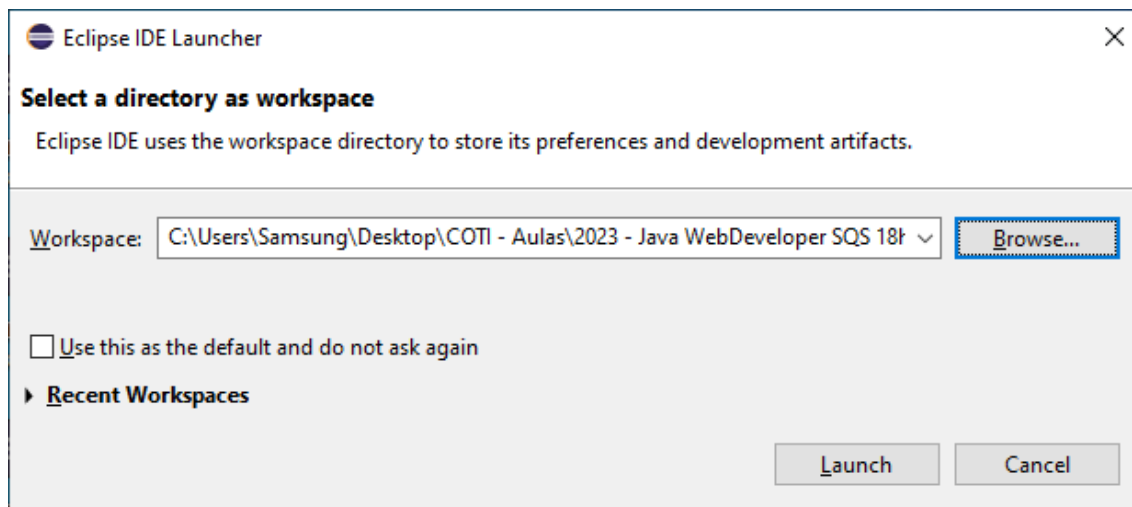
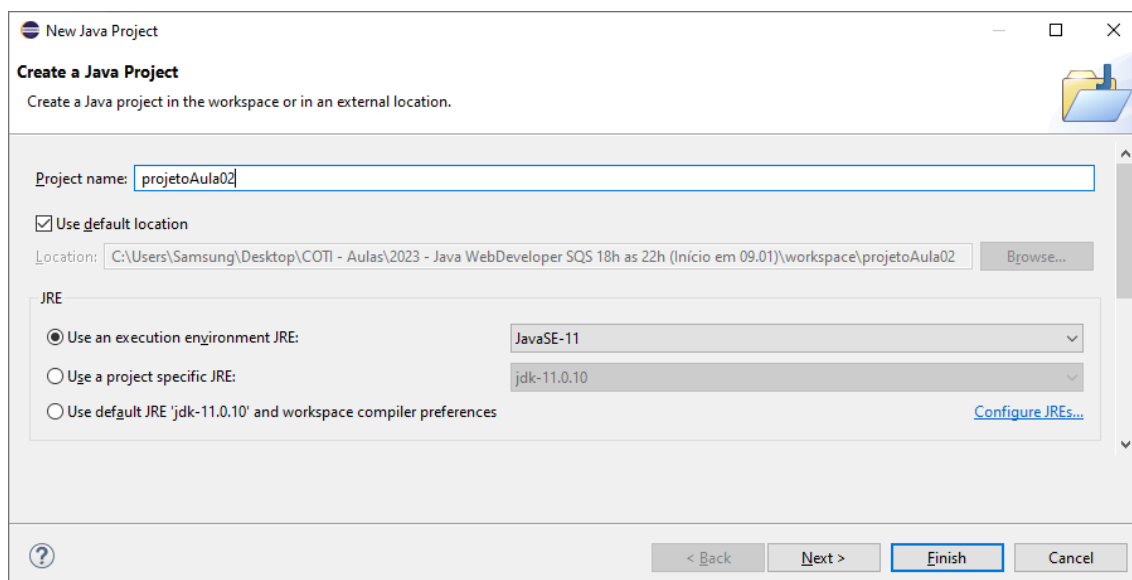
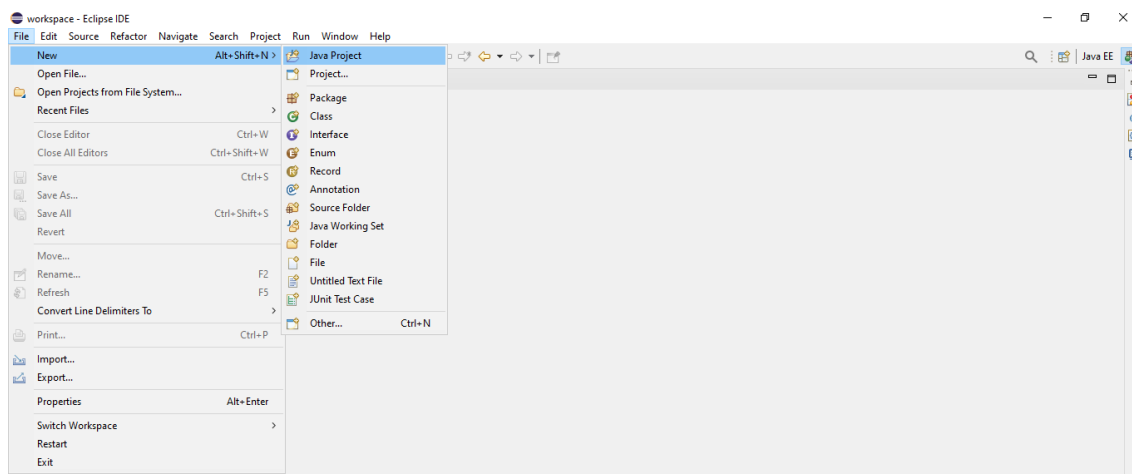
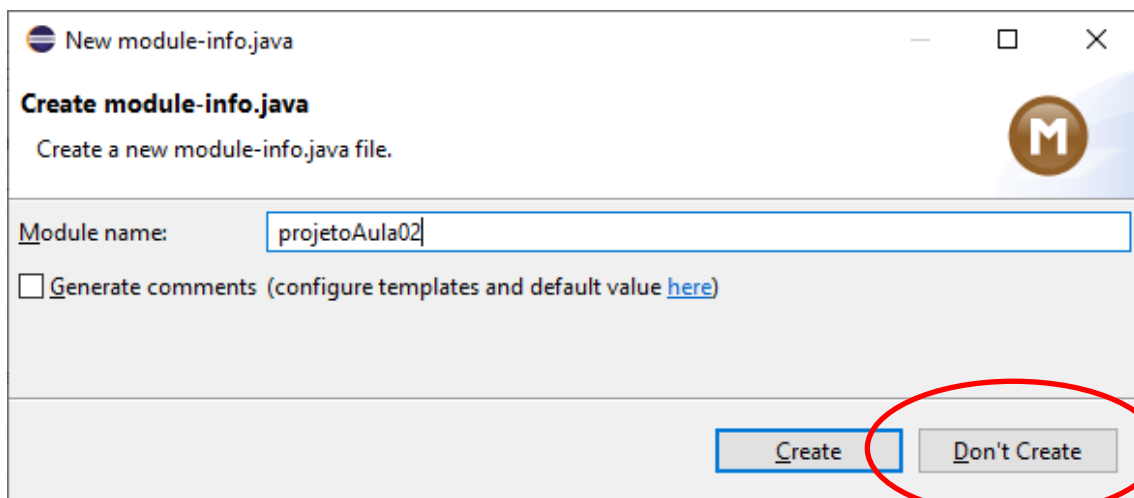


