a saída deste programa:

```
Variables values...
    i = 37
    j = 42
    k = 42
Greater than...
    i > j = false
    j > i = true
    k > j = false
Greater than or equal to...
    i >= j = false
    j >= i = true
    k >= j = true
Less than...
    i < j = true
    j < i = false
    k < j = false
Less than or equal to...
    i <= j = true
    j <= i = false
    k <= j = true
Equal to...
    i == j = false
    k == j = true
Not equal to...
    i != j = true
    k != j = false
```

## 13. Operadores Lógicos

Operadores lógicos avaliam um ou mais operandos **lógicos** que geram um único valor final **true** ou **false** como resultado da expressão. São seis os operadores lógicos: && (e lógico), & (e binário), || (ou lógico), | (ou binário), ^ (ou exclusivo binário) e ! (negação).

A operação básica para um operador lógico é:

`x1 op x2`

Onde x1 e x2 podem ser expressões, variáveis ou constantes lógicas, e op pode tanto ser &&, &, ||, | ou ^.

### 13.1. && (e lógico) e & (e binário)

<b>x1</b>	<b>x2</b>	<b>Resultado</b>
VERDADEIRO	VERDADEIRO	VERDADEIRO
VERDADEIRO	FALSO	FALSO
FALSO	VERDADEIRO	FALSO
FALSO	FALSO	FALSO

*Tabela 7: Tabela para && e &*

A diferença básica do operador && para & é que o && suporta uma avaliação de curto-circuito (ou avaliação parcial), enquanto que o & não. O que isso significa?

Dado o exemplo:

`exp1 && exp2`

o operador **e lógico** irá avaliar a expressão exp1, e, imediatamente, retornará um valor **false** se a operação exp1 for falsa. Se a expressão exp1 resultar em um valor **false** o operador nunca avaliará a expressão exp2, pois o valor de toda a expressão será falsa mesmo que o resultado isolado de exp2 seja verdadeiro. Já o operador & sempre avalia as duas partes da expressão, mesmo que a primeira tenha o valor **false**.

O programa a seguir, mostra a utilização destes operadores:

```
public class TestAND {
    public static void main( String[] args ) {
        int i = 0;
        int j = 10;
        boolean test = false;

        // demonstração do operador &&
        test = (i > 10) && (j++ > 9);
        System.out.println(i);
        System.out.println(j);
        System.out.println(test);

        // demonstração do operador &
        test = (i > 10) & (j++ > 9);
        System.out.println(i);
        System.out.println(j);
        System.out.println(test);
    }
}
```

```
    }
}
```

Como resultado, o programa produzirá a seguinte saída:

```
0
10
false
0
11
false
```

Note que o comando `j++`, na linha contendo `&&`, nunca será executado, pois o operador não o avalia, visto que a primeira parte da expressão (`i > 10`) retorna um valor booleano **false**.

### 13.2. || ( ou lógico) e | ( ou binário)

<i><b>x1</b></i>	<i><b>x2</b></i>	<i><b>Resultado</b></i>
VERDADEIRO	VERDADEIRO	VERDADEIRO
VERDADEIRO	FALSO	VERDADEIRO
FALSO	VERDADEIRO	VERDADEIRO
FALSO	FALSO	FALSO

*Tabela 8: Tabela para || e |*

A diferença básica entre os operadores `||` e `|`, é que, semelhante ao operador `&&`, o `||` também suporta a avaliação parcial. O que isso significa?

Dada a expressão,

```
exp1 || exp2
```

o operador **ou lógico** irá avaliar a expressão `exp1`, e, imediatamente, retornará um valor lógico **true** para toda a expressão se a primeira parte for avaliada como verdadeira. Se a expressão `exp1` resultar em **verdadeira** a segunda parte `exp2` nunca será avaliada, pois o valor final da expressão será **true** independentemente do resultado da segunda expressão.

