



Quem se prepara, não para.

# Algoritmos e Programação

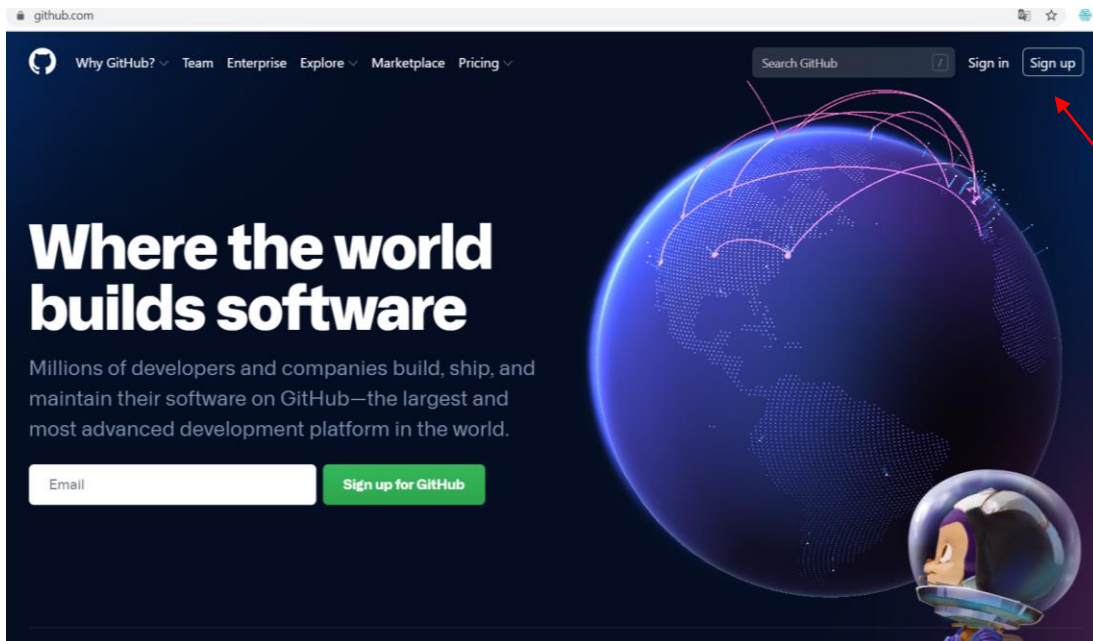
Michelle Hanne

michelle.andrade@newtonpaiva.br

# Sumário

- Criar conta no GitHub
- Criar conta no Replit.com
- Conceitos de Algoritmo
- Java
  - Tipos de Variáveis
  - Operadores
  - Comandos Condicionais
  - Comandos de Loop

# GitHub



<https://github.com/>



Plataforma de hospedagem de código-fonte e arquivos com controle de versão, podendo ser utilizado com projetos públicos ou privados.

Fazer o cadastro ou efetuar login

# GitHub


- Depois de realizar o cadastro é hora de começar a trabalhar. Clicando no botão verde "**Vamos Começar**", você obtém várias dicas de como usar o GitHub.
- Para criar um repositório vá no canto direito, ao lado do seu nome de usuário e clique no botão verde "+Novo Repositório".
- Na página que abrirá em seguida, digite o nome de seu repositório. Faça uma breve descrição. Selecione "**Inicializar este repositório com um README**". E então clique em "**Criar Repositório**".
- É recomendado incluir um README, ou um arquivo com informações sobre o seu projeto.

Após criado o repositório, ficará disponível no endereço:  
**<https://github.com/username/name>**




**Create a new repository**  
A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)


**Owner \*** **Repository name \***

 mihanne /

Great repository names are short and memorable. Need inspiration? How about [symmetrical-fortnight?](#)

**Description (optional)**

☒  **Public**  
Anyone on the internet can see this repository. You choose who can commit.

☐  **Private**  
You choose who can see and commit to this repository.

**Initialize this repository with:**  
Skip this step if you're importing an existing repository.

☐ **Add a README file**  
This is where you can write a long description for your project. [Learn more.](#)

☐ **Add .gitignore**  
Choose which files not to track from a list of templates. [Learn more.](#)

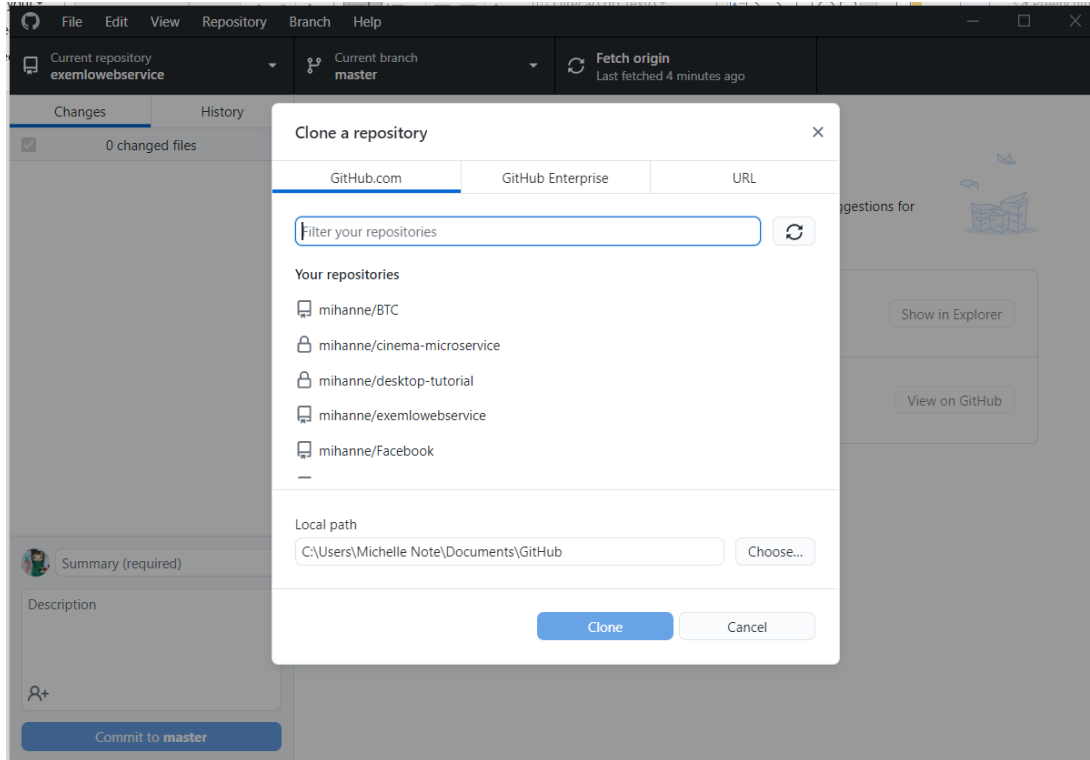
☐ **Choose a license**  
A license tells others what they can and can't do with your code. [Learn more.](#)

