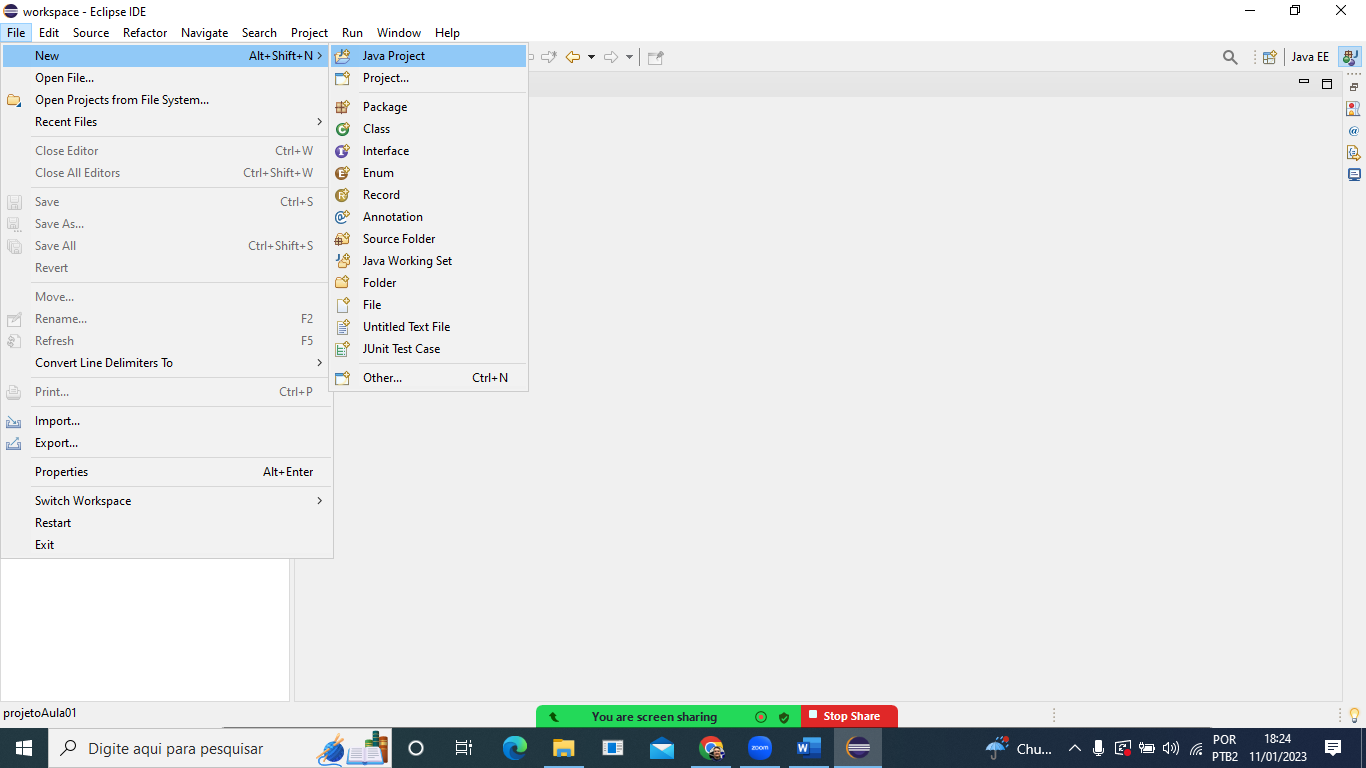
**Abrindo o eclipse:**

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Criando um projeto:

**File / New / Java Project**



Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

**packages**

São “pacotes” onde podemos incluir classes ou qualquer tipo de artefato de programação criado em Java.

Exemplo:

* **principal**

Pacote criado para inserir a classe que irá executar o projeto Java, também chamada de **Program.java**

* **entities**

Pacote utilizado para criar as classes de entidade do projeto, ou seja, classes que modelam dados que compõem o sistema (Ex: Cliente)

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo, Word

Descrição gerada automaticamente

/principal/Program.java

É a classe para executar o projeto, e isto é feito por meio de um método chamado **void main**

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

**package** principal;

**public** **class** Program {

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

}

}

**Entidades**

São classes utilizadas para modelagem de dados no projeto Java. O seu objetivo é modelar os dados de algum substantivo do projeto.