## Abrindo o eclipse:



## Criando um projeto: **File / New / Java Project**



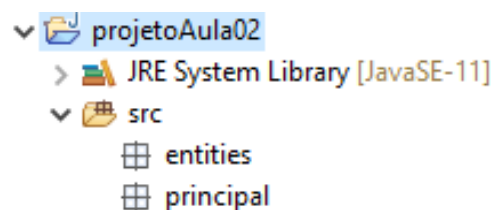
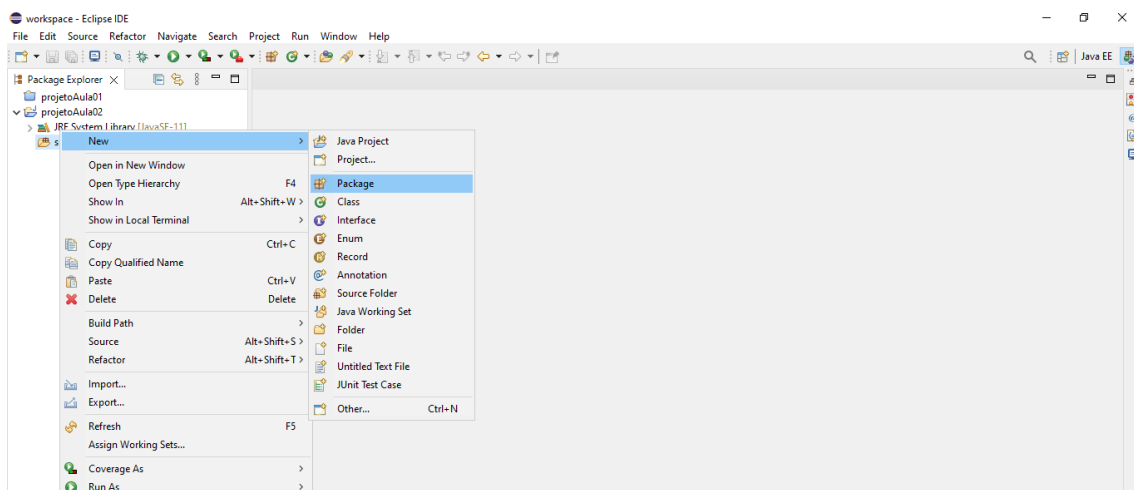