O programa a seguir, mostra a utilização destes operadores:

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

        int i = 0;
        int j = 10;
        boolean test = false;

        // demonstração do operador ||
        test = (i < 10) || (j++ > 9);
        System.out.println(i);
        System.out.println(j);
        System.out.println(test);

        // demonstração do operador |
        test = (i < 10) | (j++ > 9);
        System.out.println(i);
        System.out.println(j);
        System.out.println(test);
    }
}
```



```
}
```

Como resultado, o programa produzirá a seguinte saída:

```
0
10
true
0
11
true
```

Note que a expressão `j++` nunca será avaliada na instrução que usa o operador `||`, pois a primeira parte da expressão (`i<10`) já retorna **true** como valor final da expressão.

### 13.3. ^ (ou exclusivo binário)

<i><b>x1</b></i>	<i><b>x2</b></i>	<i><b>Resultado</b></i>
VERDADEIRO	VERDADEIRO	FALSO
VERDADEIRO	FALSO	VERDADEIRO
FALSO	VERDADEIRO	VERDADEIRO
FALSO	FALSO	FALSO

*Tabela 9: Tabela para o operador ^*

O resultado de uma expressão usando o operador **ou exclusivo binário** terá um valor **true** somente se uma das expressões for verdadeira e a outra falsa. Note que ambos os operandos são necessariamente avaliados pelo operador ^.

O programa a seguir, mostra a utilização deste operador:

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

        boolean val1 = true;
        boolean val2 = true;
        System.out.println(val1 ^ val2);

        val1 = false;
        val2 = true;
        System.out.println(val1 ^ val2);

        val1 = false;
        val2 = false;
        System.out.println(val1 ^ val2);

        val1 = true;
        val2 = false;
        System.out.println(val1 ^ val2);
    }
}
```

Como resultado, o programa produzirá a seguinte saída:

```
false
true
false
true
```

### 13.4. ! (negação)

<b>x1</b>	<b>Resultado</b>
VERDADEIRO	FALSO
FALSO	VERDADEIRO

*Tabela 10: Tabela para o operador !*

O operador de **negação** inverte o resultado lógico de uma expressão, variável ou constante, ou seja, o que era verdadeiro será falso e vice-versa.

O programa a seguir, mostra a utilização deste operador:

```
public class TestNOT {  
    public static void main( String[] args ){  
        boolean val1 = true;  
        boolean val2 = false;  
        System.out.println(!val1);  
        System.out.println(!val2);  
    }  
}
```

Como resultado, o programa produzirá a seguinte saída:

```
false  
true
```

## 14. Operador Condicional ( ?: )

O operador **condicional** é também chamado de operador **ternário**. Isto significa que ele tem 3 argumentos que juntos formam uma única expressão condicional. A estrutura de uma expressão utilizando um operador condicional é a seguinte:

```
exp1?exp2:exp3
```

Onde exp1 é uma expressão lógica que deve retornar **true** ou **false**. Se o valor de exp1 for verdadeiro, então, o resultado será a expressão exp2, caso contrário, o resultado será exp3.

O programa a seguir, mostra a utilização deste operador:

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

        String      status = "";
        int    grade = 80;

        //status do aluno
        status = (grade >= 60)?"Passed":"Fail";

        //print status
        System.out.println( status );
    }
}
```

Como resultado, o programa produzirá a seguinte saída:

```
Passed
```

Veremos na Figura 2 um fluxograma que demonstra como o operador **condicional** funciona.

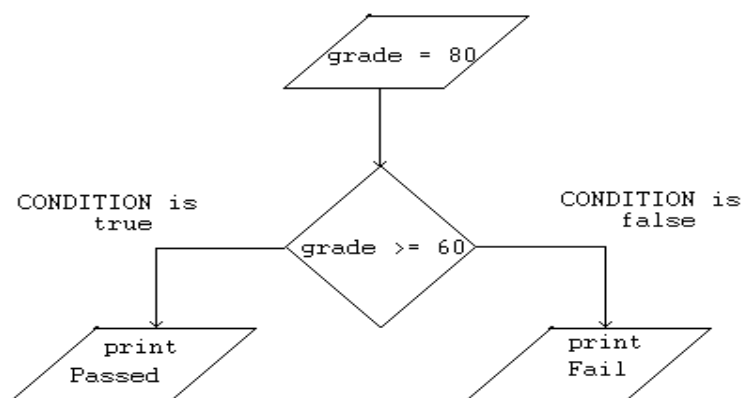


Figura 2: Fluxograma utilizando o operador **condicional**

Veremos outro programa que também utiliza o operador condicional:

```
public class ConditionalOperator {
    public static void main( String[] args ){
        int    score = 0;
        char    answer = 'a';
    }
}
```

```
        score = (answer == 'a') ? 10 : 0;  
        System.out.println("Score = " + score );  
    }  
}
```

Como resultado, o programa produzirá a seguinte saída:

```
Score = 10
```

## 15. Precedência de Operadores

A precedência serve para indicar a ordem na qual o compilador interpretará os diferentes tipos de operadores, para que ele sempre tenha como saída um resultado coerente e não ambíguo.

Ordem	Operador
1	( ) parênteses
2	++ pós-incremento e -- pós-decremento
3	++ pré-incremento e -- pré-decremento
4	! negação lógica
5	* multiplicação e / divisão
6	% resto da divisão
7	+ soma e - subtração
8	< menor que, <= menor ou igual, > maior que e >= maior ou igual
9	== igual e != não igual
10	& e binário
11	ou binário
12	^ ou exclusivo binário
13	&& e lógico
14	ou lógico
15	?: condicional
16	= atribuição

Tabela 11: Precedência de operadores

No caso de dois operadores com mesmo nível de precedência, terá prioridade o que estiver mais à esquerda da expressão. Dada uma expressão complexa como:

$$6\%2*5+4/2+88-10$$

O ideal seria fazer uso de parênteses para reescrevê-la de maneira mais clara:

$$((6\%2)*5)+(4/2)+88-10$$

### Dicas de programação:

1. Para evitar confusão na avaliação de suas expressões matemáticas, deixe-as o mais simples possível e use parênteses.

## 16. Exercícios

### 16.1. Declarar e mostrar variáveis

Dada a tabela abaixo, declare as variáveis que se seguem de acordo com seus tipos correspondentes e valores iniciais. Exiba o nomes e valor das variáveis.

<i>Nome das Variáveis</i>	<i>Tipo do dado</i>	<i>Valor inicial</i>
number	integer	10
letter	character	a
result	boolean	true
str	String	hello

O resultado esperado do exercício é:

```
number = 10
letter = a
result = true
str = hello
```

### 16.2. Obter a média entre três números

Crie um programa que obtenha a média de 3 números. Considere o valor para os três números como sendo 10, 20 e 45. O resultado esperado do exercício é:

```
número 1 com o valor 10
número 2 com o valor 20
número 3 com o valor 45
A média é 25
```

### 16.3. Exibir o maior valor

Dados três números, crie um programa que exiba na tela o maior dentre os números informados. Use o operador **?:** que já foi estudado nesta sessão (**dica:** será necessário utilizar dois operadores **?:** para se chegar ao resultado). Por exemplo, dados os números 10, 23 e 5, o resultado esperado do exercício deve ser:

```
número 1 com o valor 10
número 2 com o valor 23
número 3 com o valor 5
O maior número é 23
```

### 16.4. Precedência de operadores

Dadas as expressões abaixo, reescreva-as utilizando parênteses de acordo com a forma como elas são interpretadas pelo compilador.

1.  $a / b ^ c ^ d - e + f - g * h + i$
2.  $3 * 10 * 2 / 15 - 2 + 4 ^ 2 ^ 2$
3.  $r ^ s * t / u - v + w ^ x - y++$

## Parceiros que tornaram JEDI™ possível



### **Instituto CTS**

Patrocinador do DFJUG.

### **Sun Microsystems**

Fornecimento de servidor de dados para o armazenamento dos vídeo-aulas.

### **Java Research and Development Center da Universidade das Filipinas**

Criador da Iniciativa JEDI™.

### **DFJUG**

Detentor dos direitos do JEDI™ nos países de língua portuguesa.

### **Banco do Brasil**

Disponibilização de seus *telecentros* para abrigar e difundir a Iniciativa JEDI™.

### **Politec**

Suporte e apoio financeiro e logístico a todo o processo.

### **Borland**

Apoio internacional para que possamos alcançar os outros países de língua portuguesa.

### **Instituto Gaudium/CNBB**

Fornecimento da sua infra-estrutura de hardware de seus servidores para que os milhares de alunos possam acessar o material do curso simultaneamente.