# GitHub Desktop

Instalar o GitHub desktop - <https://desktop.github.com>

- Criar uma conta no GitHub
  - É possível subir projetos já existentes para repositórios previamente criados
  - É possível compartilhar projetos entre colegas, cada um atualizando parte do código fonte.
  - É possível usar o GitHub integrado com outros softwares
- 
- Como usar o GitHub desktop  
<https://www.youtube.com/watch?v=Fj3gtbaF8WA>

# GitHub Desktop



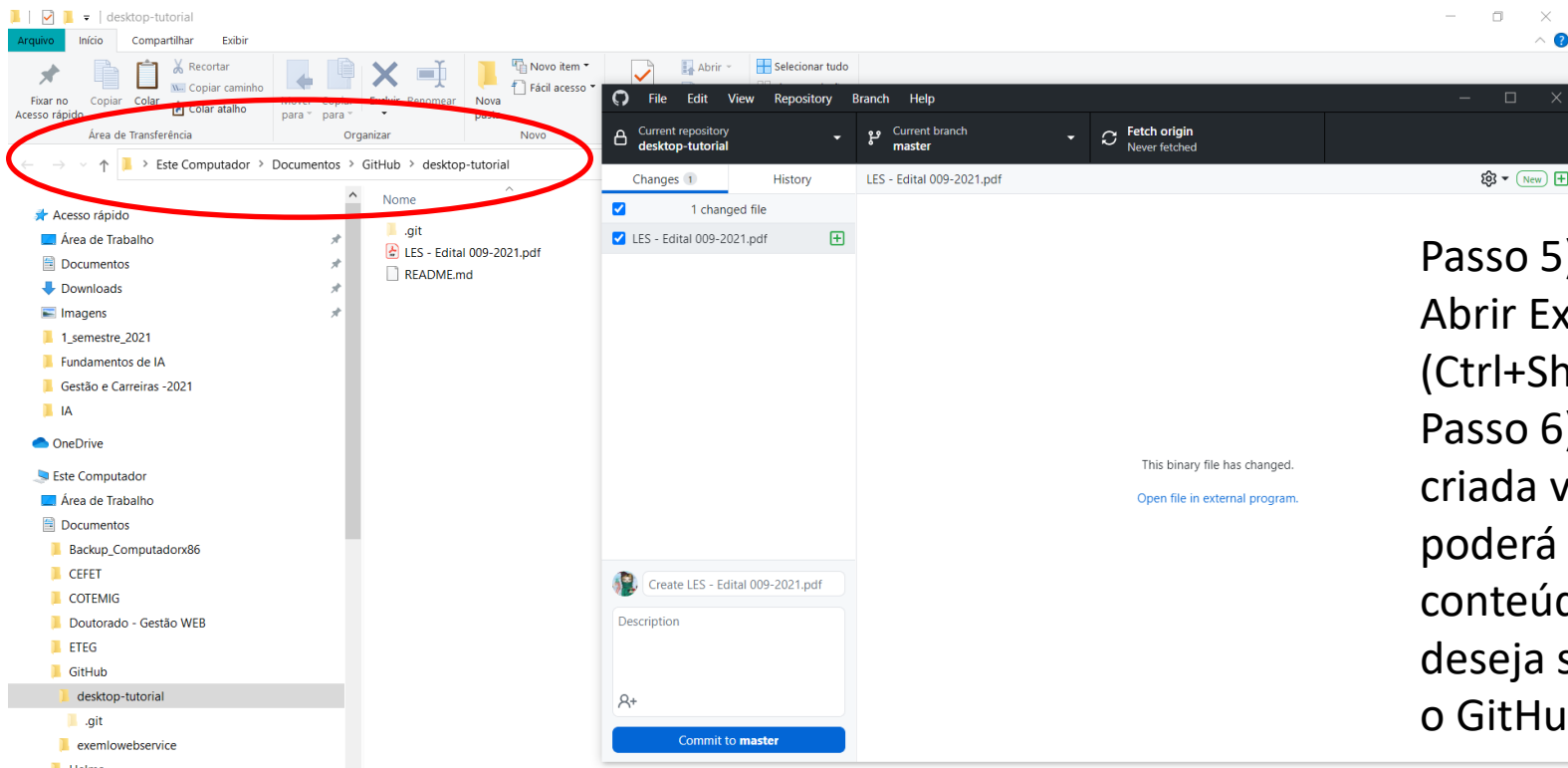
Passo 1) Criar um repositório no GitHub

Passo 2) Acessar o Git Desktop

Passo 3) Clicar em File-> Clone Repository

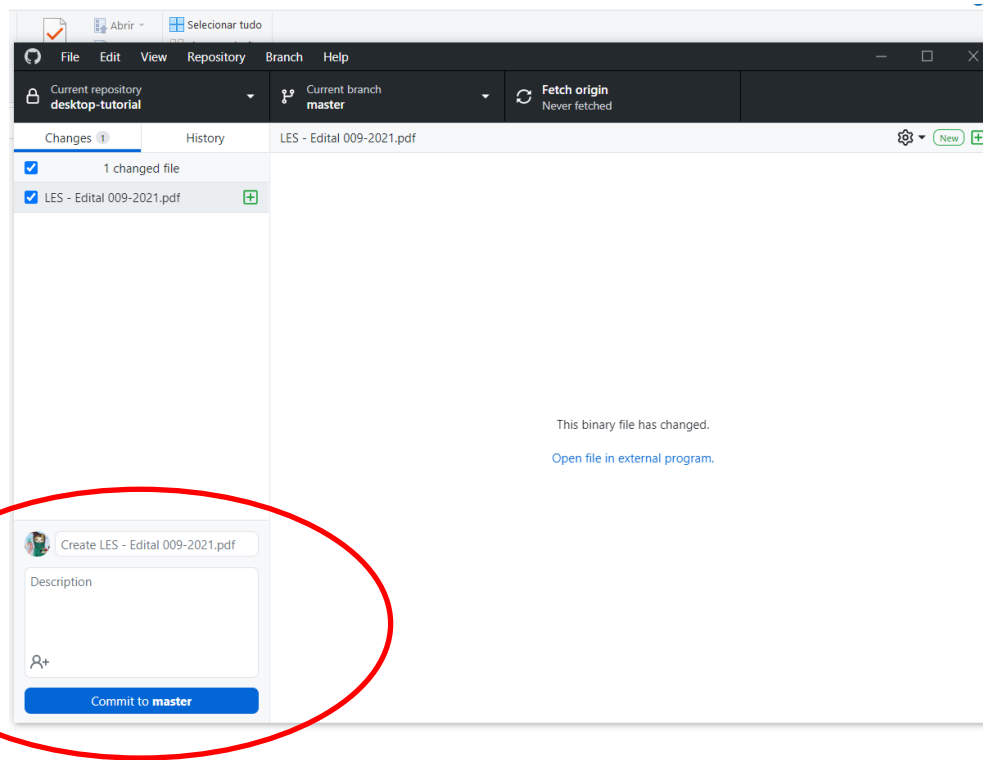
Passo 4) Escolher o repositório criado

# GitHub Desktop



Passo 5) Clicar em  
Abrir Explorer  
(Ctrl+Shift +F)  
Passo 6) Na pasta  
criada você  
poderá copiar o  
conteúdo que  
deseja subir para  
o GitHub

# GitHub Desktop

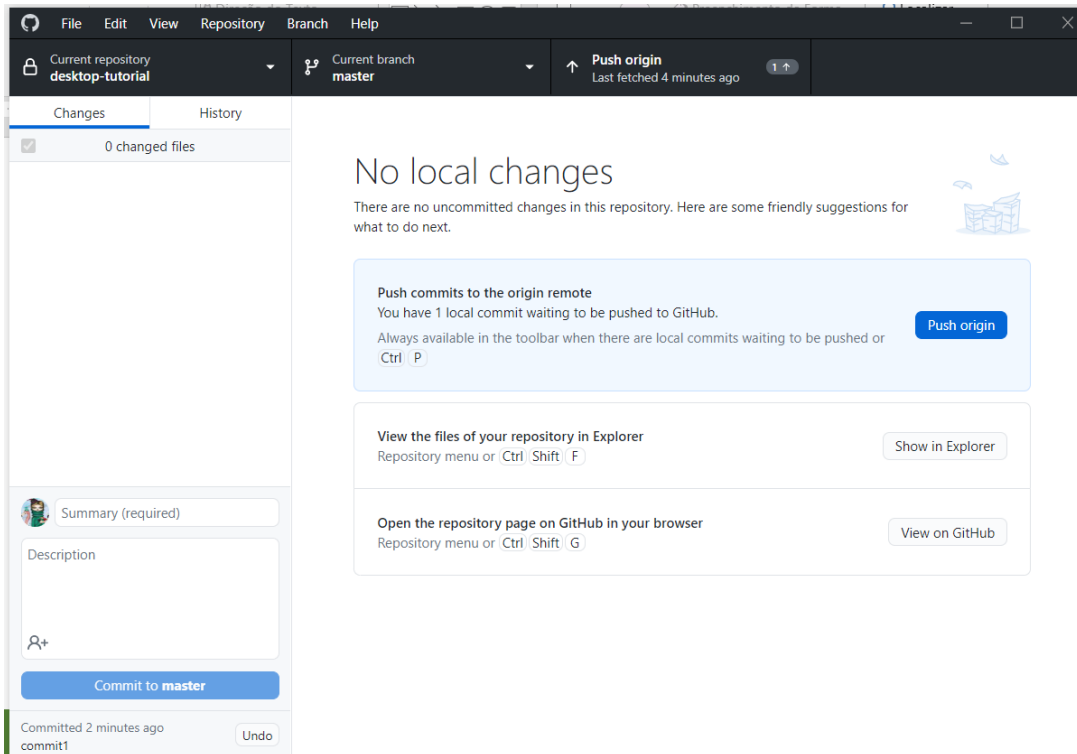


Passo 6) Preencher um título para o Commit e uma descrição (opcional)

Passo 7) Clicar no botão Commit do Master



# GitHub Desktop



Passo 9) Clicar no botão Push origin.

Pronto, o conteúdo foi inserido no seu GitHub.

# GitHub – algumas definições

- Use um branch para isolar o trabalho de desenvolvimento sem afetar outros branches no repositório. Cada repositório tem um **branch padrão** e pode ter vários outros **branches**. Você pode fazer merge de um **branch** em outro **branch** usando uma **pull request**.
- As **pull requests** permitem que você informe outras pessoas sobre as alterações das quais você fez **push** para um **branch** em um repositório no GitHub. Depois que uma **pull request** é aberta, você pode discutir e revisar as possíveis alterações com colaboradores e adicionar **commits** de acompanhamento antes que as alterações sofram merge no branch base.