## packages

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

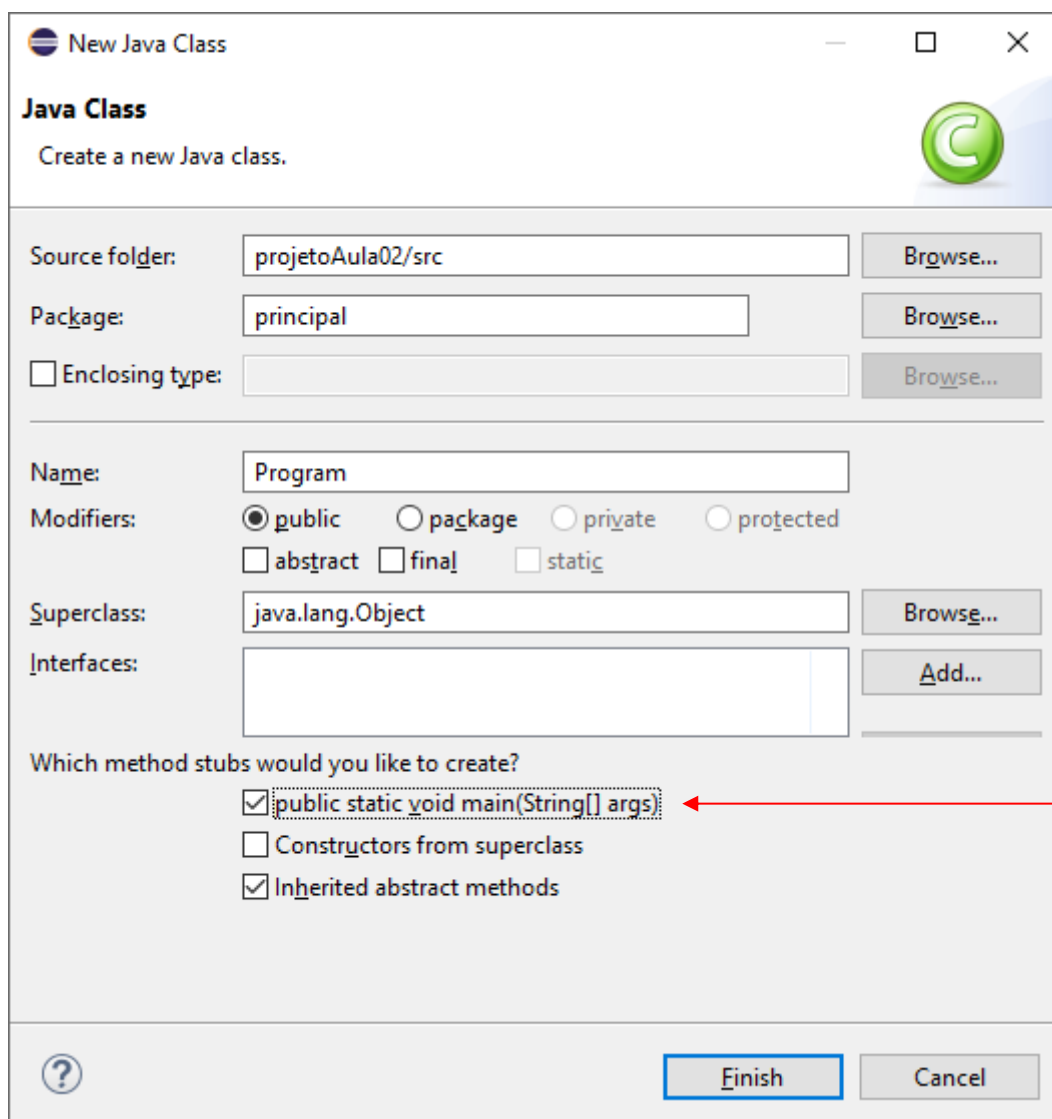
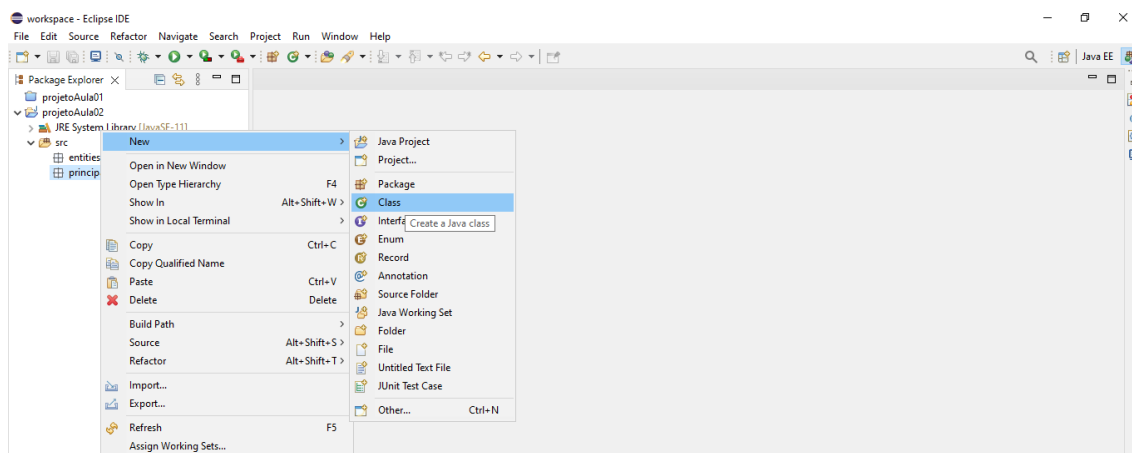
Exemplo:

- **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)



/principal/**Program.java**

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



```
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

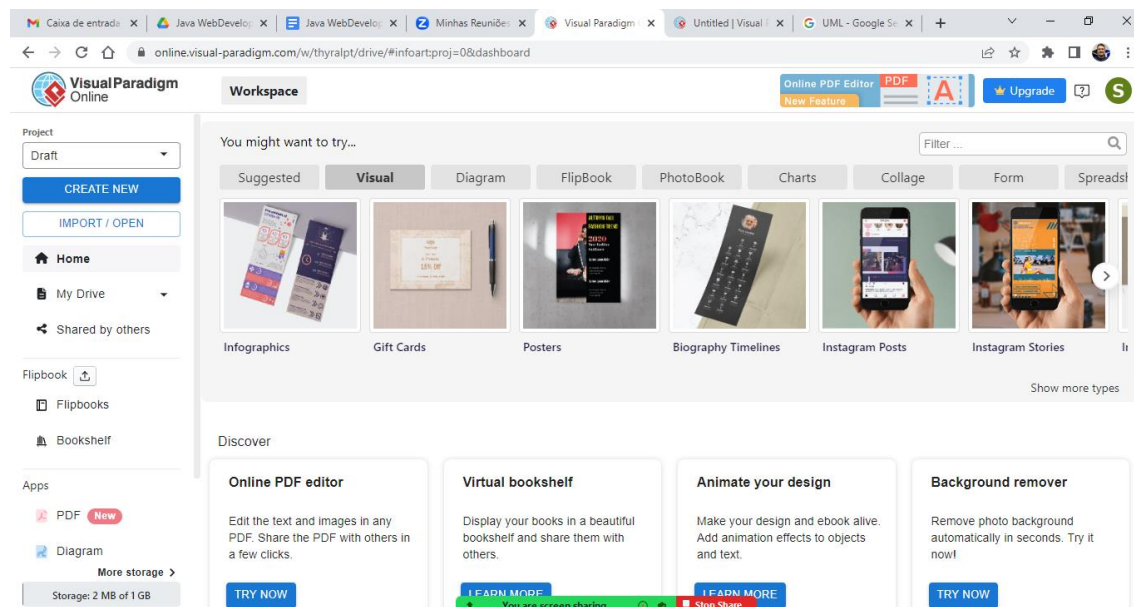
## UML – Unified Modeling Language

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

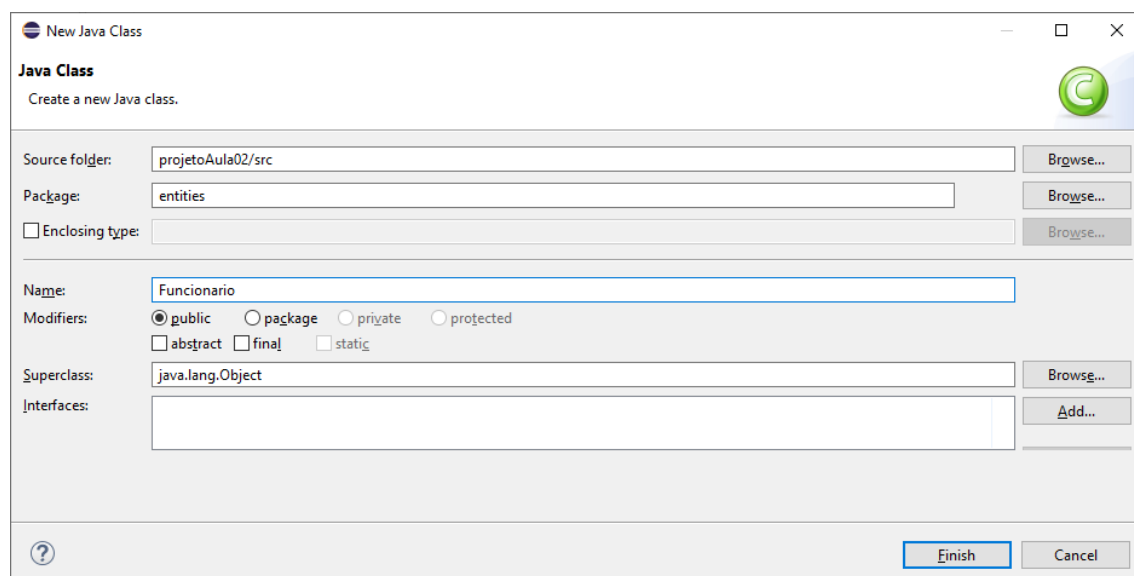
<https://online.visual-paradigm.com/>



Criando uma classe de entidade para Funcionario:

/entities

Funcionario
- idFuncionario : Integer
- nome : String
- matricula : String
- cpf : String



## Declarando os atributos da classe:

Atributos da classe possuem visibilidade "private";

