



Quem se prepara, não para.



Linguagens de Programação

3º período

Prof. Dr. João Paulo Aramuni

Sumário

- Java
 - Introdução
 - Vantagens e desvantagens
 - Tipos primitivos
 - Operadores
 - Estruturas condicionais
 - Estruturas de repetição

Sumário

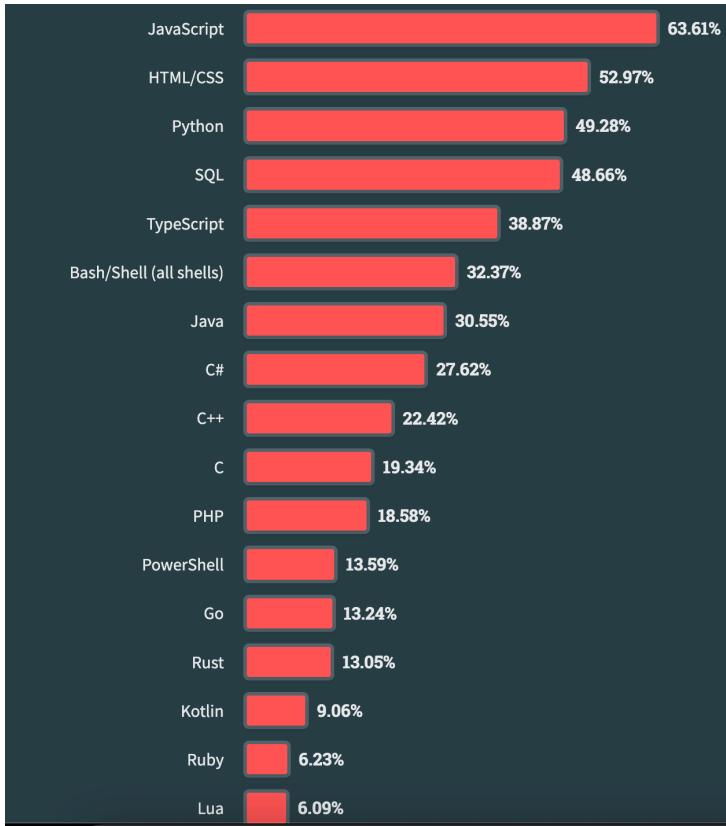
- Java
 - **Introdução**
 - Vantagens e desvantagens
 - Tipos primitivos
 - Operadores
 - Estruturas condicionais
 - Estruturas de repetição

- Java é uma linguagem simples, **orientada a objetos**, robusta, segura, independente de plataforma, interpretada e compilada, de alto desempenho.
- Java é uma linguagem de propósito geral:
 - Pode ser usada para a construção de pequenos programas (Applets) que rodam em browsers na **Web**.
 - Pode ser usada para a construção de complexas aplicações **Desktop**.

- Java possui sintaxe baseada em C.
- Tipos de dados básicos similares a C.
- Gerenciamento de memória automático.
- Vasta biblioteca que inclui recursos para Web,
Interfaces Gráficas (GUI), Banco de Dados, Redes, etc.
- Linguagem de uso gratuito.
 - Muito comum de se encontrar em projetos do governo.

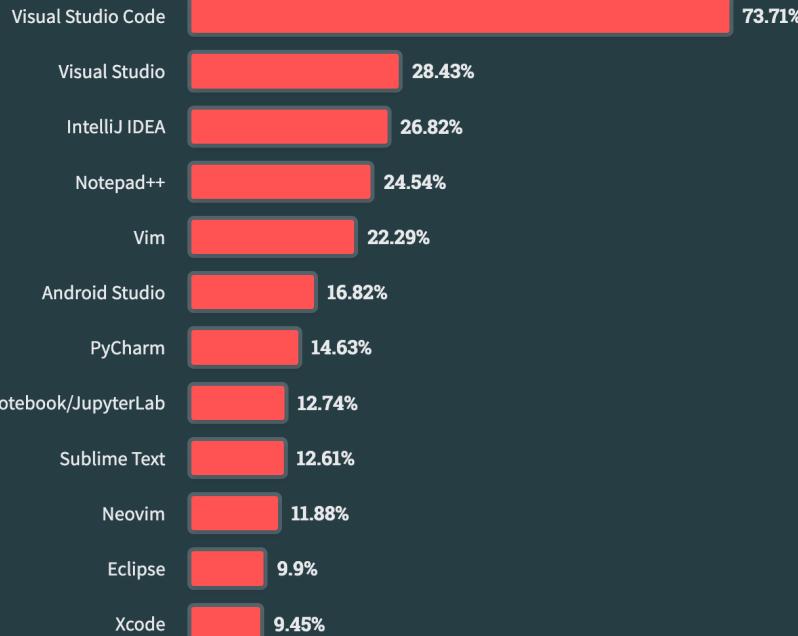


Linguagens



- **Java** é a 7^a linguagem mais popular do mundo de acordo com a pesquisa do Stack Overflow 2023





- **Eclipse** é a 11^a IDE mais popular do mundo de acordo com a pesquisa do Stack Overflow 2023



Java JDK 17

- [Download] <https://www.oracle.com/java/technologies/downloads/#java17>
- [Docs] <https://docs.oracle.com/en/java/javase/17/>



Eclipse IDE

- [Download] <https://www.eclipse.org/downloads/packages/installer>
- [Docs] <https://www.eclipse.org/documentation/>

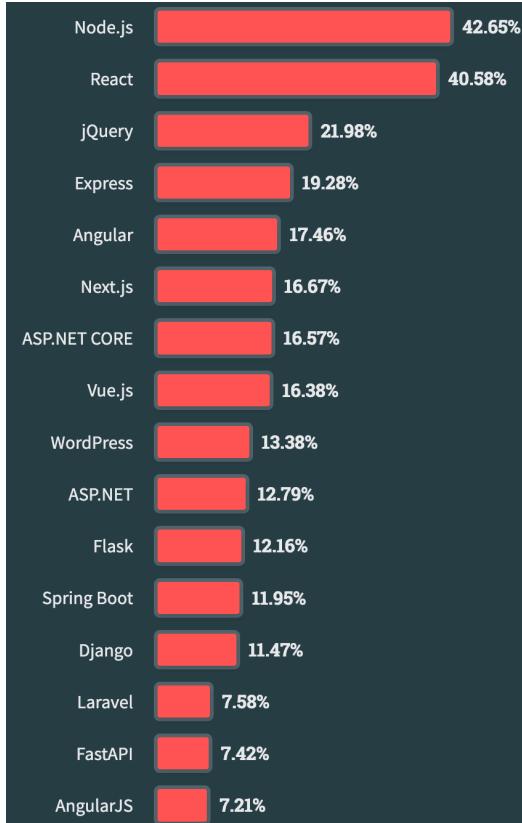


Java

- Alguns frameworks famosos:



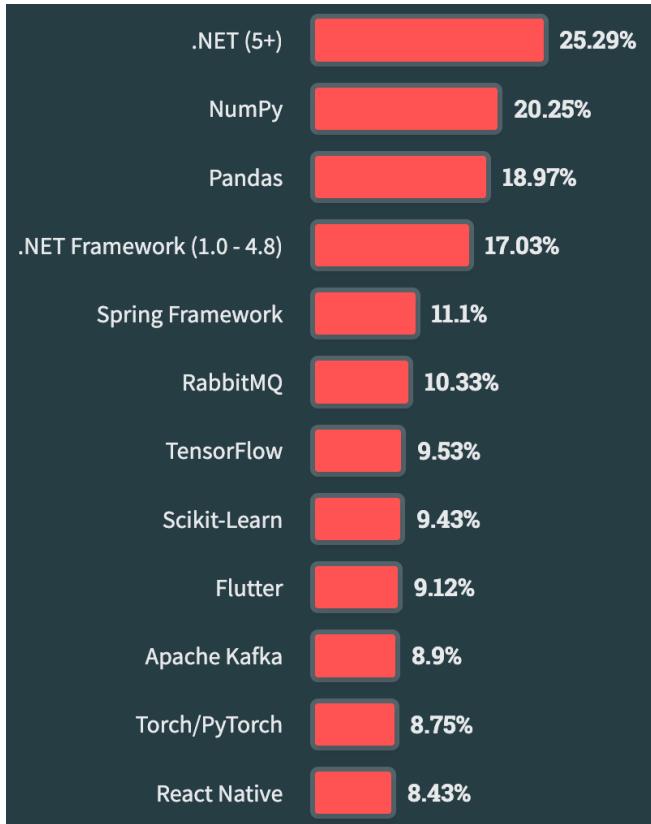
Web Frameworks



- **Spring Boot** é o 12º web framework mais popular do mundo de acordo com a pesquisa do Stack Overflow 2023



Outros Frameworks e libs



- **Spring Framework** é o 5º framework mais popular do mundo de acordo com a pesquisa do Stack Overflow 2023



- **JSE** (Java Standard Edition): Também conhecida como Java SE, é a plataforma Java básica que fornece as bibliotecas fundamentais e as ferramentas para desenvolvimento de aplicativos Java em geral.
 - É adequada para desenvolver aplicativos desktop, aplicativos de linha de comando, pequenas aplicações web e sistemas independentes.
 - Inclui todas as funcionalidades centrais da linguagem Java, como manipulação de strings, gerenciamento de exceções, threads, entrada/saída, entre outras.
 - É utilizada para criar aplicativos Java que podem ser executados em computadores e servidores sem a necessidade de uma infraestrutura de servidor complexa.

- **JEE** (Java Platform, Enterprise Edition): Também conhecida como Java EE, é uma extensão da plataforma JSE e é projetada para desenvolver aplicativos corporativos e de larga escala.
 - É voltada para a construção de aplicativos web, aplicações empresariais e sistemas distribuídos em servidores de aplicativos.
 - Inclui todas as funcionalidades da JSE e adiciona um conjunto de APIs (Java APIs) adicionais para facilitar o desenvolvimento de aplicativos corporativos, como APIs para acesso a bancos de dados, gerenciamento de transações, serviços web, segurança, entre outras.
 - Permite o desenvolvimento de aplicativos escaláveis, robustos e seguros, com recursos avançados para atender às demandas empresariais.

- **JME** (Java Platform, Micro Edition): Também conhecida como Java ME, é uma plataforma Java projetada para dispositivos com recursos limitados, como dispositivos móveis, dispositivos embarcados e sistemas embarcados.
 - É uma versão mais leve do Java que oferece apenas um subconjunto das funcionalidades disponíveis na JSE.
 - Permite o desenvolvimento de aplicativos para dispositivos com recursos restritos de memória e processamento, como telefones celulares, PDAs (Assistentes Pessoais Digitais) e dispositivos IoT (Internet of Things).

- Programas Java são compostos por classes armazenadas em arquivos texto com extensão **.java**.
- Estes programas podem ser editados por um editor de texto convencional e são armazenados em disco como um arquivo convencional.
- Por meio do processo de compilação, um código objeto é gerado a partir do código fonte. Este código objeto, denominado **bytecode**, é armazenado em disco como um ou mais arquivos de extensão **.class**.

- Uma vez gerado o código objeto Java (bytecodes), o mesmo é interpretado por uma máquina virtual (JVM), que traduz cada instrução do bytecode para uma instrução que o computador nativo possa entender.
- A máquina virtual Java (JVM) carrega os bytecodes na memória, verifica os bytecodes e os interpreta para a arquitetura da máquina real.
- A JVM na verdade é um emulador para uma máquina real.

- A plataforma do Java é formada por três partes: JVM, Linguagem Java e Bibliotecas de Classes Java (API).
- API (Application Programming Interface) Java
 - Complementa a linguagem Java com um conjunto de rotinas específicas para diversas tecnologias.
 - Possui milhares de métodos.

