

The background features several thin, light blue lines that create a sense of depth and structure. On the left side, there are vertical and horizontal lines that form a corner-like structure. On the right side, there are diagonal lines that extend from the bottom towards the top right. These lines are layered, giving the impression of a 3D architectural or technical drawing.

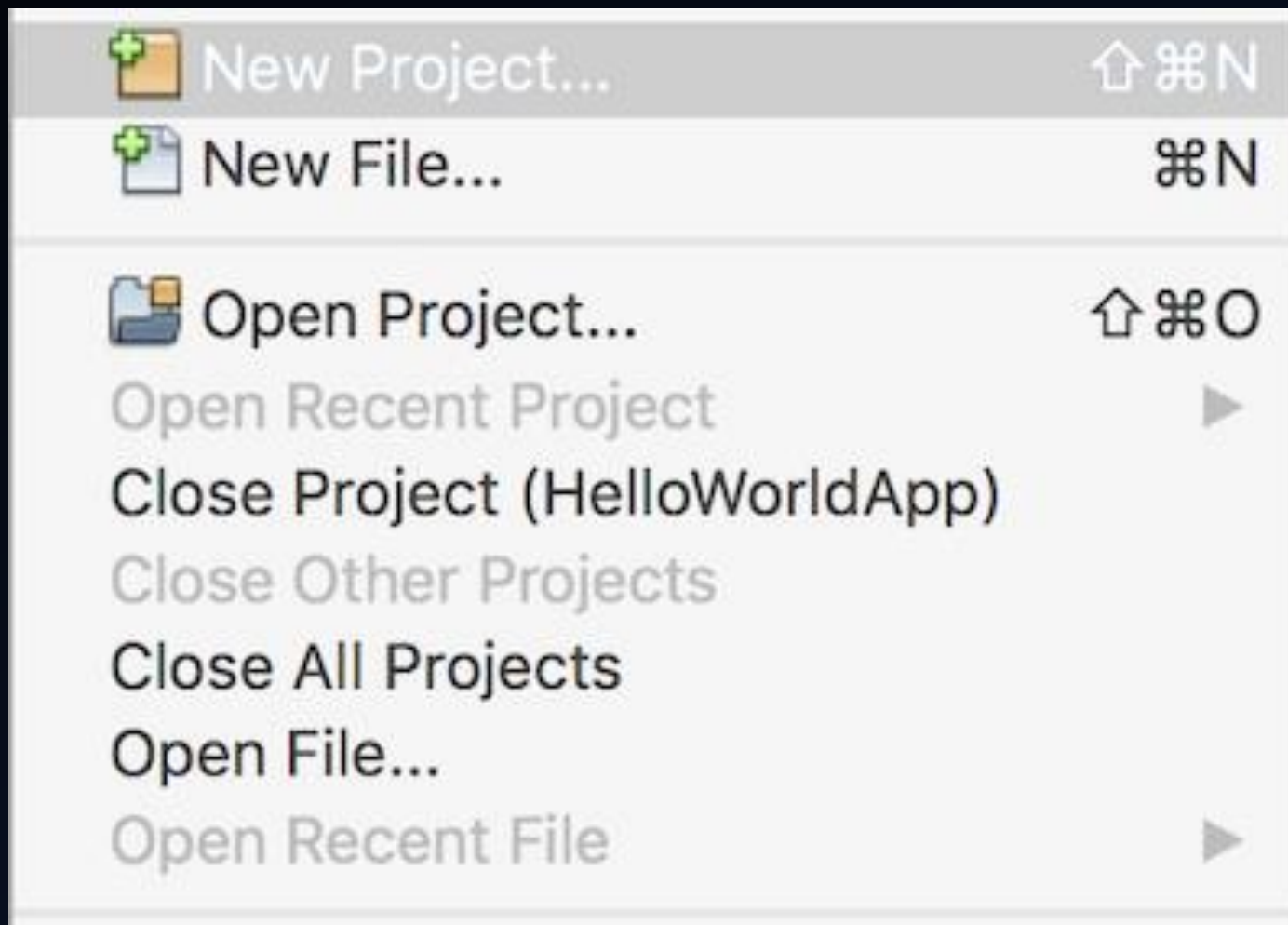
JAVA

INTRODUÇÃO A
LINGUAGEM DE PROGRAMAÇÃO
JAVA

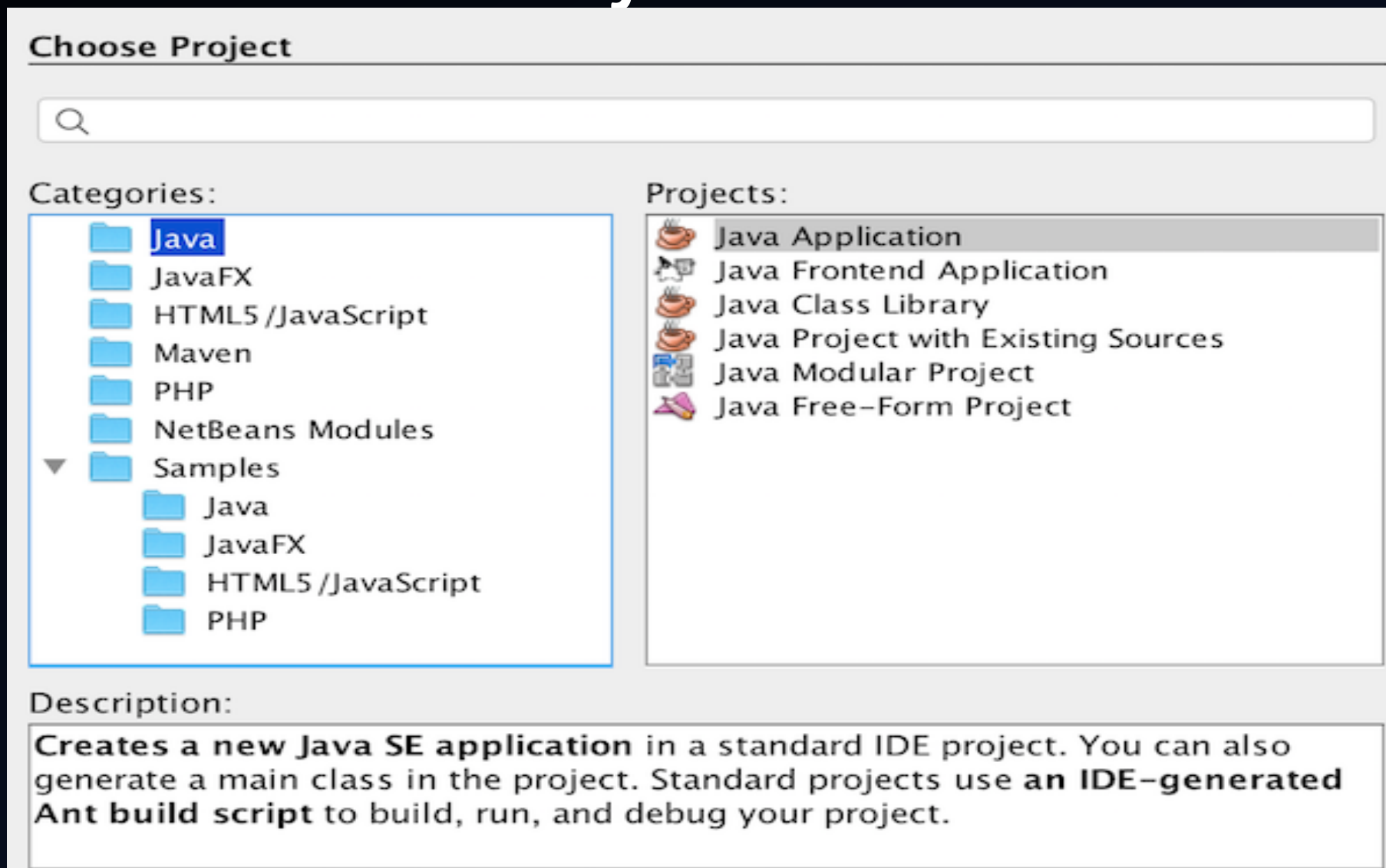
Configurações Básicas

LINGUAGEM DE PROGRAMAÇÃO
JAVA

Criando um Projeto JAVA



Criando um Projeto JAVA



Criando um Projeto JAVA

Name and Location

Project Name:

HelloWorldApp

Project Location:

/Users/geertjanwielenga/NetBeansProjects

Browse...