```
package entities;
```

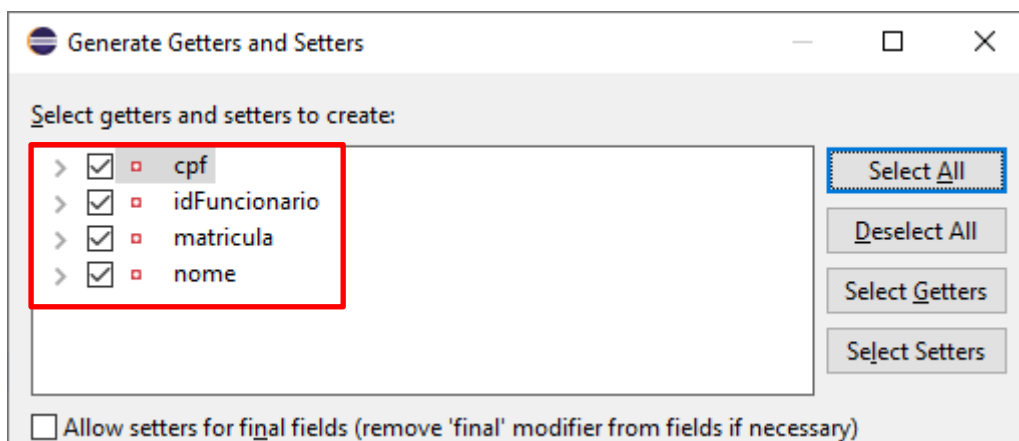
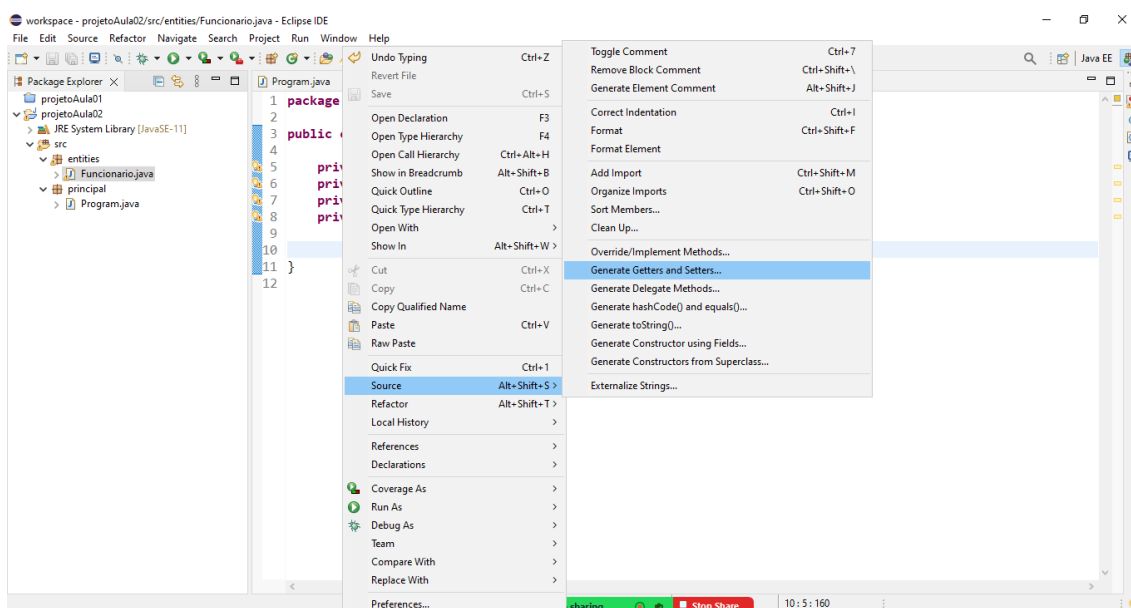
```
public class Funcionario {