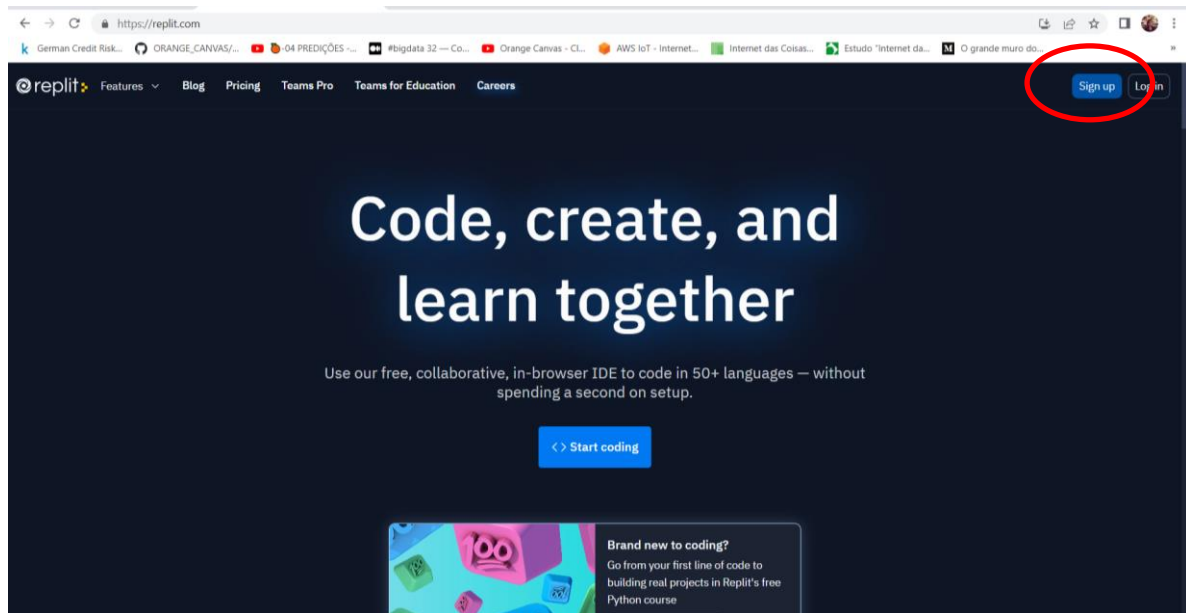
Fonte: <https://docs.github.com/pt/github/collaborating-with-issues-and-pull-requests/about-pull-requests>

# GitHub – Alguns links

- <https://www.youtube.com/watch?v=Fj3gtbaF8WA>
  - <https://www.youtube.com/watch?v=bn4CMzJa0UM>
  - <https://www.youtube.com/watch?v=Ggr7Ov6JMbU>
- 
- Download Github desktop <https://desktop.github.com/>

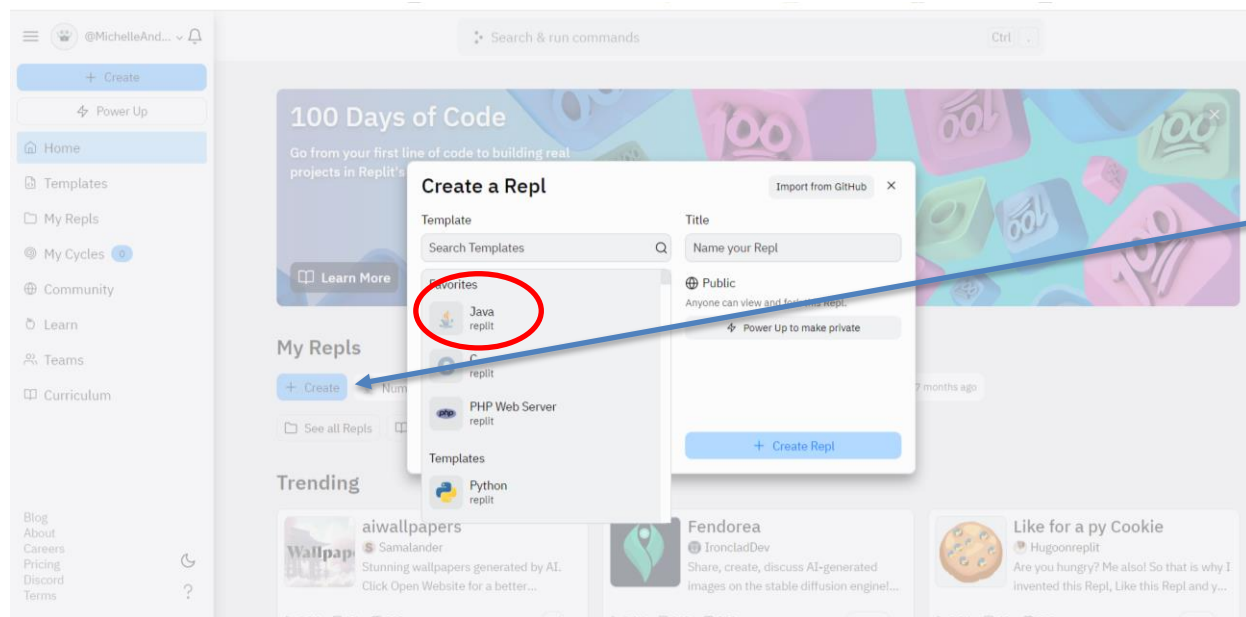
# Replit.com

IDE (Ambiente de Desenvolvimento Integrado) colaborativa que suporta mais de 50 linguagens de programação, executada no browser, usa o conceito de SaaS (Software as a Service)