Project Folder:

/Users/geertjanwielenga/NetBeansProjects/Hell

☐ Use Dedicated Folder for Storing Libraries

Libraries Folder:

Browse...

Different users and projects can share the same compilation libraries (see Help for details).

☒ Create Main Class

helloworldapp.HelloWorldApp

HelloWorldApp - Apache NetBeans IDE 10.0

284,9/432,0MB

Q Search (⌘+I)

Projects Files Services

▼ HelloWorldApp

- Source Packages
 - helloworldapp
 - HelloWorldApp.java
- Libraries

HelloWorldApp.java - Navigator

Members

<empty>

▼ HelloWorldApp

- main(String[] args)

Start Page HelloWorldApp.java

Source History

```
1  /*
2  * To change this license header, choose License Headers in Project Properties.
3  * To change this template file, choose Tools | Templates
4  * and open the template in the editor.
5  */
6  package helloworldapp;
7
8  /**
9   *
10  * @author geertjanwielenga
11  */
12  public class HelloWorldApp {
13
14      /**
15       * @param args the command line arguments
16       */
17      public static void main(String[] args) {
18          // TODO code application logic here
19      }
20
21  }
22
```

1:1 INS



ENTENDENDO O CÓDIGO JAVA...

HelloWorld.java

```
1 public class HelloWorld {  
2     public static void main(String... args) {  
3         System.out.println("Hello, world!");  
4     }  
5 }
```

O nome do arquivo: "HelloWorld.java":

Java possui uma característica bastante única quanto ao nome dos arquivos: obrigatoriamente o nome do arquivo deve ser o mesmo que o nome da principal classe implementada.

HelloWorld.java

```
1 public class HelloWorld {  
2     public static void main(String... args) {  
3         System.out.println("Hello, world!");  
4     }  
5 }
```

Uma classe é um elemento do código Java que utilizamos para representar objetos do mundo real. Dentro dela é comum declararmos atributos e métodos, que representam, respectivamente, as características e comportamentos desse objeto.

HelloWorld.java

```
1 public class HelloWorld {  
2     public static void main(String... args) {  
3         System.out.println("Hello, world!");  
4     }  
5 }
```

Um método em Java é equivalente a uma função, sub-rotina ou procedimento em outras linguagens de programação.

HelloWorld.java

```
1 public class HelloWorld {  
2     public static void main(String... args) {  
3         System.out.println("Hello, world!");  
4     }  
5 }
```

Parâmetros são valores que passamos para um método, estes valores são recebidos pelas variáveis criadas dentro do parênteses do método.

HelloWorld.java

```
1 public class HelloWorld {  
2     public static void main(String... args) {  
3         System.out.println("Hello, world!");  
4     }  
5 }
```

Perceba que os comandos executados são apenas os que estão entre chaves ("{" e "}"). Isso se dá porque as chaves delimitam o início e fim de um escopo, entenda como "o início e fim do código".

HelloWorld.java

```
1 public class HelloWorld {  
2     public static void main(String... args) {  
3         System.out.println("Hello, world!");  
4     }  
5 }
```

O objeto **System.out** é a saída padrão, que permite exibir as *Strings* no console (terminal) de comando quando o aplicativo de Java é executado. Dentro desse objeto existem métodos para gerar saídas de Strings, entre elas são: **println()**, **print()** e o **printf()**.

Principais diferenças dos métodos:

System.out.println - Insere uma nova linha, deixando o marcador posicionado na linha abaixo.

System.out.print - Mantém o cursor na mesma linha. Geralmente são utilizadas sequências de escape para pular uma linha.

System.out.printf - Especifica o formato da entrada do tipo de valor, que deve ser o mesmo tipo de dados apontado na instrução. Se possuir alguma dúvida verifique a tabela acima dos tipos de dados que podem ser usados.

Importações (Declarações *import*)

Um dos pontos fortes do Java é seu rico conjunto de classes predefinidas que você pode reutilizar em vez de “reinventar a roda”.