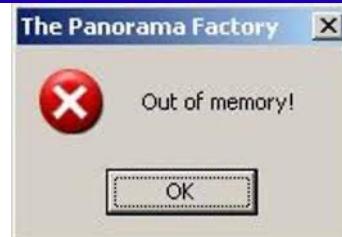
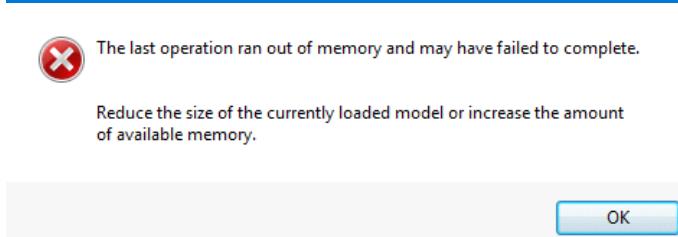
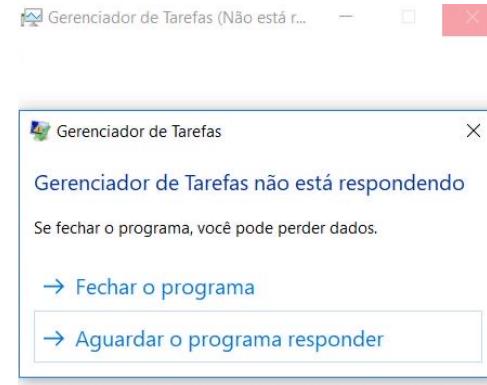
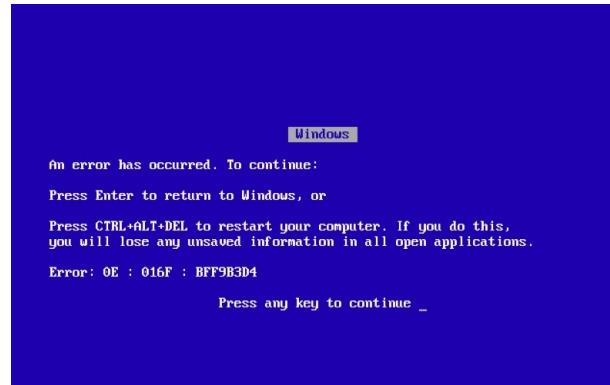
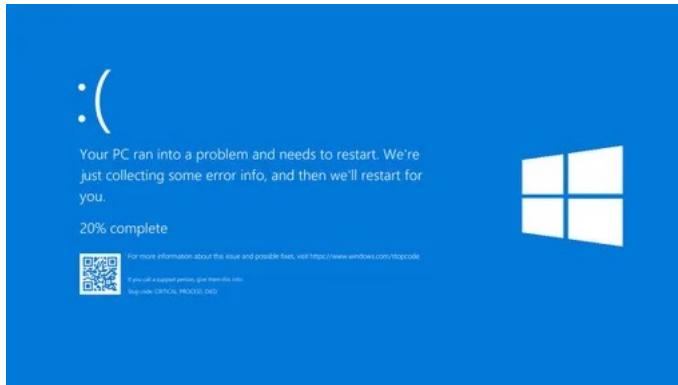
- Distribuições:
 - **JRE:** O Java Runtime Environment contém tudo aquilo que um usuário comum precisa para executar uma aplicação Java (JVM e bibliotecas), como o próprio nome já diz é o “Ambiente de execução Java”.
 - **JDK:** O Java Development Kit é composto pelo JRE, o compilador javac e as APIs Java.

- **Garbage Collector:**

- O Java possui um coletor de lixo. Um processo usado para a automação do gerenciamento de memória.
 - Em linguagens anteriores, como a Linguagem C, a pessoa programadora era encarregada de liberar essa memória manualmente. Como por exemplo por meio do comando **malloc**.

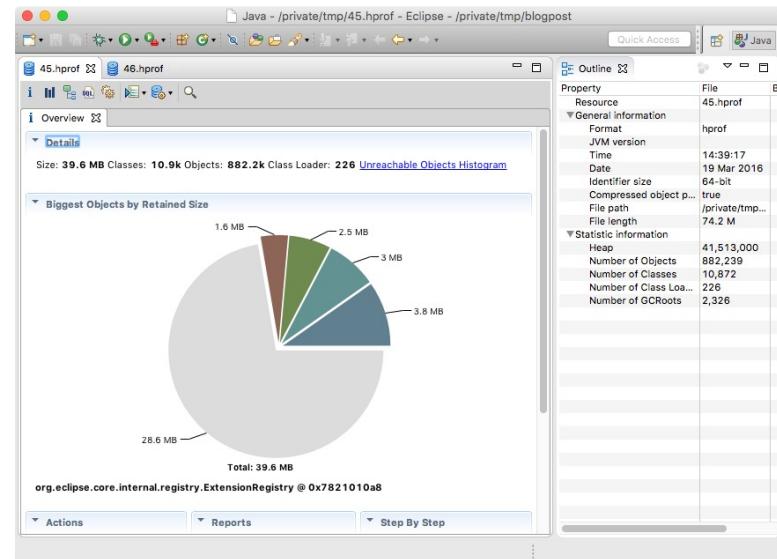
- **Garbage Collector:**
 - O Garbage Collector se resume em uma Thread em baixa prioridade.
 - Em Java a responsabilidade pela alocação de memória fica totalmente a critério da JVM.
 - Apenas os objetos de memória que não são mais necessários para sua aplicação são coletados.

- Resultado do vazamento de memória (memory leaks):

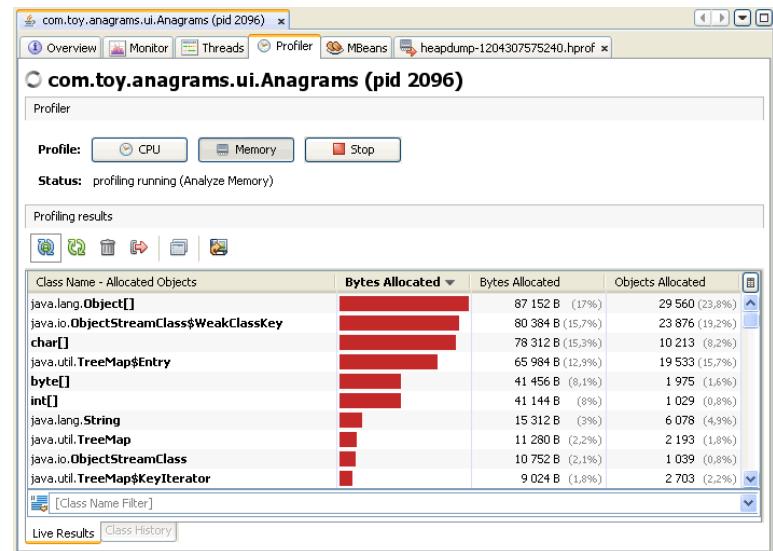
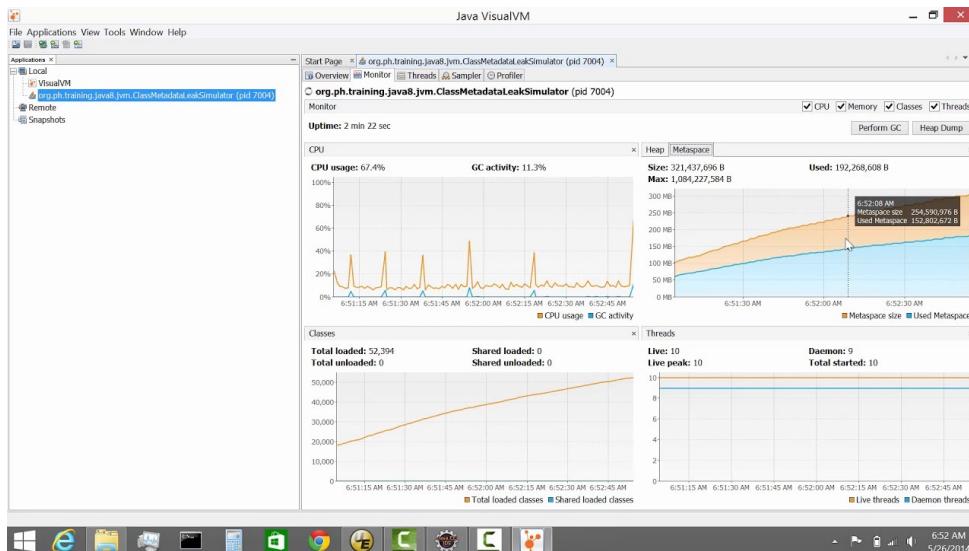


- Existem ferramentas e plugins para monitorar a memória utilizada por cada parte do seu sistema escrito em Java. Por exemplo: **Eclipse MAT**
 - <https://projects.eclipse.org/projects/tools.mat>

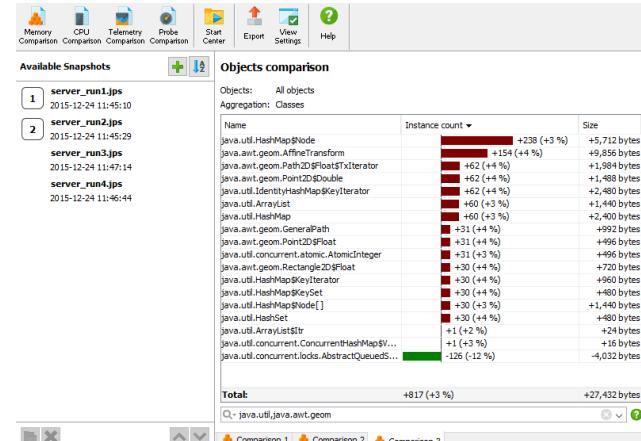
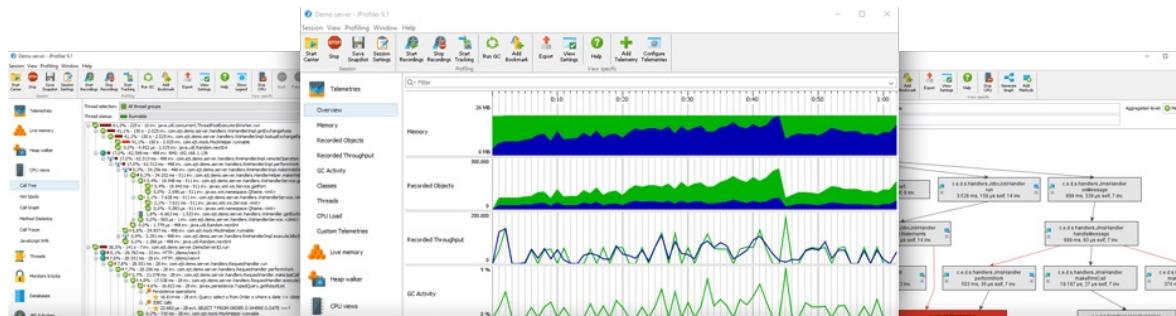
No Eclipse Mat (Memory Analiser Tool), cada fatia da pizza pode representar uma tela, uma classe, ou alguma parte do sistema que você queira monitorar o uso de memória.



- Existem ferramentas e plugins para monitorar a memória utilizada por cada parte do seu sistema escrito em Java. Por exemplo: **VisualVM**
 - <https://visualvm.github.io/download.html>



- Existem ferramentas e plugins para monitorar a memória utilizada por cada parte do seu sistema escrito em Java. Por exemplo: **JProfiler**
 - <https://www.ej-technologies.com/products/jprofiler/overview.html>



Sumário

- Java
 - Introdução
 - **Vantagens e desvantagens**
 - Tipos primitivos
 - Operadores
 - Estruturas condicionais
 - Estruturas de repetição

Java – Vantagens e Desvantagens

Vantagens

Portabilidade: O Java é uma linguagem de programação "write once, run anywhere" (escreva uma vez, execute em qualquer lugar), graças ao ambiente de máquina virtual Java (JVM). Isso permite que os programas Java sejam executados em diferentes sistemas operacionais sem a necessidade de recompilação.