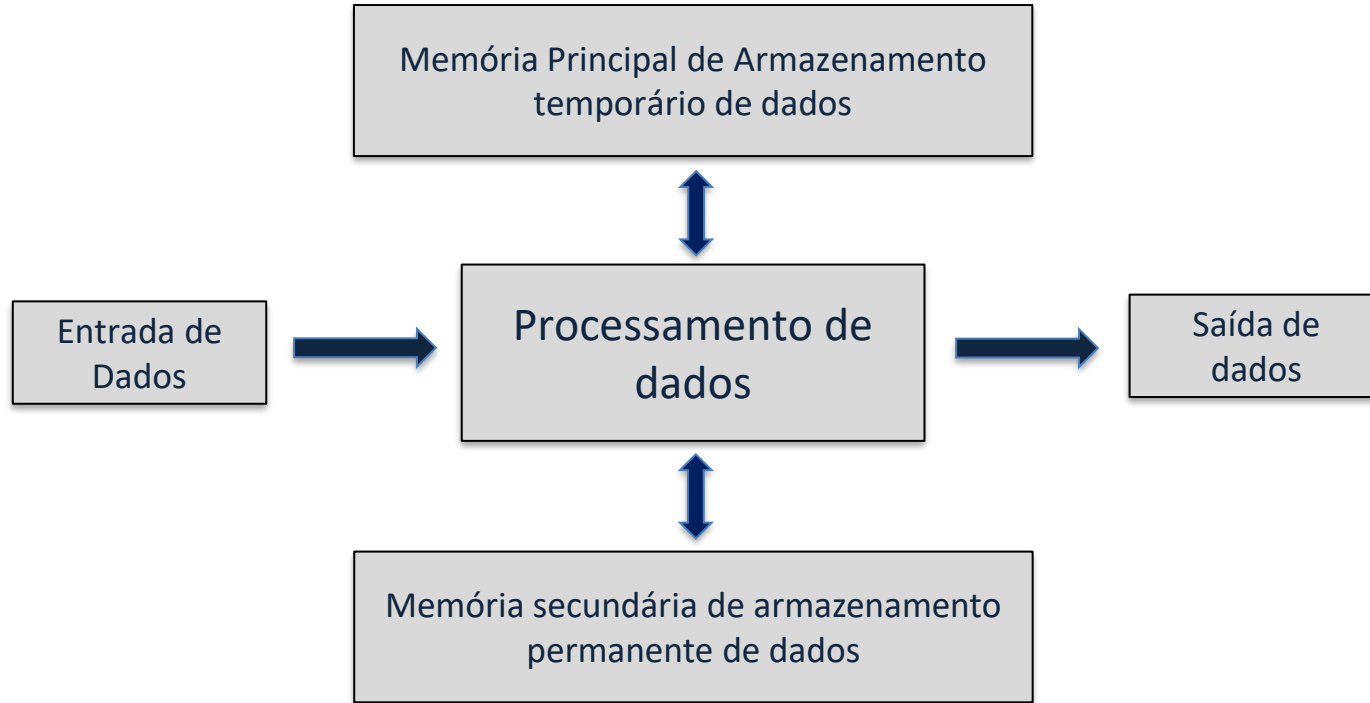
- Clicar em Sign up é possível fazer login com o Gmail, GitHub entre outros.
- Escolha a opção de fazer login com o Github.

# Replit.com



Após efetuar o login clique no botão **+Create**, será mostrado as opções de template disponíveis. Escolha a opção **Java**.

# COMPUTADOR CONTEMPORÂNEO



# Conceitos:

- ✓ Software: São os programas necessários para que o computador execute as tarefas a ele atribuídas.



# Linguagem de Programação:

- Conjunto de convenções e regras que especificam como instruir o computador a executar determinadas tarefas.
- Serve como meio de comunicação entre o indivíduo que deseja resolver um determinado problema e o computador.
- Estabelece ***regras de sintaxe*** para que o algoritmo possa ser entendido por uma máquina.



C# Delphi JAVA  
Pascal COBOL .NET  
C C++



# Conceitos:

As linguagens de programação de alto nível, por sua vez, são aquelas cujas instruções estarão mais próximas da linguagem natural, independentemente do idioma. Para Manzano (2016, p. 22), “as linguagens de alto nível possibilitam maior facilidade de comunicação, pelo fato de serem próximas à comunicação humana, pois se baseiam no idioma inglês”.

**Como exemplos de linguagem de alto nível, é possível citar: C, C++, C#, Java, JavaScript, Python, entre outras.**

# Algoritmos

Para Medina (2006, p. 15), entende-se programa de computador como “um conjunto de instruções que será executado pelo processador em uma determinada sequência e que levará o computador a executar alguma tarefa”.

Segundo Guimarães (2018), programar é basicamente construir algoritmos.

Para Manzano (2016, p. 24), algoritmos “na ciência da computação (informática) está associada a um conjunto de regras e operações bem definidas e ordenadas, destinadas à solução de um problema ou de uma classe de problemas, em um número finito de passos”.

Já para Soffner (2013, p. 21), “algoritmo é um conjunto de passos, passível de repetição, que resolve um problema”.

# Algoritmos

Em outras palavras, algoritmo é **uma sequência – finita – de passos logicamente ordenados** e que resolve determinado problema, sendo escrito em uma pseudolinguagem de programação ou pseudocódigo, com regras muito próximas da linguagem natural, o que justifica o seu uso. Assim, em termos computacionais, todo algoritmo precisará ser reescrito em uma linguagem de programação para ser “entendido” pelo computador.

Um algoritmo pode ser apresentado na forma de texto ou na forma gráfica por meio de blocos. Nos dois casos, será necessário entender o problema a ser resolvido, os dados de entrada, os processamentos necessários, bem como as saídas (resultados) alcançadas (MANZANO, 2016).

# Algoritmos:

A construção de um algoritmo passa pelas seguintes fases:

## **Fase 1**

- Defina bem o problema a ser resolvido.

## **Fase 2**

- Entenda os detalhes do problema.

## **Fase 3**

- Defina os dados de entrada que serão passados para o algoritmo.

## **Fase 4**

- Defina os dados de saída que serão devolvidos pelo algoritmo.

## **Fase 5**

- Defina o processamento, ou seja, os cálculos e os comandos necessários e a ordem destes para executar a tarefa e resolver o problema.

## **Fase 6**

- Construa o algoritmo.

# Exemplos de Algoritmos

## Algoritmo 1 – Trocar o Pneu de um Carro

Passo 1 – Desparafusar a roda



# Exemplos de Algoritmos

## Algoritmo 1 – Trocar o Pneu de um Carro

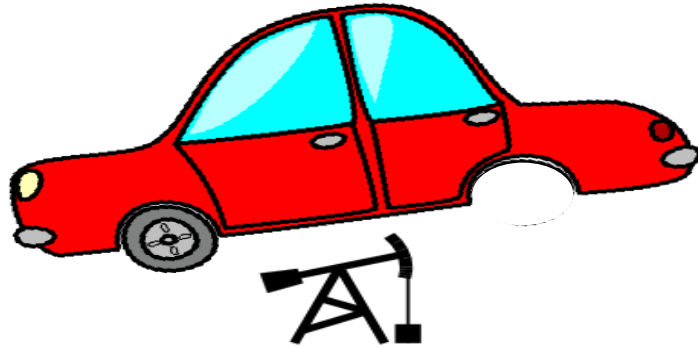
Passo 2 – Suspende o carro com um macaco



# Exemplos de Algoritmos

## Algoritmo 1 – Trocar o Pneu de um Carro

Passo 3 – Retirar a roda com o pneu furado



# Exemplos de Algoritmos

## Algoritmo 1 – Trocar o Pneu de um Carro

Passo 4 – Colocar o Step e os parafusos.