Exemplo: Cliente, Produto, Funcionario etc.

Em Java, essas classes de entidade são criadas seguindo um padrão chamado de **JavaBean**.

**JavaBean**

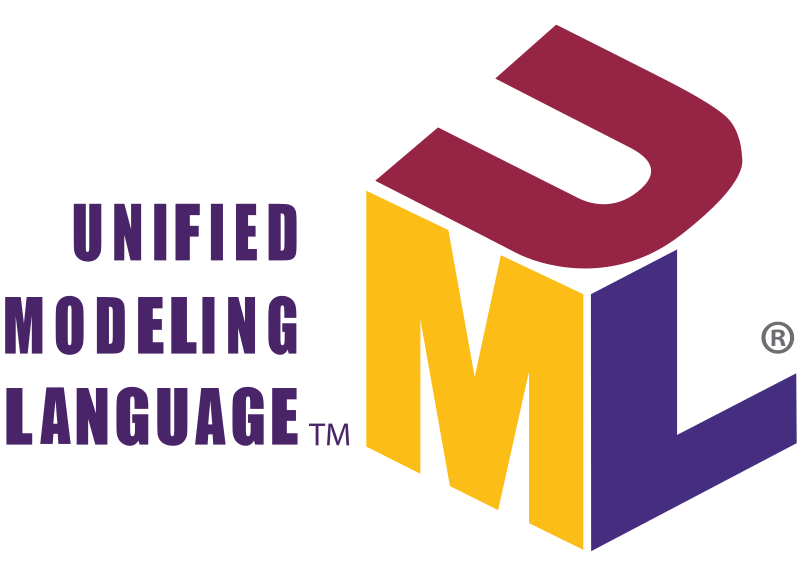
Padrão para construção de classes Java que tem como objetivo fazer a representação de dados do mundo real no paradigma orientado a objetos.

Regras do padrão JavaBean:

* Atributos da classe possuem visibilidade “private”;
* Métodos de encapsulamento para cada atributo, estes métodos são chamados de set e get.
* Métodos construtores
  + Construtor default (Sem argumentos)
  + Construtor com entrada de argumentos
* Sobrescrita de métodos
  + Implementar métodos da superclasse Object

**UML – Unified Modeling Language**

Linguagem visual para modelagem de sistemas orientados a objetos. Define uma série de diagramas para mostrar os aspectos de um sistema orientado a objetos de vários pontos de vista.



**Visual Paradigm**

<https://online.visual-paradigm.com/>

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Criando uma classe de

entidade para Funcionario:

Interface gráfica do usuário, Aplicativo, Tabela, Excel

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

**Declarando os atributos da classe:**

Atributos da classe possuem visibilidade “private”;

**package** entities;

**public** **class** Funcionario {

**private** Integer idFuncionario;

**private** String nome;

**private** String matricula;

**private** String cpf;

}

**Gerando os métodos set e get através da IDE:**

Métodos de encapsulamento para cada atributo, estes métodos são chamados de set e get.

* SOURCE / GENERATED GETTERS AND SETTERS

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Tecla de atalho para formatação do código (endentação).

**CTRL + SHIFT + F**

Tecla de atalho para importação automática de classes.

**CTRL + SHIFT + O**

Tecla de atalho para salvar o código.

**CTRL + S**

**package** entities;

**public** **class** Funcionario {

**private** Integer idFuncionario;

**private** String nome;

**private** String matricula;

**private** String cpf;

**public** Integer getIdFuncionario() {

**return** idFuncionario;

}

**public** **void** setIdFuncionario(Integer idFuncionario) {

**this**.idFuncionario = idFuncionario;

}

**public** String getNome() {

**return** nome;

}

**public** **void** setNome(String nome) {

**this**.nome = nome;

}

**public** String getMatricula() {

**return** matricula;

}

**public** **void** setMatricula(String matricula) {

**this**.matricula = matricula;

}

**public** String getCpf() {

**return** cpf;

}

**public** **void** setCpf(String cpf) {

**this**.cpf = cpf;

}

}

Voltando na classe Program.java

Criando uma rotina para ler os dados de um funcionário, informado pelo usuário do projeto.