Essas classes são agrupadas em pacotes;

Chamados de grupos de classes, coletivamente, são chamadas de biblioteca de classes Java, ou Java Application Programming Interface (API)

Importações (Declarações *import*)

Todas as declarações *import* devem aparecer antes da primeira declaração da classe no arquivo.

Inserir uma declaração *import* dentro ou depois de uma declaração de classe é um erro de sintaxe.

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



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

```
- /**  
 *  
 * @author  
 */
```

```
public class HelloWorld {
```

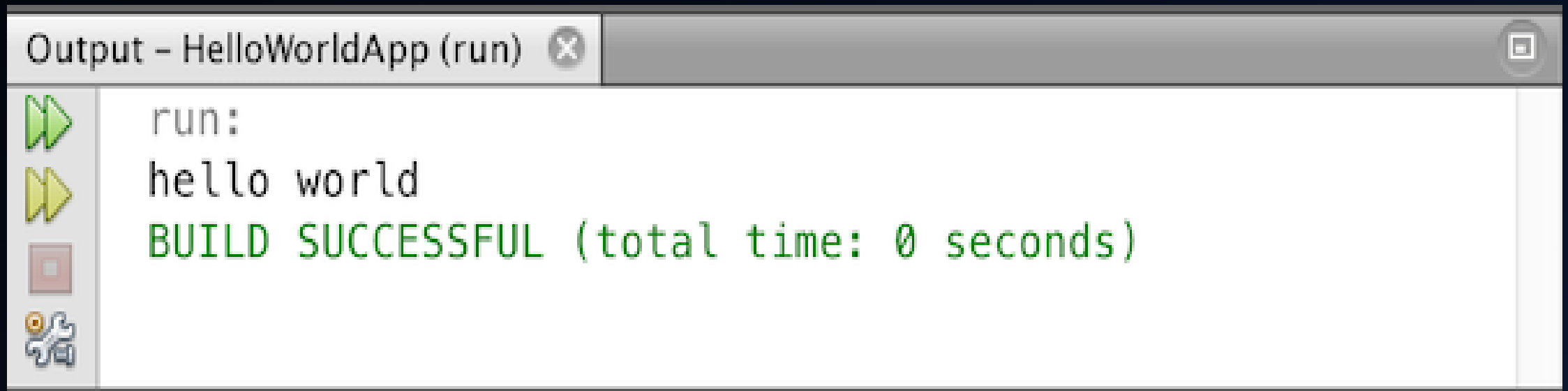
```
- /**  
 * @param args the command line arguments  
 */
```

```
+ public static void main(String[] args) { ...18 lines }
```

```
}
```

Para executar o programa:

Escolha Executar > Executar Projeto





**PRÁTICANDO
COM JAVA...**

Variáveis

LINGUAGEM DE PROGRAMAÇÃO
JAVA

Variáveis

A linguagem de programação Java define os seguintes tipos de variáveis:

- Variáveis de instância (campos não estáticos)
- Variáveis de classe (campos estáticos)
- Variáveis locais
- Parâmetros

Variáveis

Variáveis de instância (campos não estáticos)

Uma variável de instância é uma variável cujo valor é **específico ao objeto** e não à classe. Uma variável de instância em geral possui um **valor diferente** em cada objeto membro da classe.

```
NomeDaClasse nomeDoObjeto;  
nomeDoObjeto = new NomeDaClasse();
```

Variáveis

Variáveis de classe (campos estáticos)

Uma variável de classe é uma variável cujo valor é **comum a todos os objetos membros da classe**. Mudar o valor de uma variável de classe em um objeto membro automaticamente muda o valor para todos os objetos membros. Uma variável é considerada como de instância por "default". Para declarar uma variável de classe, acrescenta-se a palavra-chave **static**.

static int numeroDeInstanciasDestaClasse;

static int LEFT = 1;

Variáveis

Variáveis Globais e Locais

As variáveis que são declaradas fora de qualquer **método** (usualmente no cabeçalho da classe) e são acessíveis por qualquer método da classe. **São variáveis globais.**

Uma variável local é uma variável definida dentro de **um método**. Variáveis locais devem ser inicializadas antes de usar. Eles não têm um valor padrão e contêm dados de lixo até serem inicializados.