Orientação a objetos: O Java é uma linguagem de programação orientada a objetos, o que facilita a criação de programas organizados, modulares e reutilizáveis.

Comunidade e bibliotecas: O Java tem uma comunidade de desenvolvedores vasta e ativa, o que resulta em uma grande quantidade de bibliotecas, frameworks e recursos disponíveis para acelerar o desenvolvimento de aplicativos.

Segurança: O Java é projetado com foco em segurança. A execução de código é realizada dentro da JVM, o que ajuda a isolar o ambiente do sistema operacional e proteger contra ameaças maliciosas.

Multithreading: O Java possui suporte nativo a programação multithread, permitindo que os desenvolvedores criem aplicativos que aproveitam eficientemente os sistemas multiprocessados.

Robustez: Java é conhecido por sua robustez e capacidade de tratamento de exceções. Essas características tornam a linguagem mais resistente a erros e falhas.

Desenvolvimento empresarial: O Java é amplamente utilizado no desenvolvimento de aplicativos empresariais devido à sua escalabilidade e desempenho confiável.

Desvantagens

Lentidão: A execução do código Java pode ser mais lenta em comparação com linguagens compiladas nativamente, pois depende da JVM para a execução.

Uso intensivo de memória: A JVM consome uma quantidade considerável de memória, o que pode ser uma desvantagem para aplicativos que precisam ser executados em sistemas com recursos limitados.

Curva de aprendizado: Para alguns desenvolvedores, especialmente aqueles que estão acostumados com linguagens mais simples, a curva de aprendizado do Java pode ser íngreme devido à complexidade da linguagem e suas peculiaridades.

Verbosidade: Java pode ser mais verboso em comparação com algumas linguagens modernas, o que significa que é necessário escrever mais código para realizar tarefas simples.

Sobrecarga de pacotes e bibliotecas: Devido à grande quantidade de bibliotecas disponíveis, pode ser desafiador escolher a melhor opção e evitar a sobrecarga de pacotes em projetos menores.

Ausência de recursos modernos em versões mais antigas: Algumas versões mais antigas do Java podem não ter acesso a recursos e melhorias mais recentes, o que pode restringir o desenvolvimento em ambientes mais antigos.

Sumário

- Java
 - Introdução
 - Vantagens e desvantagens
 - **Tipos primitivos**
 - Operadores
 - Estruturas condicionais
 - Estruturas de repetição

- **Tipos primitivos:**

- O termo primitivo é usado aqui para indicar que esses tipos não são objetos no sentido da orientação a objetos e sim valores binários comuns.
- Há 8 tipos primitivos (pré-definidos):

Números Inteiros
byte 1 byte
short 2 bytes
int 4 bytes (mais usado)
long 8 bytes

Números em Ponto Flutuante
float 4 bytes
double 8 bytes (duas vezes a precisão do tipo float) (mais usado)

- **Tipos primitivos:**

- Há 8 tipos primitivos (pré-definidos):
 - Caractere: **char** 2 bytes
 - Caracteres Unicode
 - Permite até 65536 caracteres (atualmente usados cerca de 35000).
 - Primeiros 255 caracteres idênticos ao código ASCII / ANSI.
 - Representado por aspas simples 'A'.
 - Caracteres especiais:
 - \b (backspace)
 - \t (tab)
 - \n (linefeed)
 - \r (carriage return)
 - \" (aspas duplas)
 - \' (apóstrofe)
 - \\ (barra invertida)

- **Tipos primitivos:**

- Java foi projetada para uso mundial. Logo, tem que usar um conjunto de caracteres que possa representar os idiomas do mundo todo. O **Unicode** é o conjunto de caracteres padrão projetado especialmente para esse fim.

32		48	0	64	@	80	P	96	'		112	p
33	!	49	1	65	A	81	Q	97	a		113	q
34	"	50	2	66	B	82	R	98	b		114	r
35	#	51	3	67	C	83	S	99	c		115	s
36	\$	52	4	68	D	84	T	100	d		116	t
37	%	53	5	69	E	85	U	101	e		117	u
38	&	54	6	70	F	86	V	102	f		118	v
39	'	55	7	71	G	87	W	103	g		119	w
40	(56	8	72	H	88	X	104	h		120	x
41)	57	9	73	I	89	Y	105	i		121	y
42	*	58	:	74	J	90	Z	106	j		122	z
43	+	59	;	75	K	91	[107	k		123	{
44	,	60	<	76	L	92	\	108	l		124	
45	-	61	=	77	M	93]	109	m		125	}
46	.	62	>	78	N	94	^	110	n		126	~
47	/	63	?	79	O	95	_	111	o		127	

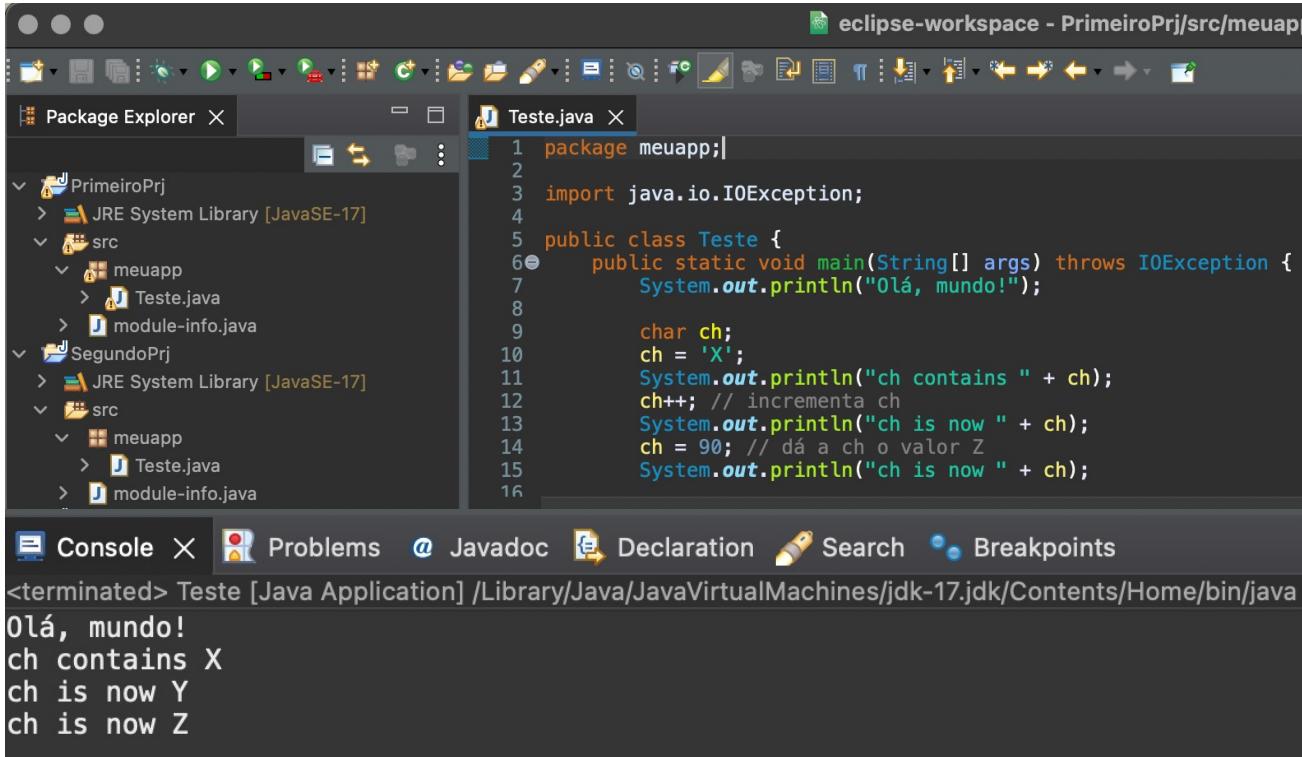
- **Tipos primitivos:**

- Há 8 tipos primitivos (pré-definidos):
 - Lógico
 - **boolean** 1 byte (valores verdadeiros (true) e falso (false)).
 - Repare que é com b minúsculo.
 - Boolean com B maiúsculo trata-se da classe Boolean.
 - Ao contrário do tipo primitivo, a classe Boolean trata **nulos**, além de outras funcionalidades. O mesmo acontece com a classe Integer.
 - O restante são objetos (exceto array).
 - Sendo assim, os tipos primitivos são:
 - boolean, byte, char, short, int, long, float, e double.

- **Tipos primitivos:**

Tipos	Primitivo	Valores possíveis		Valor Padrão	Tamanho	Exemplo
		Menor	Maior			
Inteiro	byte	-128	127	0	8 bits	byte ex1 = (byte)1;
	short	-32768	32767	0	16 bits	short ex2 = (short)1;
	int	-2.147.483.648	2.147.483.647	0	32 bits	int ex3 = 1;
	long	-9.223.372.036.854.770.000	9.223.372.036.854.770.000	0	64 bits	long ex4 = 1l;
Ponto Flutuante	float	-1,4024E-37	3.40282347E + 38	0	32 bits	float ex5 = 5.50f;
	double	-4,94E-307	1.79769313486231570E + 308	0	64 bits	double ex6 = 10.20d; ou double ex6 = 10.20;
Caractere	char	0	65535	\0	16 bits	char ex7 = 194; ou char ex8 = 'a';
Booleano	boolean	false	true	false	1 bit	boolean ex9 = true;