**Objeto**

Consiste em uma variável que contêm a referência de uma classe, ou seja, a instância ou espaço de memória de uma classe.

Exemplo:

**Funcionario funcionario = new Funcionario();**

[Classe] [Objeto] [Inicializando o objeto]

(Variável de instância) (Construtor)

Scanner

Classe antiga do Java utilizada para leitura e captura de dados no prompt de comandos. Não é utilizada em projetos web, mas apenas em projetos Java local.

**package** principal;

**import** java.util.Scanner;

**import** entities.Funcionario;

**public** **class** Program {

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

//syso + CTRL + espaço

System.***out***.println

("\n \*\*\* CADASTRO DE FUNCIONÁRIO \*\*\* \n");

Funcionario funcionario = **new** Funcionario();

Scanner scanner = **new** Scanner(System.***in***);

System.***out***.print("Entre com o id do funcionário.....: ");

funcionario.setIdFuncionario

(Integer.*parseInt*(scanner.nextLine()));

System.***out***.print("Entre com o nome do funcionário...: ");

funcionario.setNome(scanner.nextLine());

System.***out***.print("Entre com a matrícula.............: ");

funcionario.setMatricula(scanner.nextLine());

System.***out***.print("Entre com o cpf...................: ");

funcionario.setCpf(scanner.nextLine());

}

}

Resultado:

**\*\*\* CADASTRO DE FUNCIONÁRIO \*\*\***

**Entre com o id do funcionário.....: 1**

**Entre com o nome do funcionário...: Sergio Mendes**

**Entre com a matrícula.............: 2022-ABC**

**Entre com o cpf...................: 123.456.789-00**

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

**package** principal;

**import** java.util.Scanner;

**import** entities.Funcionario;

**public** **class** Program {

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

//syso + CTRL + espaço

System.***out***.println

("\n \*\*\* CADASTRO DE FUNCIONÁRIO \*\*\* \n");

Funcionario funcionario = **new** Funcionario();

Scanner scanner = **new** Scanner(System.***in***);

System.***out***.print("Entre com o id do funcionário.....: ");

funcionario.setIdFuncionario

(Integer.*parseInt*(scanner.nextLine()));

System.***out***.print("Entre com o nome do funcionário...: ");

funcionario.setNome(scanner.nextLine());

System.***out***.print("Entre com a matrícula.............: ");

funcionario.setMatricula(scanner.nextLine());

System.***out***.print("Entre com o cpf...................: ");

funcionario.setCpf(scanner.nextLine());

//Imprimir as informações do funcionário..

System.***out***.println("\nDADOS DO FUNCIONÁRIO:");

System.***out***.println("\tID........: "

+ funcionario.getIdFuncionario());

System.***out***.println("\tNOME......: "

+ funcionario.getNome());

System.***out***.println("\tMATRICULA.: "

+ funcionario.getMatricula());

System.***out***.println("\tCPF.......: "

+ funcionario.getCpf());

}

}

**Executando:**

Texto

Descrição gerada automaticamente com confiança média

Resultado:

**\*\*\* CADASTRO DE FUNCIONÁRIO \*\*\***

**Entre com o id do funcionário.....: 1**

**Entre com o nome do funcionário...: Sergio Mendes**

**Entre com a matrícula.............: 2022-0001**

**Entre com o cpf...................: 123.456.789-00**

**DADOS DO FUNCIONÁRIO:**

**ID........: 1**

**NOME......: Sergio Mendes**

**MATRICULA.: 2022-0001**

**CPF.......: 123.456.789-00**

**Diagrama de Sequência:**

Mostra as camadas (classes) e fluxo da execução do projeto:

Interface gráfica do usuário

Descrição gerada automaticamente

**Desenvolvimento baseado em camadas**

Para cada tarefa que um projeto precisa realizar, podemos criar pacotes e classes específicos para implementar cada tipo de tarefa.