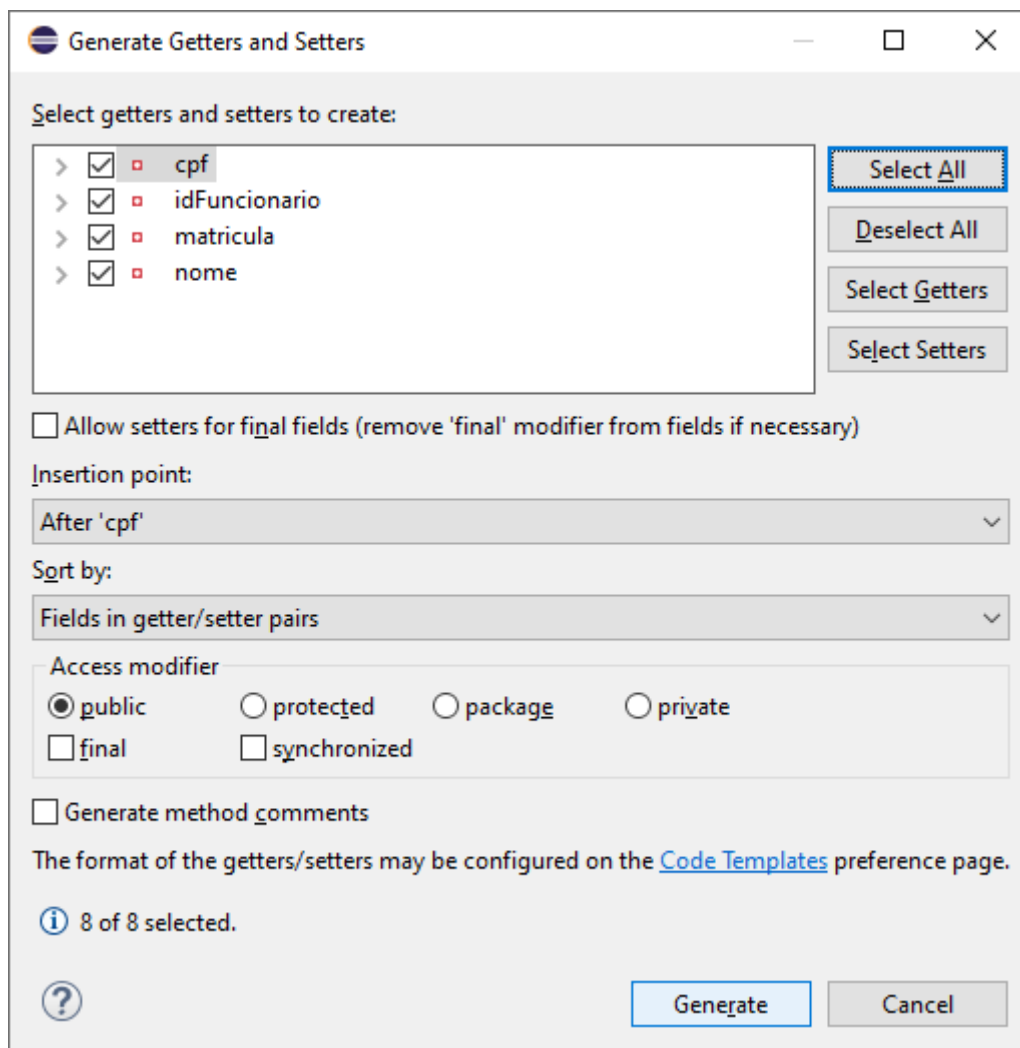
    private Integer idFuncionario;
    private String nome;
    private String matricula;
    private String cpf;
}
```

## 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





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** entidades;

**public class** Funcionario {

**private** Integer idFuncionario;

**private** String nome;

**private** String matricula;

**private** String cpf;





## 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());

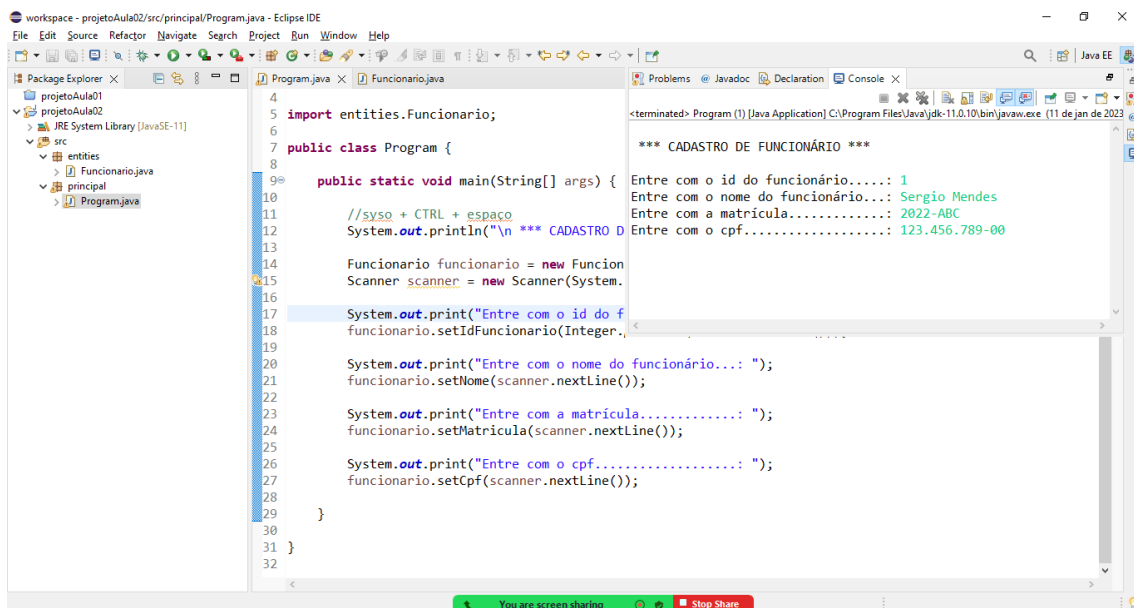
        System.out.print("Entre com o cpf.....: ");
        funcionario.setCpf(scanner.nextLine());

    }
}
```