- **Tipos primitivos:**



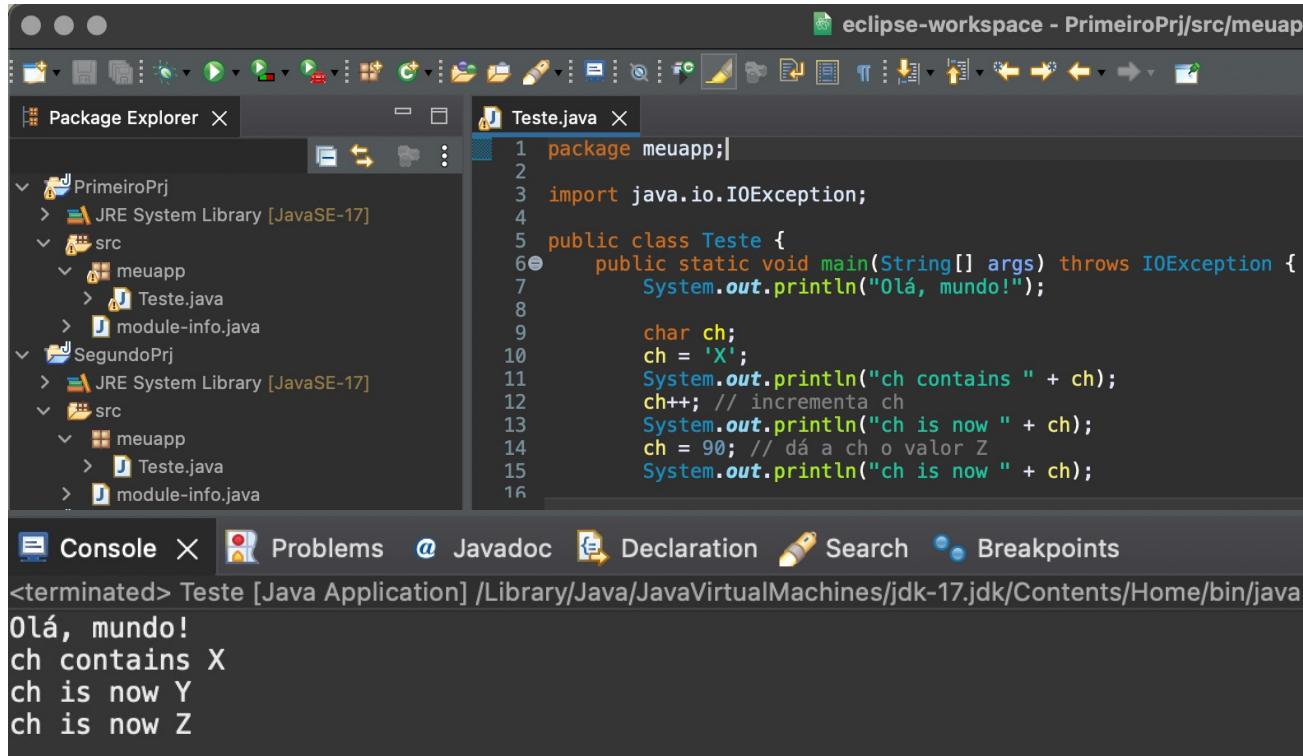
The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows two projects: "PrimeiroPrj" and "SegundoPrj". Each project has a "src" folder containing a "meuapp" package with a "Teste.java" file and a "module-info.java" file.
- Code Editor:** The "Teste.java" file is open, displaying the following Java code:

```
1 package meuapp;
2
3 import java.io.IOException;
4
5 public class Teste {
6     public static void main(String[] args) throws IOException {
7         System.out.println("Olá, mundo!");
8
9         char ch;
10        ch = 'X';
11        System.out.println("ch contains " + ch);
12        ch++; // incrementa ch
13        System.out.println("ch is now " + ch);
14        ch = 90; // dá a ch o valor Z
15        System.out.println("ch is now " + ch);
16    }
17}
```
- Console:** The output window shows the execution results:

```
<terminated> Teste [Java Application] /Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java
Olá, mundo!
ch contains X
ch is now Y
ch is now Z
```

- **Tipos primitivos:**



The screenshot shows the Eclipse IDE interface. The Package Explorer view on the left displays two Java projects: 'PrimeiroPrj' and 'SegundoPrj'. The 'src' folder under 'PrimeiroPrj' contains a package named 'meuapp' which includes a file 'Teste.java'. The 'Console' view at the bottom shows the output of running the 'Teste' class, displaying the text 'Olá, mundo!', followed by three lines of output from the 'ch' variable: 'ch contains X', 'ch is now Y', and 'ch is now Z'. The central workspace shows the code for 'Teste.java':

```
1 package meuapp;
2
3 import java.io.IOException;
4
5 public class Teste {
6     public static void main(String[] args) throws IOException {
7         System.out.println("Olá, mundo!");
8
9         char ch;
10        ch = 'X';
11        System.out.println("ch contains " + ch);
12        ch++; // increments ch
13        System.out.println("ch is now " + ch);
14        ch = 90; // sets ch to value Z
15        System.out.println("ch is now " + ch);
16    }
17}
```

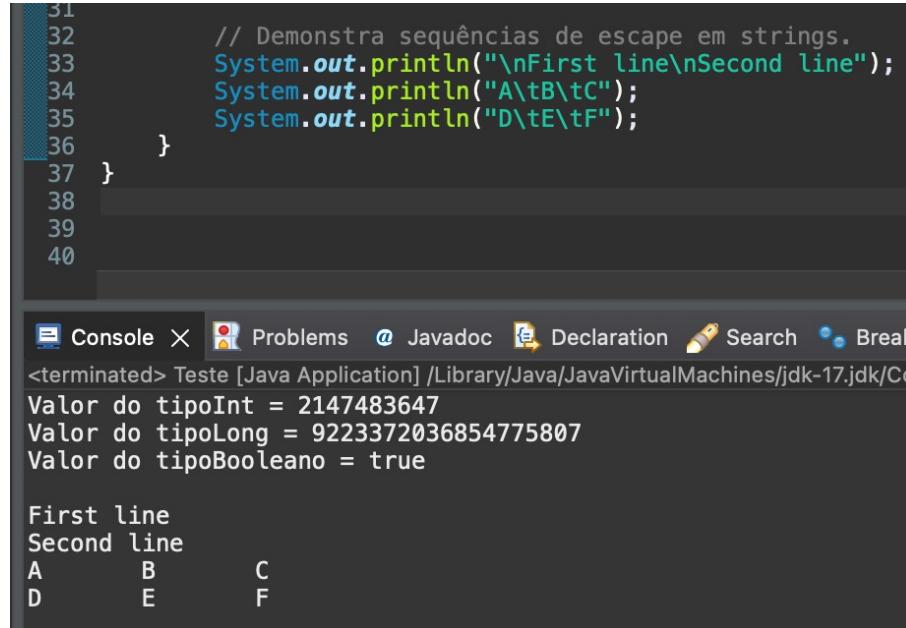
O método **main** em Java é o ponto de entrada principal para um programa Java.

Quando você executa um programa Java, o sistema procura pelo método **main** para iniciar a execução do programa.

- **Tipos primitivos:**

```
byte tipoByte = 127;
short tipoShort = 32767;
char tipoChar = 'C';
float tipoFloat = 2.6f;
double tipoDouble = 3.59;
int tipoInt = 2147483647;
long tipoLong = 9223372036854775807L;
boolean tipoBooleano = true;
System.out.println("Valor do tipoByte = " + tipoByte);
System.out.println("Valor do tipoShort = " + tipoShort);
System.out.println("Valor do tipoChar = " + tipoChar);
System.out.println("Valor do tipoFloat = " + tipoFloat);
System.out.println("Valor do tipoDouble = " + tipoDouble);
System.out.println("Valor do tipoInt = " + tipoInt);
System.out.println("Valor do tipoLong = " + tipoLong);
System.out.println("Valor do tipoBooleano = " + tipoBooleano);
```

- Java dá suporte a outro tipo de literal: o **string**. Um string é um conjunto de caracteres inserido em aspas duplas. Por exemplo, “Aula do Ara é 10”.



The screenshot shows a Java application running in an IDE. The code in the editor is:

```
31     // Demonstra sequências de escape em strings.
32     System.out.println("\nFirst line\nSecond line");
33     System.out.println("A\tB\tC");
34     System.out.println("D\tE\tF");
35 }
36 }
37 }
38
39
40
```

The console output window shows the following results:

```
Console × Problems Javadoc Declaration Search Break
<terminated> Teste [Java Application] /Library/Java/JavaVirtualMachines/jdk-17.jdk/Co
Valor do tipoInt = 2147483647
Valor do tipoLong = 9223372036854775807
Valor do tipoBooleano = true

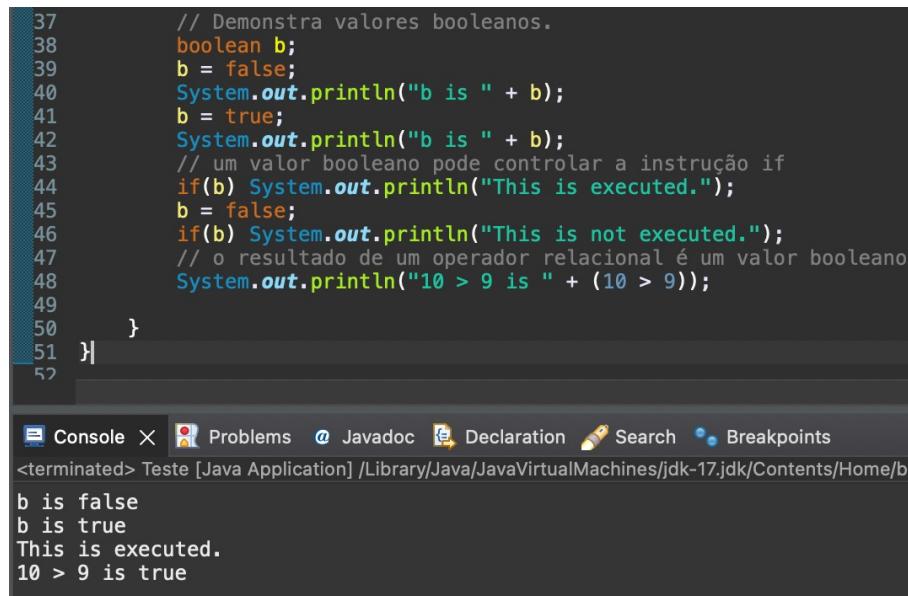
First line
Second line
A      B      C
D      E      F
```

- Java dá suporte a outro tipo de literal: o **string**. Um string é um conjunto de caracteres inserido em aspas duplas. Por exemplo, “Aula do Ara é 10”.

Sequência de escape	Descrição
\'	Aspas simples
\\"	Aspas duplas
\\"\\	Barra invertida
\r	Retorno do cursor
\n	Nova linha
\f	Avanço de página
\t	Tabulação horizontal
\b	Retrocesso
\ddd	Constante octal (onde ddd é uma constante octal)
\uxxxx	Constante hexadecimal (onde xxxx é uma constante hexadecimal)

- Exemplo com boolean

- O tipo **boolean** representa os valores **verdadeiro** e **falso**.
- Java define os valores verdadeiro e falso usando as palavras reservadas **true** e **false**.



The screenshot shows a Java code editor and a terminal window. The code editor displays the following Java code:

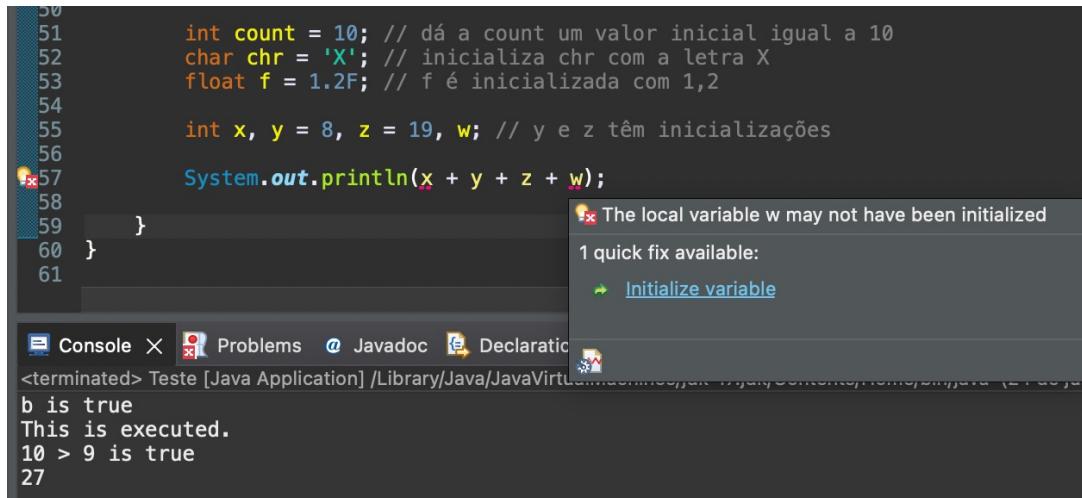
```
37     // Demonstra valores booleanos.
38     boolean b;
39     b = false;
40     System.out.println("b is " + b);
41     b = true;
42     System.out.println("b is " + b);
43     // um valor booleano pode controlar a instrução if
44     if(b) System.out.println("This is executed.");
45     b = false;
46     if(b) System.out.println("This is not executed.");
47     // o resultado de um operador relacional é um valor booleano
48     System.out.println("10 > 9 is " + (10 > 9));
49
50 }
51 }
52 }
```