Variáveis

```
class NomeDaClasse
{
    TipoDaVariavel variavel1; // variável global

    TipoDeRetorno nomeDoMetodo()
    {
        TipoDaVariavel variavel2; // variável local, definida neste método

        for( int i = 0; i < 10; i++ )
        {
            ... // a variável i é local, definida só dentro deste bloco
        }
    }
}
```

Variáveis

Parâmetros

Parâmetros são sempre classificados como "variáveis" e não "campos". São variáveis que fornecem informações extras para um método. Isso também se aplica a outras construções que aceitam parâmetros (como construtores e manipuladores de exceção).

Nomenclatura

Java possui regras sobre nomes de identificadores, sobre variáveis, métodos, classes e campos. Existem apenas três regras para lembrar para os identificadores legais:

- O nome deve começar com uma letra ou o símbolo \$ ou _.
- devem ser compostos de uma única palavra (sem espaços, vírgulas, etc.) e podem ser compostos de letras e números
- Você não pode usar o mesmo nome que uma palavra reservada do Java. Como você pode imaginar, um palavra reservada é uma palavra-chave que o Java reservou para que você não tenha permissão para usá-la.



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

Nomenclatura

A maioria dos desenvolvedores de Java segue essas convenções para nomes de identificadores:

- Os nomes dos métodos e variáveis começam com uma letra minúscula seguida por *CamelCase*.
- Os nomes das classes começam com uma letra maiúscula seguida por *CamelCase*.
- Java faz distinção entre maiúsculas e minúsculas, portanto, você pode usar versões das palavras-chave que só diferem na primeira letra.

Exemplos de variáveis:

dia,
nomeDoAluno,
rotaçõesPorMinuto,
estadoDaLâmpada,
nomeCompleto,
disciplinaDeJava,
enderecoRuaBairroCidade
nomeDaInstancia

NomeDaClasse (nome da classe começa com letra maiúscula)

Comentários

Um comentário curto relativo a uma linha de código pode ser incluído no fim da linha, precedido de `//` :

//Comentário sobre o comando ...

Um comentário precedido de `/*` e seguido de `*/` pode ocupar várias linhas:

/ Comentário maior,
que pode ocupar várias linhas. */*

Obviamente, isto serve também para forçar o compilador a ignorar temporariamente um trecho de código.

Comentários

O JDK oferece um recurso para gerar automaticamente um arquivo HTML documentando uma classe.

Comentários que forem precedidos de `/**` e seguidos de `*/` serão incluídos neste arquivo:

```
/** Este comentário será incluído no arquivo HTML  
documentando a classe. */
```

Para gerar a documentação relativa à(s) classe(s) cuja(s) fonte(s) estão no arquivo *MeuPrograma.java*, digita-se na linha de comando: `javadoc MeuPrograma.java`

Tipos de dados

JAVA

Tipos de dados primitivos

Java tem oito tipos de dados primitivos. Estes oito tipos de dados representam os blocos de construção para objetos Java, porque todos os objetos Java são apenas coleção complexa desses tipos de dados primitivos.

byte

long

boolean

short

float

char

int

double

Tipos de dados primitivos

byte

O byte é um inteiro de complemento de dois com sinal de 8 bits.

Tem um valor mínimo de -128 e um valor máximo de 127 (inclusive).

Um byte são 8 bits.

Um bit tem dois valores possíveis. (Estas são definições básicas de ciência da computação que você deve memorizar.)

```
byte numero = 110;
```

Tipos de dados primitivos

short

short: possui 2 bytes de informação ou 16 bits;

Tem um valor mínimo de -32.768 e um valor máximo de 32.767 (inclusive).

```
short numero = 16096;
```

Tipos de dados primitivos

int

Por padrão, o int é um tipo de dados inteiro de complemento de dois com sinal de 32 bits, que tem um valor mínimo de -2^{31} e um valor máximo de $2^{31} - 1$.

No Java SE 8 e posterior, você pode usar o int tipo de dados para representar um inteiro não assinado de 32 bits, que tem um valor mínimo de 0 e um valor máximo de $2^{32} - 1$.

Mais de 2 milhões de números inteiros.

```
int numero = 9109545;
```

Tipos de dados primitivos

long

O long é um inteiro de complemento de dois de 64 bits.