### 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
```



```

workspace - projetoAula02/src/principal/Program.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer X
  projetoAula01
  projetoAula02
    JRE System Library [JavaSE-11]
    src
      Funcionario.java
      principal
        Program.java
  Problems X
  Javadoc X
  Declaration X
  Console X
  <terminated> Program (1) [Java Application] C:\Program Files\Java\jdk-11.0.10\bin\javaw.exe (11 de jan de 2023)
    *** 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

4  import entities.Funcionario;
5
6
7  public class Program {
8
9      public static void main(String[] args) {
10
11          //syso + CTRL + espaço
12          System.out.println("\n *** CADASTRO D
13
14          Funcionario funcionario = new Funcion
15          Scanner scanner = new Scanner(System.
16
17          System.out.print("Entre com o id do f
18          funcionario.setIdFuncionario(Integer.
19
20          System.out.print("Entre com o nome do funcionário... ");
21          funcionario.setNome(scanner.nextLine());
22
23          System.out.print("Entre com a matrícula..... ");
24          funcionario.setMatricula(scanner.nextLine());
25
26          System.out.print("Entre com o cpf..... ");
27          funcionario.setCpf(scanner.nextLine());
28
29      }
30
31  }
32
  
```

```
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());

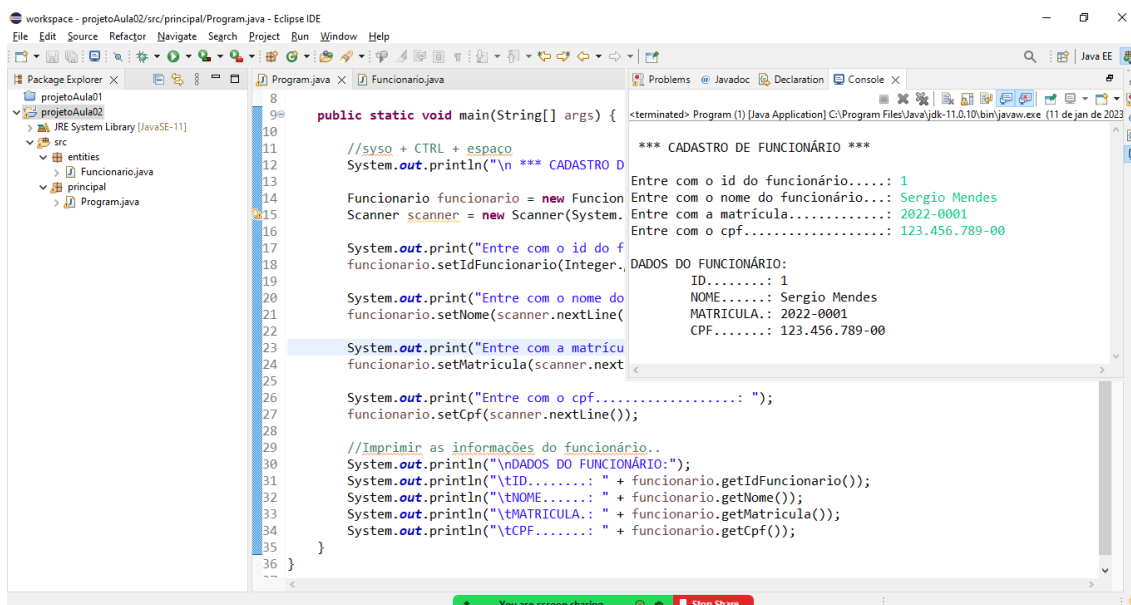
        System.out.println("\tNOME.....: "
            + funcionario.getNome());

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

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

```

## Executando:



## 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:

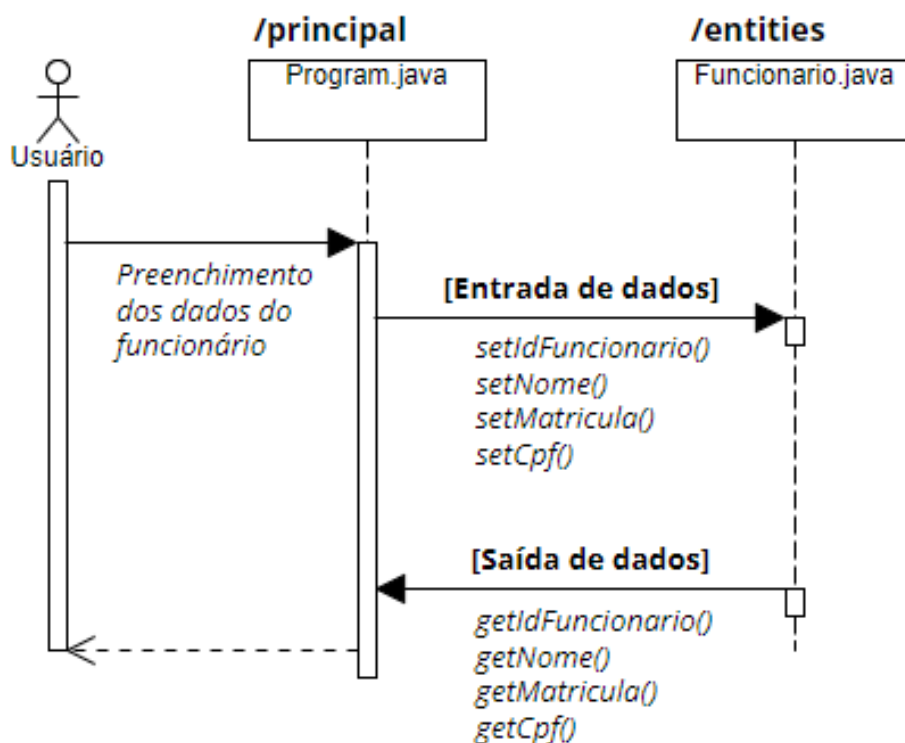
```

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:



## 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

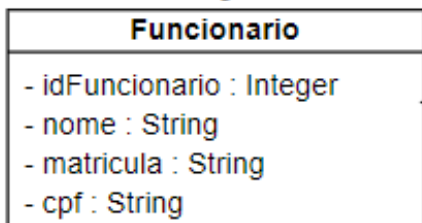
Na programação, o **Princípio da responsabilidade única** declara que cada módulo ou classe deve ter responsabilidade sobre uma única parte da funcionalidade fornecida pelo software.