The terminal window below shows the output of the code execution:

```
Console X Problems Javadoc Declaration Search Breakpoints
<terminated> Teste [Java Application] /Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin
b is false
b is true
This is executed.
10 > 9 is true
```

- Inicialização de variável

- Em geral, devemos dar um valor à variável antes de usá-la.
- Uma maneira de dar um valor a uma variável é por uma instrução de atribuição.
- Exemplo de uso de constantes como inicializadores:



The screenshot shows a Java code editor with the following code:

```
50     int count = 10; // dá a count um valor inicial igual a 10
51     char chr = 'X'; // inicializa chr com a letra X
52     float f = 1.2F; // f é inicializada com 1,2
53
54
55     int x, y = 8, z = 19, w; // y e z têm inicializações
56
57     System.out.println(x + y + z + w);
58
59 }
60 }
61 }
```

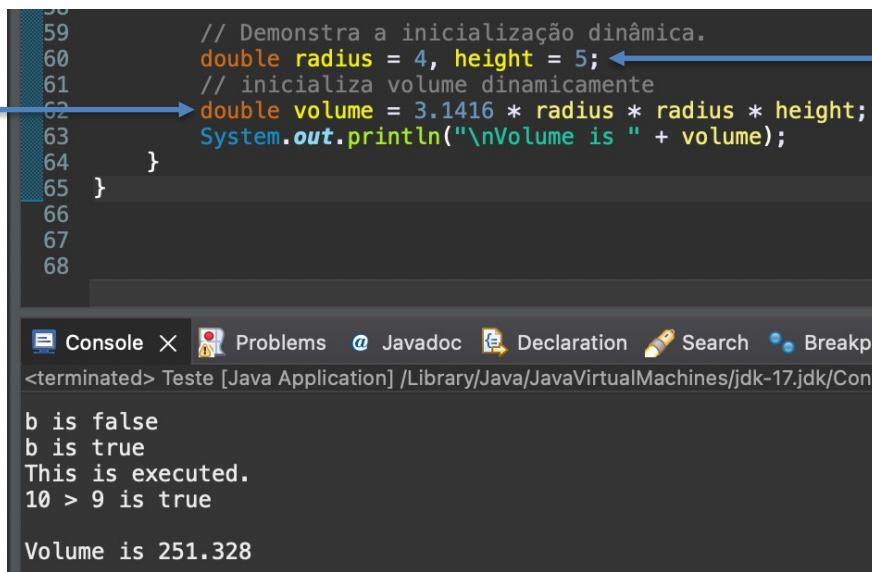
A tooltip window is open over the line `w` in the `System.out.println` statement, displaying the message: "The local variable w may not have been initialized". It also indicates "1 quick fix available" and provides a link to "Initialize variable".

The IDE interface includes tabs for Console, Problems, Javadoc, and Declarati... (partially visible). The console output shows:

```
b is true
This is executed.
10 > 9 is true
27
```

- **Inicialização de variável**

- **Inicialização dinâmica** ocorre no momento que a variável é declarada dentro do programa.
- Exemplo de um programa que calcula o volume de um cilindro dado o raio de sua base e sua altura:



```
59      // Demonstra a inicialização dinâmica.
60      double radius = 4, height = 5; ← Inicialização por constante
61      // inicializa volume dinamicamente
62      double volume = 3.1416 * radius * radius * height;
63      System.out.println("\nVolume is " + volume);
64  }
65
66
67
68
```

Inicialização dinâmica →

← Inicialização por constante

Console X Problems Javadoc Declaration Search Breakpo
<terminated> Teste [Java Application] /Library/Java/JavaVirtualMachines/jdk-17.jdk/Cont
b is false
b is true
This is executed.
10 > 9 is true

Volume is 251.328

- **Escopo e tempo de vida das variáveis**

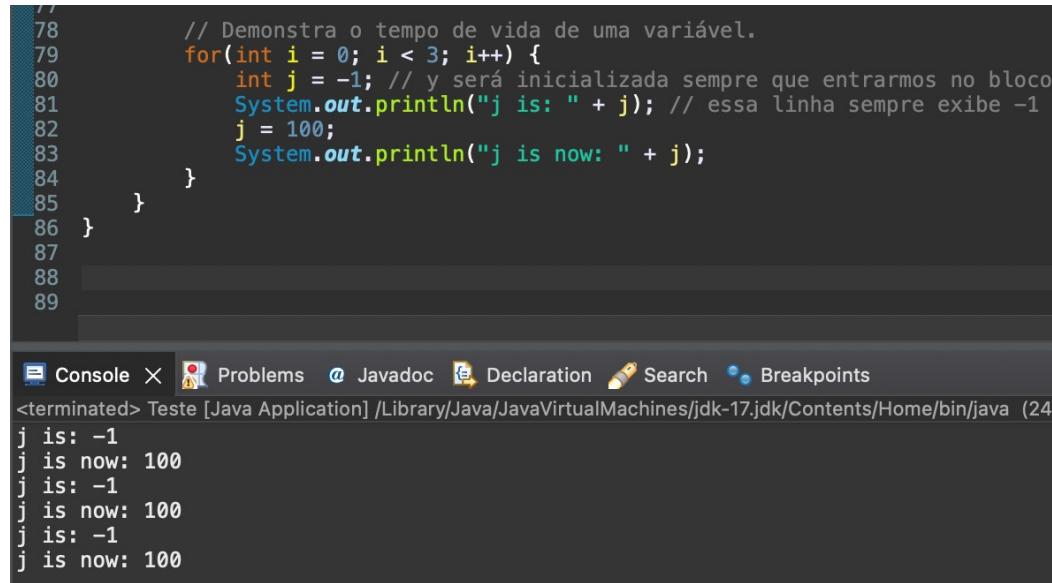
- Java permite que as variáveis sejam declaradas dentro de qualquer bloco.
- Um **escopo** determina que objetos estarão visíveis para outras partes de seu programa e o tempo de vida desses objetos.
- Por exemplo, sempre que você criar um bloco de código, estará criando um novo escopo **aninhado**.
 - Quando isso ocorre, o escopo externo engloba o escopo interno. Ou seja, os objetos declarados no escopo externo poderão ser vistos por um código que estiver dentro do escopo interno.
 - O inverso não é verdadeiro.

- Escopo e tempo de vida das variáveis
 - Exemplo escopo

```
// Demonstra o escopo de bloco.  
int m; // conhecida pelo código dentro de main()  
m = 10;  
if(m == 10) { // inicia novo escopo  
    int n = 20; // conhecida apenas nesse bloco  
    // tanto m quanto n são conhecidas aqui.  
    System.out.println("m and n: " + m + " " + n);  
    m = n * 2;  
}  
n = 100; // Erro! n não é conhecida aqui  
// m ainda é conhecida aqui.  
System.out.println("m is " + m);
```

- Escopo e tempo de vida das variáveis

- Exemplo tempo de vida



The screenshot shows a Java code editor and a terminal window. The code editor displays a snippet of Java code demonstrating variable scope and lifetime. The terminal window shows the execution of the code, printing the value of variable j at different points in the loop.

```
// Demonstrando o tempo de vida de uma variável.  
for(int i = 0; i < 3; i++) {  
    int j = -1; // j será inicializada sempre que entrarmos no bloco  
    System.out.println("j is: " + j); // essa linha sempre exibe -1  
    j = 100;  
    System.out.println("j is now: " + j);  
}  
}  
  
<terminated> Teste [Java Application] /Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java (24 c  
j is: -1  
j is now: 100  
j is: -1  
j is now: 100  
j is: -1  
j is now: 100
```

Sumário

- Java
 - Introdução
 - Vantagens e desvantagens
 - Tipos primitivos
 - **Operadores**
 - Estruturas condicionais
 - Estruturas de repetição

- **Operadores**

- Um operador é um símbolo que solicita ao compilador que execute uma operação matemática ou lógica específica.
- Java tem quatro classes gerais de operadores: **aritmético, bitwise, relacional e lógico**.

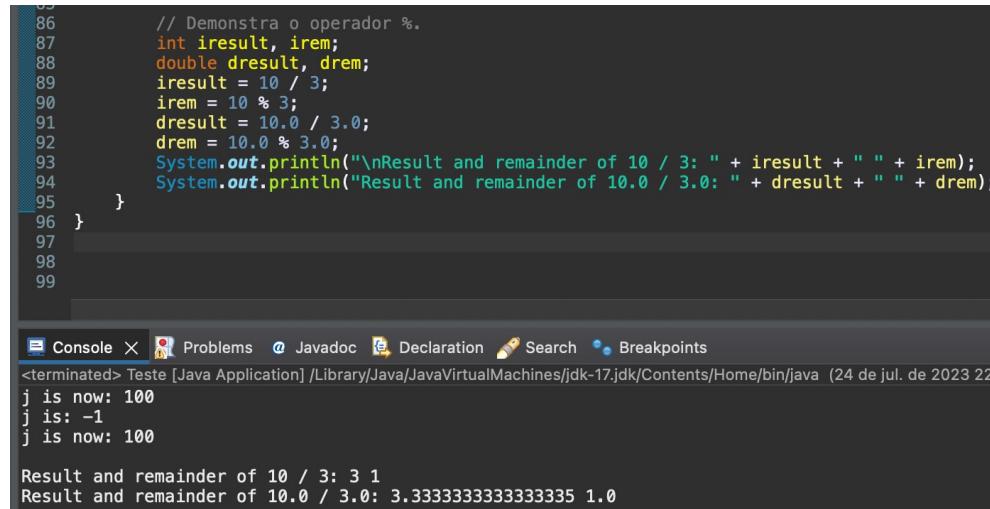
- **Operadores Aritméticos**

- Os operadores `+`, `-`, `*` e `/` podem ser aplicados a qualquer tipo de dado numérico interno. **Também podem ser usados em objetos de tipo char.**

Operador	Significado
<code>+</code>	Adição (também mais unário)
<code>-</code>	Subtração (também menos unário)
<code>*</code>	Multiplicação
<code>/</code>	Divisão
<code>%</code>	Módulo
<code>++</code>	Incremento
<code>--</code>	Decremento

• Operadores Aritméticos

- Em Java, o operador % é chamado de operador de resto ou operador de módulo.
- Ele é usado para calcular o resto da divisão de dois números inteiros.
- O operador % retorna o valor do resto após dividir o primeiro número pelo segundo.



The screenshot shows a Java code editor and a terminal window. The code in the editor demonstrates the modulus operator (%). It includes integer division (10 / 3), floating-point division (10.0 / 3.0), and printing the results and remainders. The terminal window shows the execution of the code and its output.

```
85      // Demonstra o operador %.
86
87     int irest, irem;
88     double drest, drem;
89
90     irest = 10 / 3;
91     irem = 10 % 3;
92     drest = 10.0 / 3.0;
93     drem = 10.0 % 3.0;
94
95     System.out.println("\nResult and remainder of 10 / 3: " + irest + " " + irem);
96     System.out.println("Result and remainder of 10.0 / 3.0: " + drest + " " + drem);
97
98
99 }
```

Console X Problems @ Javadoc Declaration Search Breakpoints

<terminated> Teste [Java Application] /Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java (24 de jul. de 2023 22:11)

```
j is now: 100
j is: -1
j is now: 100

Result and remainder of 10 / 3: 3 1
Result and remainder of 10.0 / 3.0: 3.3333333333333335 1.0
```

• Operadores Relacionais e Lógicos

- Relacional se refere aos relacionamentos que os valores podem ter uns com os outros, e lógico se refere às maneiras como os valores verdadeiro e falso podem estar conectados.