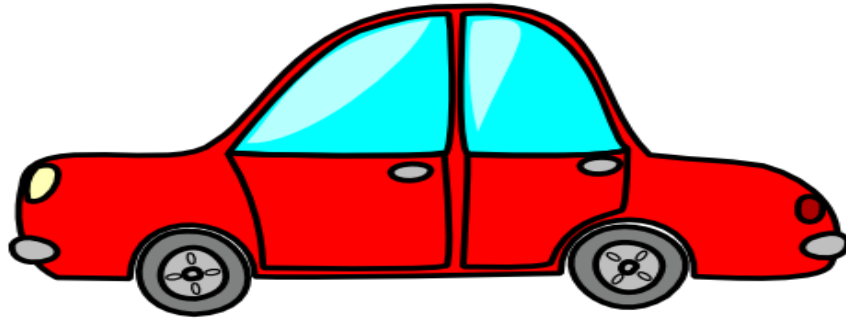




# Exemplos de Algoritmos

## Algoritmo 1 – Trocar o Pneu de um Carro

Passo 5 – Abaixar o carro e apertar os parafusos da roda.



# Construção de Algoritmos – Tipos mais utilizados de Algoritmos

## ➤ Descrição Narrativa:

A descrição narrativa consiste em analisar o enunciado do problema e escrever, utilizando uma linguagem natural ( por exemplo, a língua portuguesa), os passos a serem seguidos para a sua solução.

**Vantagem:** não é necessário aprender nenhum conceito novo, pois a língua natural é bem conhecida.

**Desvantagem:** A língua natural abre espaço para várias interpretações, o que posteriormente dificultará a transcrição desse algoritmo para programa.

## ➤ Fluxograma:

O fluxograma consiste em analisar o enunciado do problema e escrever, utilizando símbolos gráficos predefinidos, os passos a serem seguidos para sua solução.

**Vantagem:** o entendimento de símbolos gráficos é mais simples que o entendimento de textos.

**Desvantagem:** é necessário aprender a simbologia dos fluxogramas e, além disso, o algoritmo resultante não apresenta muitos detalhes, dificultando a sua transcrição para um programa.

## ➤ Pseudocódigo ou Portugal:

O pseudocódigo ou portugal consiste em analisar o enunciado do problema e escrever, por meio de regras predefinidas, os passos a serem seguidos para a sua solução.

**Vantagem:** a passagem do algoritmo para qualquer linguagem de programação é quase imediata, bastando conhecer as palavras reservadas da linguagem que será utilizada.

**Desvantagem:** é necessário aprender as regras do pseudocódigo.

# Construção de Algoritmos – Descrição

## Narrativa

### Algoritmo 1 – Sacar dinheiro no banco 24 horas

Passo 1 – Ir até um banco 24 horas.

Passo 2 – Colocar o cartão.

Passo 3 – Digitar a senha.

Passo 4 – Solicitar a quantia desejada.

Passo 5 – Se o saldo for maior ou igual à quantia desejada, sacar o dinheiro; caso contrário, mostrar uma mensagem de impossibilidade de saque.

Passo 6 – Retirar o cartão.

Passo 7 – Sair do banco 24 horas.

### Algoritmo 1 – Pegar um Uber

Passo 1 – Entra no aplicativo.

Passo 2 – Escolher o local de origem.

Passo 3 – Escolher o local de destino.

Passo 4 – Aguardar o cálculo do valor da corrida.

Passo 5 – Escolher o tipo de transporte.

Passo 6 – Escolher a forma de pagamento e pagar.

Passo 7 – Aguardar o carro.

Passo 8 – Entra no carro e ir até o destino desejado.

# Java

Conforme já mencionado, Java é baseada em C++, mas foi projetada para ser **menor, mais simples e mais confiável**. Java inclui uma forma relativamente simples de controle de concorrência por meio de seu modificador ***synchronized***.

Em Java, é relativamente fácil criar processos concorrentes, chamados de linhas de execução (***threads***).

Java usa liberação implícita de armazenamento para seus objetos, geralmente chamada de coleta de lixo.

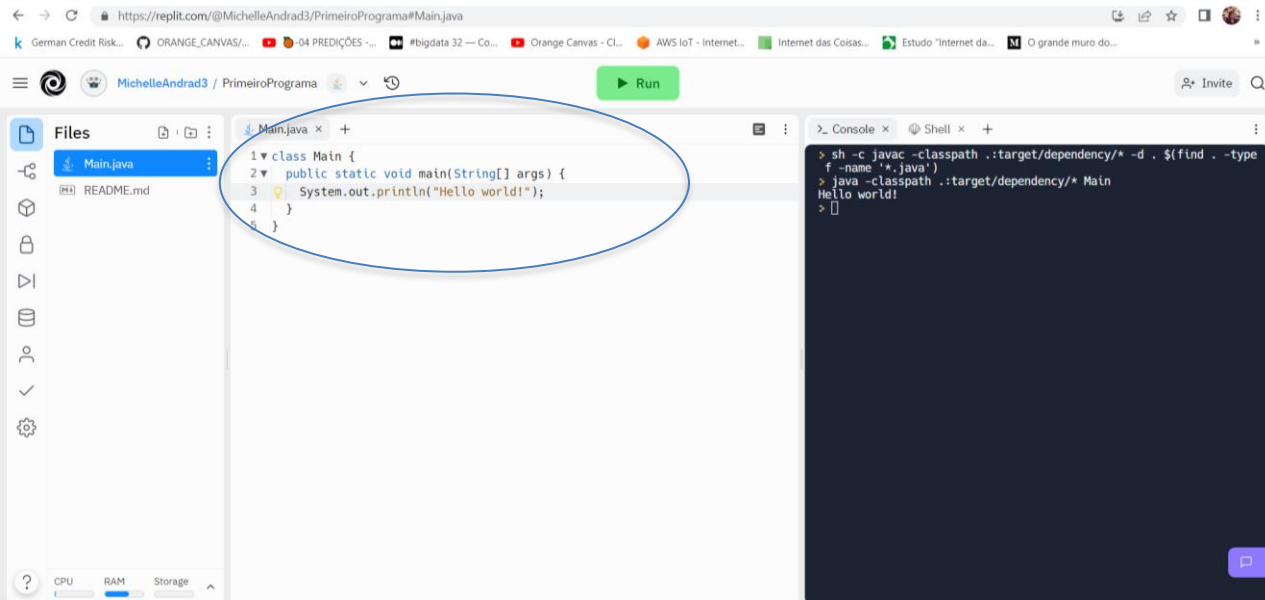
# Ambiente de Programação

Um ambiente de programação é a coleção de ferramentas usadas no desenvolvimento de software. Essa coleção pode consistir em apenas um sistema de arquivos, um editor de textos, um compilador, entre outros recursos