É comum em um projeto Java encontrarmos pacotes com os nomes:

* entities
* repositories
* controllers
* dtos
* services
* etc...

**Repository**

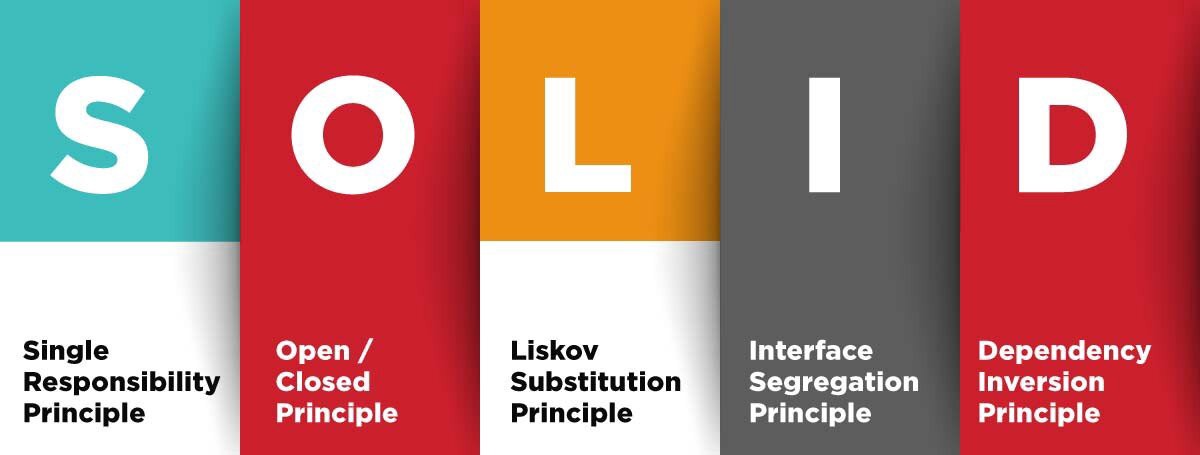
Nome dado para classes de um projeto que acessam algum meio de armazenamento físico de dados, como por exemplo bancos de dados, arquivos em disco etc.

Estas classes chamadas de Repositórios irão gravar, ler, alterar, excluir dados em algum banco de dados ou arquivo em disco.

**SOLID**

Os princípios **SOLID** devem ser aplicados no desenvolvimento de software de forma que o software produzido tenha as seguintes características:

* **Seja fácil de manter, adaptar e se ajustar às constantes mudanças exigidas pelos clientes;**
* **Seja fácil de entender e testar;**
* **Seja construído de forma a estar preparado para ser facilmente alterado com o menor esforço possível;**
* **Seja possível de ser reaproveitado;**
* **Exista em produção o maior tempo possível;**
* **Que atenda realmente as necessidades dos clientes para o qual foi criado;**



**SRP – Princípio da responsabilidade única**

Na programação, o **Princípio da responsabilidade única** declara que cada módulo ou classe deve ter responsabilidade sobre uma única parte da funcionalidade fornecida pelo software.

Como esse princípio nos ajuda a criar um software melhor? Vamos ver alguns dos seus benefícios:

1. **Teste** — Uma classe com uma responsabilidade terá muito menos casos de teste
2. **Menor acoplamento**— menos funcionalidade em uma única classe terá menos dependências
3. **Organização** — Classes menores e bem-organizadas são mais fáceis de pesquisar do que as classes monolíticas

Criando uma classe para gravarmos os dados do funcionário em um arquivo de extensão .TXT

Interface gráfica do usuário, Aplicativo, Tabela, Excel

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

**package** repositories;

**import** entities.Funcionario;

**public** **class** FuncionarioRepository {

//método para gravar os dados de um funcionário em arquivo

**public** **void** exportar(Funcionario funcionario) {

//**TODO**

}

}

**Tratamento de exceções**

Consiste em criarmos rotinas em Java que possam capturar **erros que ocorrem em tempo de execução** nos projetos.