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

Operador	Significado
<code>&</code>	AND
<code> </code>	OR
<code>^</code>	XOR (exclusive OR)
<code> </code>	OR de curto-circuito
<code>&&</code>	AND de curto-circuito
<code>!</code>	NOT

- Podemos comparar termos para verificar se são iguais ou diferentes com o uso de `==` e `!=`.
- No entanto, os operadores de comparação `<`, `<=` ou `>=` só podem ser aplicados aos tipos que dão suporte a um relacionamento sequencial.

- **Operadores Relacionais e Lógicos**

- Importante: Cuidado ao tentar comparar objetos com ==.
- O operador == funciona bem em tipos primitivos, porém em objetos o operador == irá comparar endereços de memória ao invés de comparar os conteúdos dos objetos.
 - Em outras palavras, ele verifica se ambos os objetos apontam para a mesma localização de memória, ou seja, se eles são exatamente o mesmo objeto.
- Em objetos utilizamos o método **equals**.

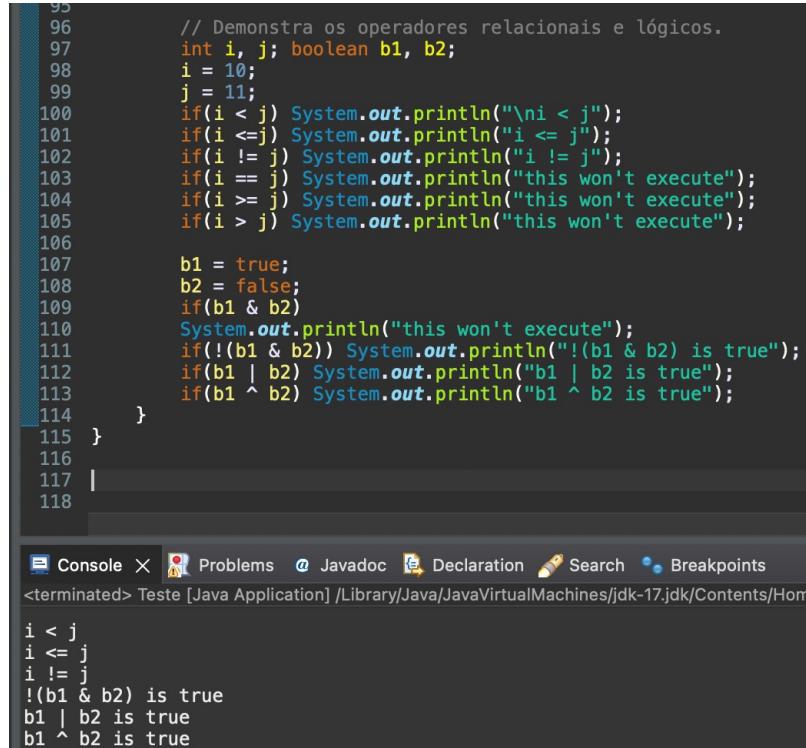
- Operadores Relacionais e Lógicos

- Exemplos de and (&), or (|), xor (^) e not (!):

		and	or	xor	not
p	q	p & q	p q	p ^ q	!p
Falso	Falso	Falso	Falso	Falso	Verdadeiro
Verdadeiro	Falso	Falso	Verdadeiro	Verdadeiro	Falso
Falso	Verdadeiro	Falso	Verdadeiro	Verdadeiro	Verdadeiro
Verdadeiro	Verdadeiro	Verdadeiro	Verdadeiro	Falso	Falso

• Operadores Relacionais e Lógicos

```
95      // Demonstra os operadores relacionais e lógicos.
96      int i, j; boolean b1, b2;
97      i = 10;
98      j = 11;
99
100     if(i < j) System.out.println("\ni < j");
101     if(i <= j) System.out.println("i <= j");
102     if(i != j) System.out.println("i != j");
103     if(i == j) System.out.println("this won't execute");
104     if(i >= j) System.out.println("this won't execute");
105     if(i > j) System.out.println("this won't execute");
106
107     b1 = true;
108     b2 = false;
109     if(b1 & b2)
110         System.out.println("this won't execute");
111     if(!(b1 & b2)) System.out.println("!(b1 & b2) is true");
112     if(b1 | b2) System.out.println("b1 | b2 is true");
113     if(b1 ^ b2) System.out.println("b1 ^ b2 is true");
114 }
115 }
116 |
117 |
118 
```



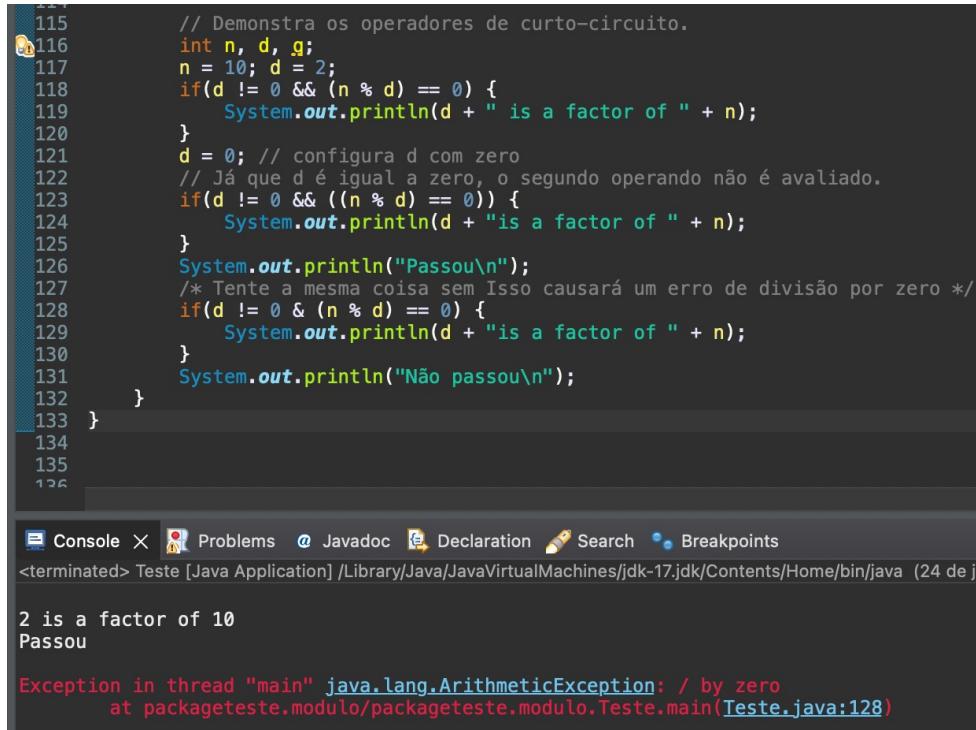
The screenshot shows an IDE interface with a code editor and a terminal window. The code editor contains the Java code above. The terminal window is titled 'Console' and shows the following output:

```
i < j
i <= j
i != j
!(b1 & b2) is true
b1 | b2 is true
b1 ^ b2 is true
```

- **Operadores Lógicos de Curto Circuito**

- Java fornece versões especiais de curto-circuito de seus operadores lógicos AND e OR que podem ser usadas para produzir código mais eficiente.
- O operador AND de curto-circuito é `&&` e o operador OR de curto-circuito é `||`.
 - Seus equivalentes comuns são `&` e `|`.
- A única diferença entre as versões comum e de curto-circuito é que a versão comum sempre avalia cada operando e **a versão de curto-circuito só avalia o segundo operando quando necessário.**

• Operadores Lógicos de Curto Circuito



The screenshot shows a Java code editor with the following code:

```
115     // Demonstra os operadores de curto-circuito.
116     int n, d, q;
117     n = 10; d = 2;
118     if(d != 0 && (n % d) == 0) {
119         System.out.println(d + " is a factor of " + n);
120     }
121     d = 0; // configura d com zero
122     // Já que d é igual a zero, o segundo operando não é avaliado.
123     if(d != 0 && ((n % d) == 0)) {
124         System.out.println(d + "is a factor of " + n);
125     }
126     System.out.println("Passou\n");
127     /* Tente a mesma coisa sem Isso causará um erro de divisão por zero */
128     if(d != 0 & (n % d) == 0) {
129         System.out.println(d + "is a factor of " + n);
130     }
131     System.out.println("Não passou\n");
132 }
133
134
135
136
```

The code demonstrates the short-circuit behavior of the logical AND (`&&`) and logical OR (`&`) operators. It prints "2 is a factor of 10" and "Passou" because the second operand of the second if-statement is never evaluated due to the short-circuit nature of the operator. However, it throws an `ArithmaticException` at line 128 because division by zero is attempted.

Console X Problems Javadoc Declaration Search Breakpoints

<terminated> Teste [Java Application] /Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java (24 de ju)

```
2 is a factor of 10
Passou

Exception in thread "main" java.lang.ArithmaticException: / by zero
at packageteste.modulo/packageteste.modulo.Teste.main(Teste.java:128)
```

- **Operadores Bitwise**

- Os operadores de bitwise, também conhecidos como operadores **bit a bit**, são operadores especiais em linguagens de programação que realizam operações em nível de bit em números inteiros.
- Esses operadores manipulam os bits individuais dos números, permitindo que você realize operações lógicas e aritméticas em cada bit separadamente.
- Os operadores de bitwise estão disponíveis em muitas linguagens de programação, incluindo Java, C, C++, Python, entre outras.
- Esses operadores podem ser usados em valores de tipo **long, int, short, char ou byte**. As operações bitwise não podem ser usadas com tipos **boolean, float, double ou tipos de classe**. Eles são chamados de bitwise por serem usados para testar, configurar ou deslocar os bits individuais que compõem um valor.

- Operadores Bitwise

p	q	p & q	p q	p ^ q	~p
0	0	0	0	0	1
1	0	0	1	1	0
0	1	0	1	1	1
1	1	1	1	0	0

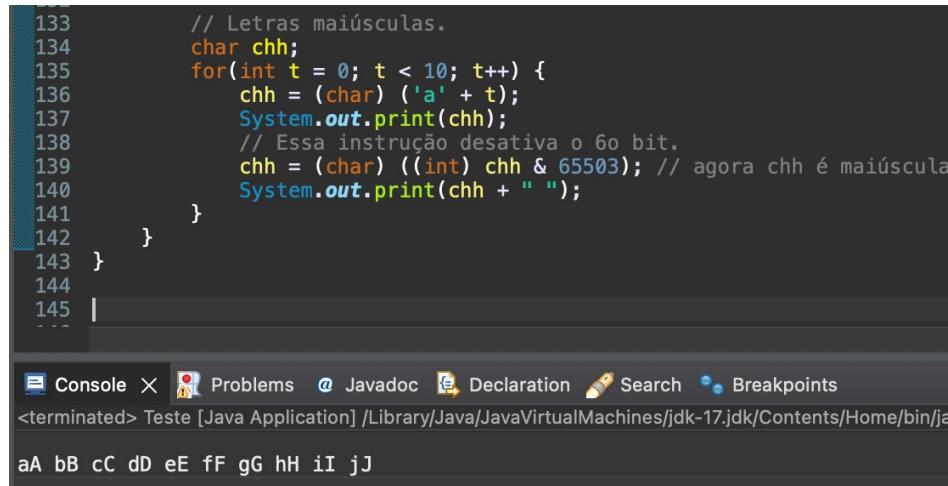
No que diz respeito ao seu uso mais comum, você pode considerar o AND bitwise como uma maneira de desativar bits. Isto é, qualquer bit que for 0 em um dos operandos fará o bit correspondente do resultado ser configurado com 0. Por exemplo:

$$\begin{array}{r} 1101\ 0011 \\ \&\ 1010\ 1010 \\ \hline 1000\ 0010 \end{array}$$

Operador	Resultado
&	AND bitwise
	OR bitwise
^	Exclusive OR bitwise
>>	Deslocamento para a direita
>>>	Deslocamento para a direita sem sinal
<<	Deslocamento para a esquerda
~	Complemento de um (NOT unário)

• Operadores Bitwise

- O programa a seguir demonstra o operador **&** ao transformar letras minúsculas em maiúsculas pela redefinição do 6º bit com 0. Como definido no conjunto de caracteres Unicode/ASCII, as letras minúsculas são iguais às maiúsculas exceto pelo fato de minúsculas terem o valor maior em exatamente 32 unidades. Logo, para transformar uma letra minúscula em maiúscula, apenas desative o 6º bit:



The screenshot shows a Java code editor with the following code:

```
133     // Letras maiúsculas.
134     char chh;
135     for(int t = 0; t < 10; t++) {
136         chh = (char) ('a' + t);
137         System.out.print(chh);
138         // Essa instrução desativa o 6º bit.
139         chh = (char) ((int) chh & 65503); // agora chh é maiúscula
140         System.out.print(chh + " ");
141     }
142 }
143 }
144
145 |
```

The code prints the first ten lowercase letters ('a' through 'j') followed by their uppercase counterparts ('A' through 'J'). The line `chh = (char) ((int) chh & 65503);` performs a bitwise AND operation with the value `65503`, which has a binary representation of `1000000000000001`. This operation effectively clears the 6th bit of each character's internal representation, thus converting it to its uppercase equivalent.