O long tem um valor mínimo de -2^{63} e um valor máximo de $2^{63} - 1$. No Java SE 8 e posterior, você pode usar o long tipo de dados para representar um comprimento de 64 bits sem sinal, que tem um valor mínimo de 0 e um valor máximo de 2^{64} .

Mais de 9 quintilhões de números.

```
long numero = 10798564659564;
```

byte

-128 a 127

short

-32.768 a 32.767

int

-2.147.483.648 a 2.147.483.647

long

-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807

Tipos de dados primitivos

float

O float é um ponto flutuante de 32 bits de precisão simples.

Este tipo de dados nunca deve ser usado para valores precisos, como moeda.

O float podem representar valores numéricos com ponto flutuante dentro da faixa $1.40129846432481707e-45$ a $3.40282346638528860e+38$ (positivos e negativos) mas sem precisão exata.

```
float numero = 48.8989;
```


Tipos de dados primitivos

double

O double é um número ponto flutuante de 64 bits e precisão dupla. Para valores decimais, esse tipo de dados geralmente é a escolha padrão.

Campos e variáveis do tipo nativo double podem representar valores numéricos com ponto flutuante que estejam dentro da faixa $4.94065645841246544e-324$ a $1.79769313486231570e+308$ (positivos e negativos) com mais precisão numérica do que o tipo float.

```
double numero = 10.659898941;
```

Tipos de dados primitivos

boolean

O boolean tem apenas dois valores possíveis para uso:

TRUE ou **FALSE**

Use esse tipo de dados para sinalizadores simples que rastreiam condições **verdadeiro/falso**.

Esse tipo de dado representa um bit de informação, mas seu "tamanho" não é algo definido com precisão.

boolean ligado = **TRUE**;

Tipos de dados primitivos

char

O char é um único caractere Unicode de 16 bits.


Tem um valor mínimo de 0 e um valor máximo 65.535 (inclusive).

Caracteres podem ser processados como valores numéricos que são mapeados para letras, dígitos, símbolos e outros.

```
char character = 'R';
```

Operadores

JAVA



Todos os tipos comuns de operadores existentes nas linguagens de programação estruturadas estão presentes em Java.

Veja quais são eles, suas principais características e formas de sua utilização.

Operadores Aritméticos

Operadores aritméticos são aqueles utilizados para efetuar operações matemáticas

OPERADOR	SINTAXE	FUNÇÃO
*	$a * b$	Multiplica a por b
/	a / b	Divide a por b
%	$a \% b$	Resto da divisão de a por b
-	$a - b$	Subtrai a de b
+	$a + b$	Soma a e b

Operadores Aritméticos

ATENÇÃO

Esses operadores são aplicados a tipos numéricos, que possuem tamanhos e características diferentes, conforme seu tipo. Assim, caso você opere com duas variáveis numéricas de tipos diferentes, por exemplo, um int multiplicado por um double, o resultado será sempre do maior tipo; no exemplo anterior, um double.

Operadores de Incremento e Decremento

Operadores de Incremento e Decremento têm a função de aumentar ou diminuir em uma unidade o valor de uma variável.

Deve ser dada atenção especial à posição do operador, pois, dependendo dela, pode-se efetuar a operação desejada antes ou depois de uma avaliação numa determinada expressão ou condição.

Operadores de Incremento e Decremento

OPERADOR	SINTAXE	FUNÇÃO
++	i ++	Incrementa a variável em 1; avalia a expressão antes do incremento
++	++ i	Incrementa a variável em 1; o incremento está antes de avaliar a expressão
--	i --	Decrementa a variável em 1; avalia a expressão antes do decremento
--	-- i	Decrementa a variável em 1, antes de avaliar a expressão

Operadores Relacionais

Operadores Relacionais comparam duas variáveis e determinam seu relacionamento com um resultado lógico, ou seja, **true** (para verdadeiro) ou **false** (para falso).

Operadores Igualdade

Os operadores de igualdade são associados da esquerda para a direita.