\*\* Primeiro, ao criar métodos em Java precisamos sinalizar que o método pode gerar erros em tempo de exceção, ou em outras palavras, precisamos dizer que um método pode **lançar exceções**.

Exceções

São erros que ocorrem em tempo de execução.

**throws Exception**

Indica que o método, ao ser executado, precisa de tratamento de erros (tratamento de exceções).

Exemplo:

**class** Calculadora {

**public** Double somar(Double numero1, Double numero2)

**throws Exception {**

**return** numero1 + numero2;

}

}

**class** Test {

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

Calculadora calculadora = **new** Calculadora();

**try {**

**Double resultado = calculadora.somar(10.0, 20.0);**

System.***out***.println("A soma é: " + resultado);

**}**

**catch(Exception e) {**

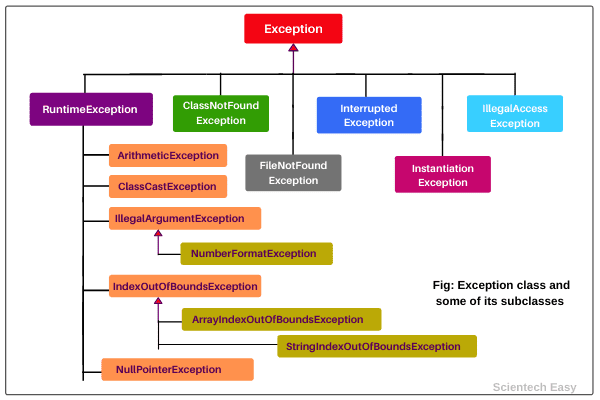
System.***out***.println("Ocorreu um erro!");

e.printStackTrace();

**}**

}

}



**package** repositories;

**import** java.io.PrintWriter;

**import** entities.Funcionario;

**public** **class** FuncionarioRepository {

// método para gravar os dados de um funcionário em arquivo

**public** **void** exportar(Funcionario funcionario) **throws** Exception {

// criando um arquivo em modo de escrita

PrintWriter printWriter

= **new** PrintWriter("c:\\temp\\funcionario.txt");

// escrever o conteudo do arquivo

printWriter.write("\nDADOS DO FUNCIONÁRIO:");

printWriter.write

("\nID.........: " + funcionario.getIdFuncionario());

printWriter.write

("\nNOME.......: " + funcionario.getNome());

printWriter.write

("\nMATRICULA..: " + funcionario.getMatricula());

printWriter.write

("\nCPF........: " + funcionario.getCpf());

// salvar o arquivo

printWriter.flush();

// fechar o arquivo

printWriter.close();

}

}

Voltando na classe **Program.java**

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

package principal;

import java.util.Scanner;

import entities.Funcionario;

import repositories.FuncionarioRepository;

public class Program {

public static void main(String[] args) {

//syso + CTRL + espaço

System.out.println("\n \*\*\* CADASTRO DE FUNCIONÁRIO \*\*\* \n");

Funcionario funcionario = new Funcionario();

Scanner scanner = new Scanner(System.in);

System.out.print("Entre com o id do funcionário.....: ");

funcionario.setIdFuncionario(Integer.parseInt(scanner.nextLine()));

System.out.print("Entre com o nome do funcionário...: ");

funcionario.setNome(scanner.nextLine());

System.out.print("Entre com a matrícula.............: ");

funcionario.setMatricula(scanner.nextLine());

System.out.print("Entre com o cpf...................: ");

funcionario.setCpf(scanner.nextLine());

try {

//criando um objeto para a classe FuncionarioRepository

FuncionarioRepository funcionarioRepository

= new FuncionarioRepository();

//gravando os dados do funcionário em arquivo

funcionarioRepository.exportar(funcionario);

//imprimindo..

System.out.println

("\nFUNCIONÁRIO CADASTRADO COM SUCESSO!");

System.out.println("\tID........: " + funcionario.getIdFuncionario());

System.out.println("\tNOME......: " + funcionario.getNome());

System.out.println("\tMATRICULA.: " + funcionario.getMatricula());

System.out.println("\tCPF.......: " + funcionario.getCpf());

}

catch(Exception e) {

//imprimindo..