Below the code editor is a terminal window showing the output:

```
Console X Problems @ Javadoc Declaration Search Breakpoints
<terminated> Teste [Java Application] /Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java
aA bB cC dD eE fF gG hH iI jJ
```

- **Operadores de Incremento e Decremento**

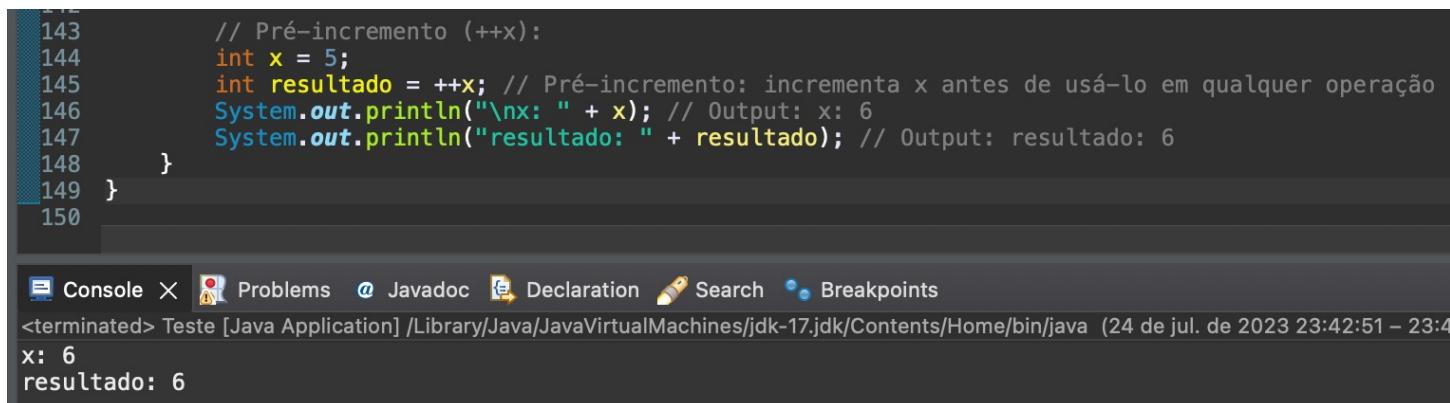
- `++` e `--` são os operadores Java de incremento e decremento.
 - `x = x + 1;`
 - é o mesmo que `x++;`
 - `x = x - 1;`
 - é o mesmo que `x--;`

- **Operadores de Incremento e Decremento**

- `++` e `--` são os operadores Java de incremento e decremento.
 - Tanto o operador de incremento quanto o de decremento podem preceder (prefixar) ou vir após (posfixar) o operando.
 - Por exemplo:
 - `x = x + 1;` pode ser escrito como
 - `++x; // forma prefixada` ou como
 - `x++; // forma postfixada`

- **Operadores de Incremento e Decremento**

- **++ e --** são os operadores Java de incremento e decremento.
 - **Pré-incremento (++x):** No pré-incremento, o valor da variável é incrementado antes que qualquer outra operação seja realizada usando o valor atual da variável. Em outras palavras, a variável é incrementada antes de ser usada em qualquer expressão ou atribuição.



The screenshot shows a Java code editor and a terminal window. The code editor displays the following Java code:

```
143     // Pré-incremento (++x):
144     int x = 5;
145     int resultado = ++x; // Pré-incremento: incrementa x antes de usá-lo em qualquer operação
146     System.out.println("nx: " + x); // Output: x: 6
147     System.out.println("resultado: " + resultado); // Output: resultado: 6
148 }
149 }
150 }
```

The terminal window below shows the output of the code execution:

- Console X Problems Javadoc Declaration Search Breakpoints
- <terminated> Teste [Java Application] /Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java (24 de jul. de 2023 23:42:51 – 23:44)
- x: 6
- resultado: 6

• Operadores de Incremento e Decremento

- `++` e `--` são os operadores Java de incremento e decremento.
 - **Pós-incremento (`x++`):** No pós-incremento, o valor atual da variável é usado primeiro em qualquer operação ou atribuição e, em seguida, a variável é incrementada após a operação ser concluída.



The screenshot shows a Java code editor with the following code:

```
148      // Pós-incremento (x++)
149      x = 5;
150      resultado = x++; // Pós-incremento: usa o valor atual de x e depois o incrementa
151      System.out.println("\nx: " + x); // Output: x: 6 (incrementado após a atribuição)
152      System.out.println("resultado: " + resultado); // Output: resultado: 5 (valor antes do incremento)
153
154  }
155 }
```

Below the code, the IDE interface includes tabs for Console, Problems, Javadoc, Declaration, Search, and Breakpoints. The Console tab shows the output:

```
<terminated> Teste [Java Application] /Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java (24 de jul. de 2023 23:45:58 – 23:45:58) [pid: 6]
x: 6
resultado: 5
```

- **Atribuição abreviada**

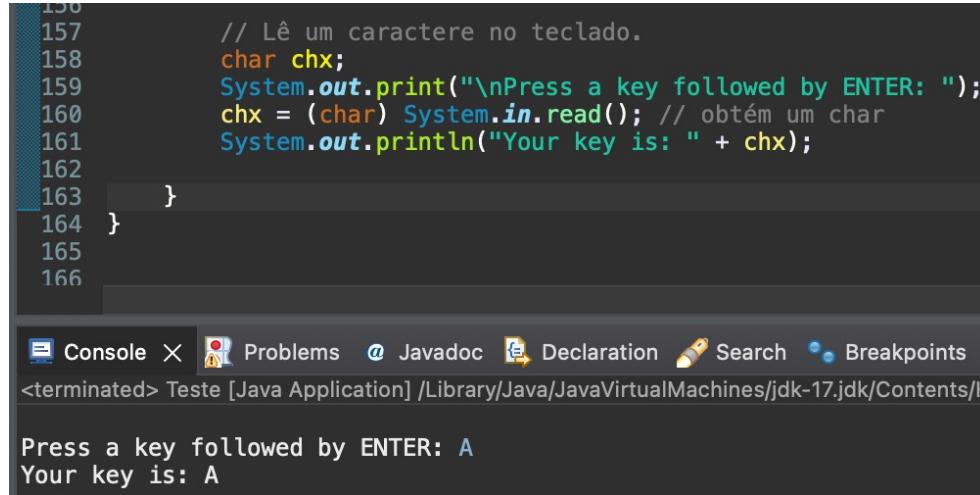
- O par de operadores `+=` solicita ao compilador que atribua a `x` o valor de `x` mais 10, por exemplo.
- Outro exemplo: A instrução `x = x - 100;` é igual a `x -= 100;`

<code>+=</code>	<code>-=</code>	<code>*=</code>	<code>/=</code>
<code>%=</code>	<code>&=</code>	<code> =</code>	<code>^=</code>

Já que esses operadores combinam uma operação com uma atribuição, são formalmente chamados de **operadores de atribuição compostos**.

- **Leitura pelo teclado**

- Iremos precisar daqui em diante:



The screenshot shows a Java code editor with the following code:

```
156
157     // Lê um caractere no teclado.
158     char chx;
159     System.out.print("\nPress a key followed by ENTER: ");
160     chx = (char) System.in.read(); // obtém um char
161     System.out.println("Your key is: " + chx);
162
163 }
164
165
166
```

Below the code editor is a terminal window showing the output of the program. The terminal interface includes tabs for Console, Problems, Javadoc, Declaration, Search, and Breakpoints. The message in the terminal is:

```
Console × Problems @ Javadoc Declaration Search Breakpoints
<terminated> Teste [Java Application] /Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/H

Press a key followed by ENTER: A
Your key is: A
```

Já que `System.in.read()` está sendo usado, o programa deve especificar a cláusula `throws java.io.IOException`.
Essa linha é necessária para tratar erros de entrada.

`public static void main(String args[]) throws java.io.IOException`

Sumário

- Java
 - Introdução
 - Vantagens e desvantagens
 - Tipos primitivos
 - Operadores
 - **Estruturas condicionais**
 - Estruturas de repetição

- **Estruturas condicionais**

- Instrução If

```
if(condição) instrução;
else instrução;
```

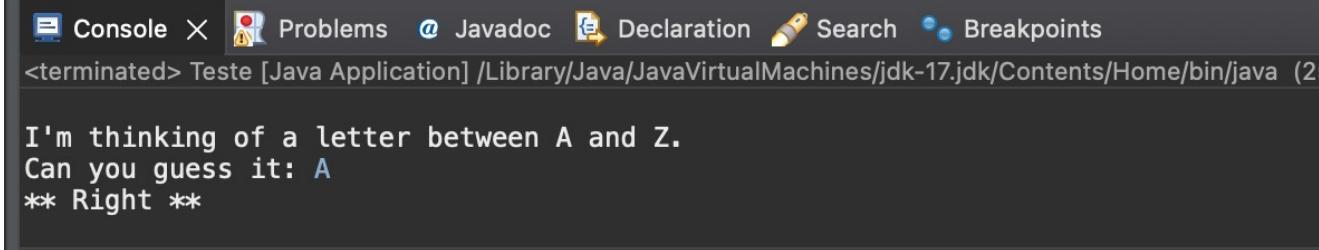
- Onde os alvos de **if** e **else** são instruções individuais. A cláusula **else** é opcional. Os alvos tanto de **if** quanto de **else** podem ser **blocos de instruções, como a seguir:**

```
if(condição) {
    sequência de instruções
}
else {
    sequência de instruções
}
```

```
if(condição)
{
    sequência de instruções
}
else
{
    sequência de instruções
}
```

- Estruturas condicionais

```
162
163     // Adivinhe a letra do jogo.
164     char chxy, answer = 'A';
165     System.out.println("\nI'm thinking of a letter between A and Z.");
166     System.out.print("Can you guess it: ");
167     chxy = (char) System.in.read(); // lê um char no teclado
168     if(chxy == answer) System.out.println("** Right **");
169     else System.out.println("...Sorry, you're wrong.");
170
171 }
172 }
173 |
174
175
```