$(x == y)$ - x é igual a y

$(x != y)$ x é não igual a y

OPERADOR	SINTAXE	FUNÇÃO
>	a > b	Avalia se a variável a é maior que a variável b (e retorna true ou false , dependendo dos valores de a e b)
>=	a >= b	Avalia se a variável a é maior ou igual à variável b (e retorna true ou false , dependendo dos valores de a e b)
<	a < b	Avalia se a variável a é menor que a variável b (e retorna true ou false , dependendo dos valores de a e b)
<=	a <= b	Avalia se a variável a é menor ou igual à variável b (e retorna true ou false , dependendo dos valores de a e b)
==	a == b	Avalia se a variável a é igual à variável b (e retorna true ou false , dependendo dos valores de a e b)
!=	a != b	Avalia se a variável a é diferente da variável b (e retorna true ou false , dependendo dos valores de a e b)

Operadores Lógicos

Operadores Lógicos avaliam um ou mais operandos lógicos que geram um único valor lógico (**true ou false**) como resultado final da expressão avaliada. Existem seis Operadores Lógicos:

- && - and (operador e) lógico -- & - and (operador e) binário;
- || - or (operador ou) lógico -- | - or (operador ou) binário;
- ^ - ou exclusivo binário;
- ! - operador de negação.

**&&
&
AND**

OPERANDO A	OPERANDO B	RESULTADO
verdadeiro	verdadeiro	verdadeiro
verdadeiro	falso	falso
falso	verdadeiro	falso
falso	falso	falso

**||
|
OR**

OPERANDO A	OPERANDO B	RESULTADO
verdadeiro	verdadeiro	verdadeiro
verdadeiro	falso	verdadeiro
falso	verdadeiro	verdadeiro
falso	falso	falso

^

OPERANDO A	OPERANDO B	RESULTADO
verdadeiro	verdadeiro	falso
verdadeiro	falso	verdadeiro
falso	verdadeiro	verdadeiro
falso	falso	falso

!

OPERANDO A	RESULTADO
verdadeiro	falso
falso	verdadeiro

Precedência de operadores

A precedência de operadores serve para indicar a ordem na qual o compilador interpretará os diferentes tipos de operadores, para que ele sempre tenha como saída um resultado consistente com a regra que quer ser aplicada pelo programador, evitando ambiguidades e inconsistências de operações.

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 (mod)
7	+ soma e - subtração
8	< menor que, <= menor ou igual, > maior que, >= maior ou igual
9	== igual e != diferente
10	& and binário
11	or binário
12	^ ou exclusivo binário
13	&& and lógico
14	or lógico
15	= operador de atribuição

Leitura de dados

JAVA

Classe Scanner

Um Scanner permite a um programa ler os dados (por exemplo, números e strings) para utilização nele. Os dados podem ser provenientes de várias origens, como os digitados pelo usuário ou um arquivo do disco. Antes de utilizar um Scanner, você deve criá-lo e especificar a origem dos dados. Uma variável do tipo instância de (Scanner) input deve ser inicializada na sua declaração.

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

Classe Scanner

Essa expressão utiliza a palavra-chave **new** para criar um **objeto Scanner** que lê caracteres digitados pelo usuário no teclado.

O objeto de entrada padrão, **System.in**, permite que aplicativos leiam bytes de informações digitadas pelo usuário. O Scanner traduz esses bytes em tipos (como tipo inteiro).

```
number = input.nextInt();
```

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

```
public class EntradaDados {
```

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

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

```
        System.out.println(x: "Digite o primeiro numero inteiro: ");
```

```
        int numero1 = entrada.nextInt();
```

```
        System.out.println(x: "Digite o segundo numero inteiro: ");
```

```
        int numero2 = entrada.nextInt();
```

```
        System.out.println(x: "Digite o terceiro numero inteiro: ");
```

```
        int numero3 = entrada.nextInt();
```

```
        System.out.println(x: "Digite o quarto numero inteiro: ");
```

```
        int numero4 = entrada.nextInt();
```

```
        System.out.println("O Primeiro numero inteiro eh: " + numero1);
```

```
        System.out.println("O Segundo numero inteiro eh: " + numero2);
```

```
        System.out.println("O Terceiro numero inteiro eh: " + numero3);
```

```
        System.out.println("O Quarto numero inteiro eh: " + numero4);
```

```
    }
```

```
}
```



Obrigado!

INTRODUÇÃO A LINGUAGEM DE
PROGRAMAÇÃO JAVA