System.out.println("\nFALHA AO CADASTRAR FUNCIONÁRIO!");

//imprimir o log do erro

e.printStackTrace();

}

}

}

**Executando:**

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

**\*\*\* CADASTRO DE FUNCIONÁRIO \*\*\***

**Entre com o id do funcionário.....: 1**

**Entre com o nome do funcionário...: Sergio Mendes**

**Entre com a matrícula.............: 2022-1234**

**Entre com o cpf...................: 123.456.789-00**

**FUNCIONÁRIO CADASTRADO COM SUCESSO!**

**ID........: 1**

**NOME......: Sergio Mendes**

**MATRICULA.: 2022-1234**

**CPF.......: 123.456.789-00**

Arquivo gravado:

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

Sobrecarga de métodos (**Overloading**)

É uma prática em Java onde o desenvolvedor cria métodos em uma classe que possuem o mesmo nome, porém, com entrada de argumentos diferentes:

**class** Impressao {

**public void imprimir() {**

System.***out***.println("Olá!");

}

**public void imprimir(String nome) {**

System.***out***.println("Olá, " + nome + "!");

}

**public void imprimir(String nome, String sobrenome) {**

System.***out***.println("Olá, " + nome + " "

+ sobrenome + "!");

}

}

Dessa forma, podemos chamar os métodos da seguinte forma:

**class** Test {

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

**Impressao impressao = new Impressao();**

**impressao.imprimir("Ana", "Paula");**

}

}

Sempre que você for chamar um método de uma classe, verifique se este método possui outros com o mesmo nome, ou seja, se possui sobrecarga:

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

**package** repositories;

**import** java.io.File;

**import** java.io.FileOutputStream;

**import** java.io.PrintWriter;

**import** entities.Funcionario;

**public** **class** FuncionarioRepository {

// método para gravar os dados de um funcionário em arquivo

**public** **void** exportar(Funcionario funcionario) **throws** Exception {

// criando um arquivo em modo de escrita

**PrintWriter printWriter = new PrintWriter**

**(new FileOutputStream**

**(new File("c:\\temp\\funcionario.txt"), true));**

// escrever o conteudo do arquivo

printWriter.write("\nDADOS DO FUNCIONÁRIO:");

printWriter.write("\nID.........: "

+ funcionario.getIdFuncionario());

printWriter.write("\nNOME.......: "

+ funcionario.getNome());

printWriter.write("\nMATRICULA..: "

+ funcionario.getMatricula());

printWriter.write("\nCPF........: "

+ funcionario.getCpf());

// salvar o arquivo

printWriter.flush();

// fechar o arquivo

printWriter.close();

}

}

Executando:

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

**\*\*\* CADASTRO DE FUNCIONÁRIO \*\*\***

**Entre com o id do funcionário.....: 3**

**Entre com o nome do funcionário...: João Paulo**

**Entre com a matrícula.............: 2022-12345**

**Entre com o cpf...................: 123.432.234-00**

**FUNCIONÁRIO CADASTRADO COM SUCESSO!**

**ID........: 3**

**NOME......: João Paulo**

**MATRICULA.: 2022-12345**

**CPF.......: 123.432.234-00**

Arquivo gerado com sucesso:

Tela de computador com texto preto sobre fundo branco

Descrição gerada automaticamente

**GIT HUB**

É o maior repositório de projetos da web. Utilizado por desenvolvedores para publicar o código fonte de seus projetos e compartilhá-lo com outros desenvolvedores.

<https://github.com/>

Tela de computador

Descrição gerada automaticamente

<https://github.com/signup?ref_cta=Sign+up&ref_loc=header+logged+out&ref_page=%2F&source=header-home>

Tela de computador

Descrição gerada automaticamente

**GIT BASH**

<https://gitforwindows.org/>

Terminal de linhas de comando para utilizarmos os recursos do GITHUB. Através dele poderemos publicar código no GIT, clonar projeto, criar branches etc.

Interface gráfica do usuário, Aplicativo

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente

Interface gráfica do usuário, Texto, Aplicativo

Descrição gerada automaticamente