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

## /entities (JavaBean)

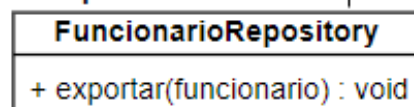


### Responsabilidade:

Modelar e armazenar os dados de um funcionário.

Utilizar

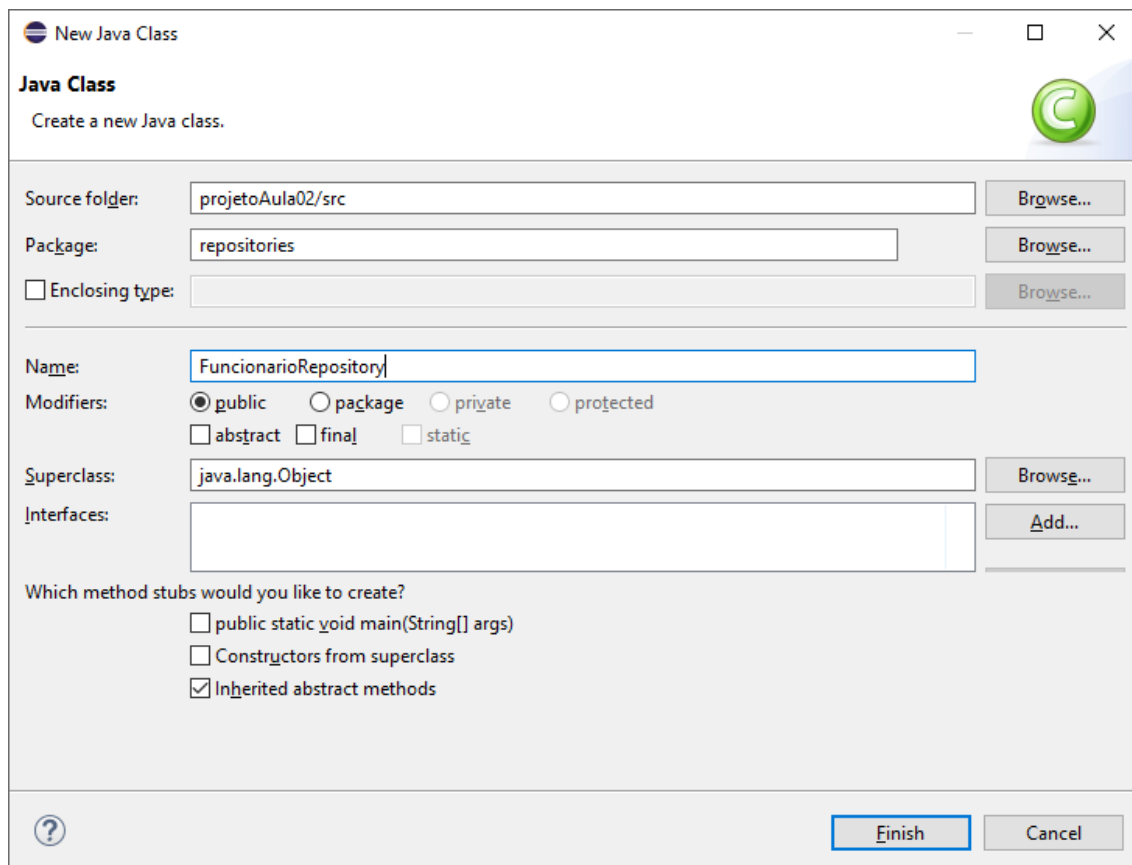
## /repositories



### Responsabilidade:

Gravar, ler, alterar ou excluir os dados do funcionário em arquivo.

Arquivo  
TXT



**New Java Class**

Java Class  
Create a new Java class.

Source folder: projetoAula02/src Browse...

Package: repositories Browse...

☐ Enclosing type: Browse...

Name: FuncionarioRepository

Modifiers: ☒ public ☐ package ☐ private ☐ protected  
☐ abstract ☐ final ☐ static

Superclass: java.lang.Object Browse...

Interfaces: Add...

Which method stubs would you like to create?

☐ public static void main(String[] args)  
☐ Constructors from superclass  
☒ Inherited abstract methods

? Finish Cancel

```
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 {

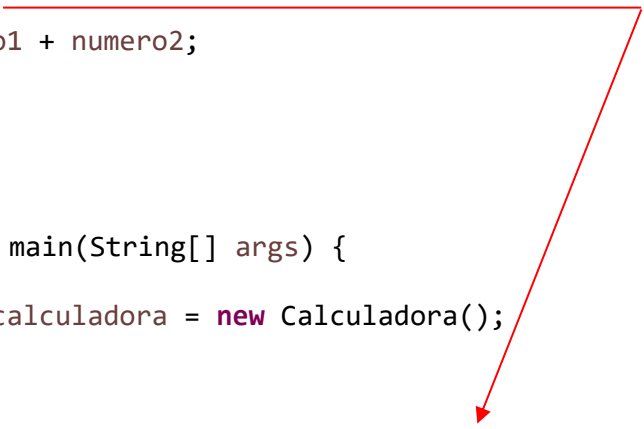
    public static void main(String[] args) {

        Calculadora calculadora = new Calculadora();

        try {

            Double resultado = calculadora.somar(10.0, 20.0);
            System.out.println("A soma é: " + resultado);

        }
    }
}
```