The screenshot shows a Java application running in an IDE. The code is a simple program that asks the user to guess a letter between A and Z. The output in the console window shows the program's message, the user's input, and the program's response.

```
Console X Problems @ Javadoc Declaration Search Breakpoints
<terminated> Teste [Java Application] /Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java (25)

I'm thinking of a letter between A and Z.
Can you guess it: A
** Right **
```

- **Estruturas condicionais**

- A escada ou cadeia if-else-if
 - As expressões condicionais são avaliadas de cima para baixo.
 - Assim que uma condição verdadeira é encontrada, a instrução associada a ela é executada e o resto da escada é ignorado.
 - Se nenhuma das condições for verdadeira, a instrução else final será executada.

```
if(condição)
    instrução;
else if(condição)
    instrução;
else if(condição)
    instrução;
.
.
.
else
    statement;
```

- **Operador Ternário**

- O operador ternário é uma expressão condicional compacta que permite fazer uma escolha entre dois valores com base em uma condição booleana.
- Ele é chamado de "ternário" porque é composto por três partes: a condição, o valor a ser retornado se a condição for verdadeira e o valor a ser retornado se a condição for falsa.

condição ? valor_se_verdadeiro : valor_se_falso

```
// Operador Ternário
int numero = 10;
// Usando o operador ternário para verificar se o número é par ou ímpar
String result = (numero % 2 == 0) ? "par" : "ímpar";
System.out.println("O número " + numero + " é " + result + ".");
```

- **Switch**

- A instrução **switch** fornece uma ramificação com vários caminhos. Logo, ela permite que o programa faça uma seleção entre várias alternativas.
- Embora uma série de instruções **if** aninhadas possam executar testes com vários caminhos, em muitas situações, **switch** é uma abordagem mais eficiente e limpa.
- A expressão que controla o **switch** deve ser do tipo **byte**, **short**, **int**, **char**, **String** ou uma **enumeração**.

```
switch(expressão) {  
    case constante1:  
        sequência de instruções  
        break;  
    case constante2:  
        sequência de instruções  
        break;  
    case constante3:  
        sequência de instruções  
        break;  
    .  
    .  
    .  
    default:  
        sequência de instruções  
}
```

- **Enum**

- Em Java, um enum é um tipo de dado especial que representa um conjunto fixo de **constantes nomeadas**.
- Ele é uma abreviação para "enumeration" (enumeração), e foi introduzido na linguagem a partir do Java 5 para fornecer uma maneira mais elegante e segura de definir um conjunto de valores constantes relacionados.
- Ao definir um enum, você está criando um novo tipo de dado que consiste em um conjunto limitado de valores pré-definidos, conhecidos como "enum constants".
- Cada um desses valores é tratado como uma instância da própria classe de enumeração e possui um nome significativo associado.

- **Enum**

```
// Declaração do enum
enum DiaDaSemana {
    SEGUNDA, TERCA, QUARTA, QUINTA, SEXTA, SABADO, DOMINGO
}

// Uso do enum
DiaDaSemana dia = DiaDaSemana.QUARTA;

// Exemplo de uso do switch-case com enum
switch (dia) {
    case SEGUNDA:
    case TERCA:
    case QUARTA:
    case QUINTA:
    case SEXTA:
        System.out.println("Dia de trabalho.");
        break;
    case SABADO:
    case DOMINGO:
        System.out.println("Final de semana.");
        break;
}
```

Também é possível um switch fazer parte da sequência de instruções de um switch externo.

Isso é chamado de switch aninhado.

Sumário

- Java
 - Introdução
 - Vantagens e desvantagens
 - Tipos primitivos
 - Operadores
 - Estruturas condicionais
 - **Estruturas de repetição**

- **Estruturas de repetição**

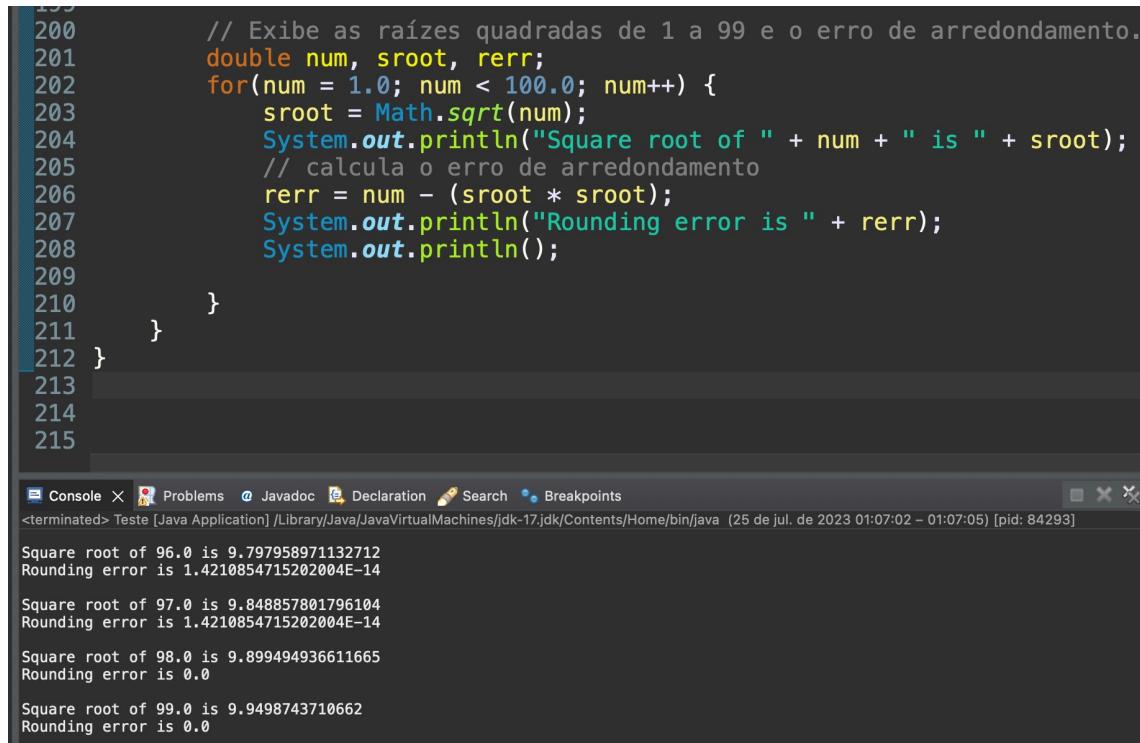
- **For**
- A forma geral do laço **for** para a repetição de uma única instrução é:

for(inicialização; condição; iteração) instrução;

- Para a repetição de um bloco, a forma geral é:

for(inicialização; condição; iteração) { sequência de instruções }

- Estruturas de repetição



The image shows a Java application running in an IDE. The code is a simple loop that calculates the square root of numbers from 1.0 to 99.0 and prints both the result and the rounding error. The output in the console shows the results for each iteration.

```
199
200     // Exibe as raízes quadradas de 1 a 99 e o erro de arredondamento.
201     double num, sroot, rerr;
202     for(num = 1.0; num < 100.0; num++) {
203         sroot = Math.sqrt(num);
204         System.out.println("Square root of " + num + " is " + sroot);
205         // calcula o erro de arredondamento
206         rerr = num - (sroot * sroot);
207         System.out.println("Rounding error is " + rerr);
208         System.out.println();
209
210     }
211 }
212 }
213
214
215
```

Console X Problems Javadoc Declaration Search Breakpoints
<terminated> Teste [Java Application] /Library/Java/JavaVirtualMachines/jdk-17.jdk/Contents/Home/bin/java (25 de jul. de 2023 01:07:02 – 01:07:05) [pid: 84293]

```
Square root of 96.0 is 9.797958971132712
Rounding error is 1.4210854715202004E-14

Square root of 97.0 is 9.848857801796104
Rounding error is 1.4210854715202004E-14

Square root of 98.0 is 9.899494936611665
Rounding error is 0.0

Square root of 99.0 is 9.9498743710662
Rounding error is 0.0
```

- Estruturas de repetição

- Variações do laço For

- Nesse caso, vírgulas separam as duas instruções de inicialização e as duas expressões de iteração. Quando o laço começa, tanto i quanto j são inicializadas.
- Sempre que o laço se repete, i é incrementada e j é decrementada. O uso de múltiplas variáveis de controle de laço com frequência é conveniente e pode simplificar certos algoritmos.

```
for(i=0, j=10; i < j; i++, j--)  
    System.out.println("i and j: " + i + " " + j);
```

- Estruturas de repetição

- Variações do laço For

- A condição que controla o laço pode ser qualquer expressão booleana válida. Ela não precisa envolver a variável de controle de laço.

```
// Executa o laço até um S ser digitado.  
System.out.println("Press S to stop.");  
for(i = 0; (char) System.in.read() != 'S'; i++)  
    System.out.println("Pass #" + i);
```

- Estruturas de repetição

- Variações do laço For

- Algumas variações interessantes do laço for são criadas quando deixamos vazias partes da definição do laço.

```
// Partes de for podem estar vazias.  
for(i = 0; i < 10; ) {  
    System.out.println("Pass #" + i);  
    i++; // incrementa a variável de controle de laço  
}
```

- **Estruturas de repetição**

- **Laço infinito**

- Você pode criar um laço infinito (um laço que nunca termina) usando **for** se deixar a expressão condicional vazia.
- É possível interromper um laço desse tipo com o uso da instrução **break**.

```
for(;;) // laço intencionalmente infinito
{
//...
}
```

- **Estruturas de repetição**

- **Laço sem corpo**

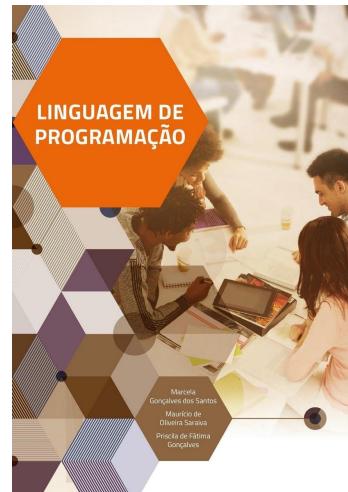
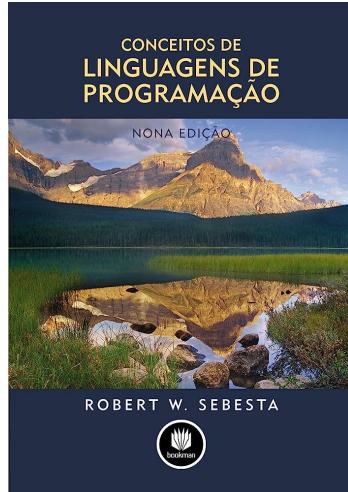
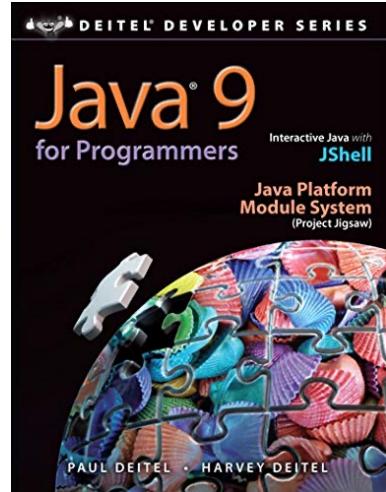
- Em Java, o corpo associado a um laço **for** (ou qualquer outro laço) **pode estar vazio**.
- Isso ocorre porque uma instrução nula é sintaticamente válida.
- Laços sem corpo costumam ser úteis.

```
// O corpo de um laço pode estar vazio.

int sum = 0;
// soma os números até 5
for(i = 1; i <= 5; sum += i++);

System.out.println("Sum is " + sum);
```

Referências





Obrigado!

joaopauloaramuni@gmail.com