IDE – Ambiente de desenvolvimento integrado de software



# Primeiro Exemplo no Java



The screenshot shows a web-based IDE interface for a Java project named 'PrimeiroPrograma'. On the left, a 'Files' sidebar lists 'Main.java' and 'README.md'. The main editor displays the code for 'Main.java', which is circled in blue. The code is as follows:

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

On the right, a 'Console' window shows the execution output:

```
> sh -c javac -classpath .:target/dependency/* -d . $(find . -type f -name "*.java")  
> java -classpath .:target/dependency/* Main  
Hello world!  
> []
```

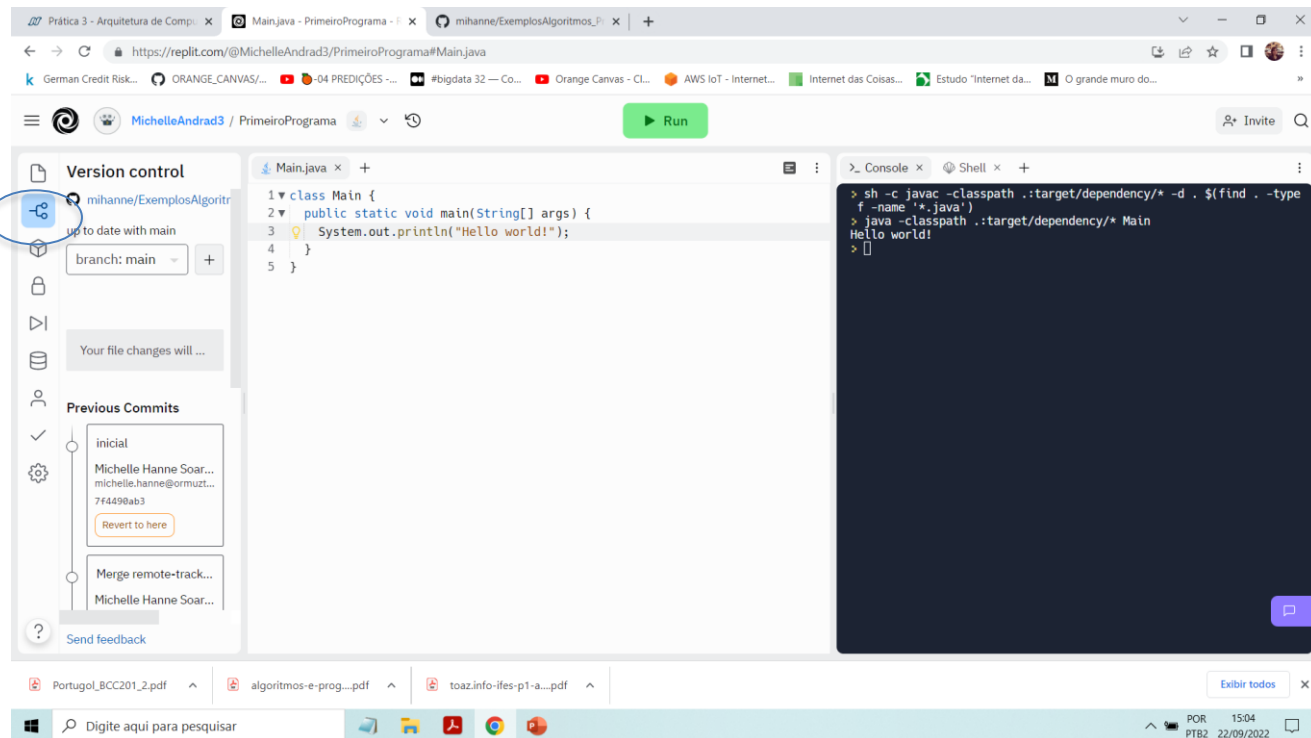
A green 'Run' button is visible at the top of the editor area.

Escolher **um novo arquivo em Java.**

Será mostrado o **primeiro exemplo de código fonte** com o comando `System.out.println("Hello");`

Clique no botão **Run** para ver a execução

# Primeiro Exemplo no Java



Podemos **vincular o Replit com o GitHub**. Basta clicar no ícone **Version Control**.

Recomendo que crie primeiro um repositório no GitHub, **em seguida escolha a opção vincular a conta e aponte par um repositório criado**.

Após codificar você poderá escolher a **opção Comit All & Push**, os arquivos serão enviados para o GitHub

# JAVA

<b>Simples</b>	Java tem um conjunto de recursos conciso e coeso que a torna fácil de aprender e usar.
<b>Segura</b>	Java fornece um meio seguro de criar aplicativos de Internet.
<b>Portável</b>	Os programas Java podem ser executados em qualquer ambiente para o qual houver um sistema de tempo de execução Java.
<b>Orientada a objetos</b>	Java incorpora a moderna filosofia de programação orientada a objetos.
<b>Robusta</b>	Java incentiva a programação sem erros por ser fortemente tipada e executar verificações de tempo de execução.
<b>Várias threads</b>	Java fornece suporte integrado à programação com várias threads.
<b>Neutra quanto à arquitetura</b>	Java não tem vínculos com uma determinada máquina ou arquitetura de sistema operacional.
<b>Interpretada</b>	Java dá suporte a código para várias plataformas com o uso do bytecode.
<b>Alto desempenho</b>	O bytecode Java é altamente otimizado para obtenção de velocidade de execução.
<b>Distribuída</b>	Java foi projetada visando o ambiente distribuído da Internet.
<b>Dinâmica</b>	Os programas Java carregam grandes quantidades de informações de tipo que são usadas na verificação e resolução de acessos a objetos no tempo de execução.

A programação orientada a objetos (OOP, *object oriented programming*) é a essência de Java. A metodologia orientada a objetos é inseparável da linguagem, e todos os programas Java são, pelo menos até certo ponto, orientados a objetos



# JAVA – Tipos Primitivos

**Tabela 2-1** Tipos de dados primitivos internos de Java

Tipo	Significado
boolean	Representa os valores verdadeiro/falso
byte	Inteiro de 8 bits
char	Caractere
double	Ponto flutuante de precisão dupla
float	Ponto flutuante de precisão simples
int	Inteiro
long	Inteiro longo
short	Inteiro curto

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.

# Tipo Inteiros

## Inteiros

Java define quatro tipos inteiros: **byte**, **short**, **int** e **long**, que são mostrados aqui

Tipo	Tamanho em bits	Intervalo
byte	8	-128 a 127
short	16	-32.768 a 32.767
int	32	-2.147.483.648 a 2.147.483.647
long	64	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807

Como a tabela mostra, **todos os tipos inteiros são valores de sinal positivo e negativo**. Java não suporta inteiros sem sinal (somente positivos).

O tipo float tem 32 bits e o tipo double tem 64 bits.

## 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

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

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

```
class Main {
    public static void main(String[] args) {
        double resultado;
        double resto;
        resultado= 10/3;
        resto = 10%3;
        System.out.println("Resultado da Divisão de 10
por 3: " + resultado);
        System.out.println("Resto da Divisão de 10 por 3:
" + resto);

    }
}
```

## Operadores Relacionais e lógicos

Nos termos operador relacional e operador lógico, 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
= =	Igual a
!=	Diferente de
>	Maior que
<	Menor que
>=	Maior ou igual a
<=	Menor ou igual a

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

Podemos comparar todos os objetos para ver 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

Nos termos operador relacional e operador lógico, 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.

		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

```
// Demonstra os operadores relacionais e lógicos.
class RelLogOps {
    public static void main(String args[]) {
        int i, j; boolean b1, b2;
        i = 10;
        j = 11;
        if(i < j) System.out.println("i < j");
        if(i <=j) System.out.println("i <= j");
        if(i != j) System.out.println("i != j");
        if(i == j) System.out.println("this won't execute");
        if(i >= j) System.out.println("this won't execute");
        if(i > j) System.out.println("this won't execute");

        b1 = true;
        b2 = false;
        if(b1 & b2)
            System.out.println("this won't execute");
        if(!(b1 & b2)) System.out.println("!(b1 & b2) is true");
        if(b1 | b2) System.out.println("b1 | b2 is true");
        if(b1 ^ b2) System.out.println("b1 ^ b2 is true");
    }
}
```

## Incremento e Decremento

`++` e `--` são os operadores Java de incremento e decremento.

```
x = x + 1;  
é o mesmo que  
x++;
```

```
e  
x = x - 1;  
é o mesmo que  
x--;
```

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 posfixada
```



## Atribuição Abreviada

O par de operadores += solicita ao compilador que atribua a x o valor de x mais 10.

Veja outro exemplo. A instrução

`x = x - 100;`

é igual a

`x -= 100;`

+=	-=	*=	/=
%=	&=	=	^=

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

## 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  
}
```

## A escada 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;
```

## Instrução 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**.

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

A expressão que controla switch deve ser de tipo byte, short, int, char ou uma enumeração. A partir do JDK 7, a expressão também pode ser de tipo String.

```
// Demonstra switch.
class SwitchDemo {
    public static void main(String args[]) {
        int i;
        for(i=0; i<10; i++)
            switch(i) {
                case 0:
                    System.out.println("i is zero");
                    break;
                case 1:
                    System.out.println("i is one");
                    break;
                case 2:
                    System.out.println("i is two");
                    break;
                case 3:
                    System.out.println("i is three");
                    break;
                case 4:
                    System.out.println("i is four");
                    break;
                default:
                    System.out.println("i is five or more");
            }
    }
}
```

## Instrução Switch

### Instruções switch aninhadas:

- É possível um switch fazer parte da sequência de instruções de um switch externo. **Isso é chamado de switch aninhado.**

## Instrução 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*

*}*

```
// Exibe as raízes quadradas de 1 a 99 e o erro de
arredondamento.
class SqrRoot {
    public static void main(String args[]) {
        double num, sroot, rerr;
        for(num = 1.0; num < 100.0; num++) {
            sroot = Math.sqrt(num);
            System.out.println("Square root of " + num + "
is " + sroot);
            // calcula o erro de arredondamento
            rerr = num - (sroot * sroot);
            System.out.println("Rounding error is " +
rerr);
            System.out.println();
        }
    }
}
```

## 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.

```
// Use vírgulas em uma instrução for.
class Comma {
    public static void main(String args[]) {
        int i, j;
        for(i=0, j=10; i < j; i++, j--)
            System.out.println("i and j: " + i + " " + j);
    }
}
```

## 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. class ForTest {  
    public static void main(String args[]) throws java.io.IOException  
{  
        int i;  
        System.out.println("Press S to stop.");  
        for(i = 0; (char) System.in.read() != 'S'; i++)  
            System.out.println("Pass #" + i);  
    }  
}
```



## 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.
class Empty {
    public static void main(String args[]) {
        int i;
        for(i = 0; i < 10; ) {
            System.out.println("Pass #" + i);
            i++; // incrementa a variável de controle de laço
        }
    }
}
```

## 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
{
// ...
}
```

## 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.
class Empty3 {
    public static void main(String args[]) {
        int i;

        int sum = 0;
        // soma os números até 5
        for(i = 1; i <= 5; sum += i++) ;
        System.out.println("Sum is " + sum);
    }
}
```

# Referências

SANTOS, Marcela Gonçalves dos. **Linguagem de programação**. SAGAH, 2018. ISBN digital: 9788595024984.

SEBESTA, Robert W. **Conceitos de Linguagens de Programação**. Bookman, 2018. ISBN digital: 9788582604694.

SCHILDT, Herbert. Java para iniciantes. Disponível em: Minha Biblioteca, (6th edição). Grupo A, 2015.

SILVA, Fabricio Machado da. **Paradigmas de programação**. SAGAH, 2019. ISBN digital: 9788533500426.