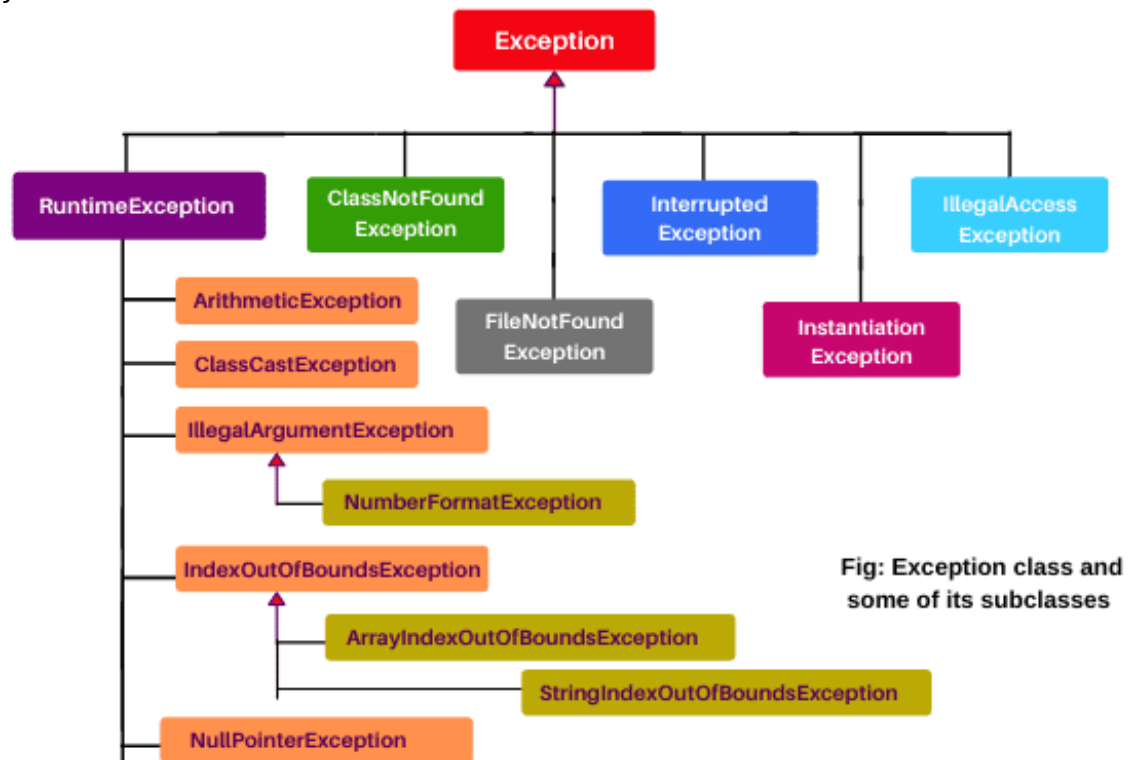


```

    catch(Exception e) {

        System.out.println("Ocorreu um erro!");
        e.printStackTrace();
    }
}

```



```

package repositories;

import java.io.PrintWriter;
import entidades.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 conteúdo do arquivo
        printWriter.write("\nDADOS DO FUNCIONÁRIO:");

        printWriter.write
            ("\nID.....: " + funcionario.getIdFuncionario());
    }
}

```



```

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

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

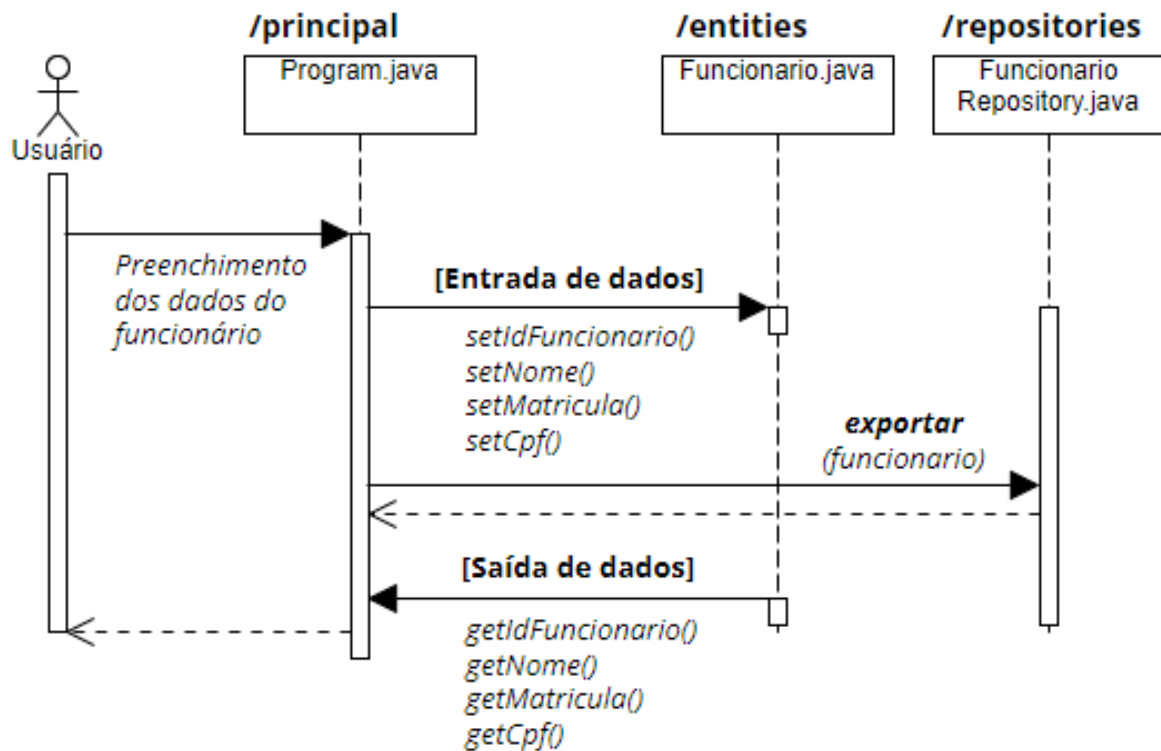
        printWriter.write
        ("\nCPF.....: " + funcionario.getCpf());

        // salvar o arquivo
        printWriter.flush();

        // fechar o arquivo
        printWriter.close();
    }
}

```

Voltando na classe **Program.java**



```

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);

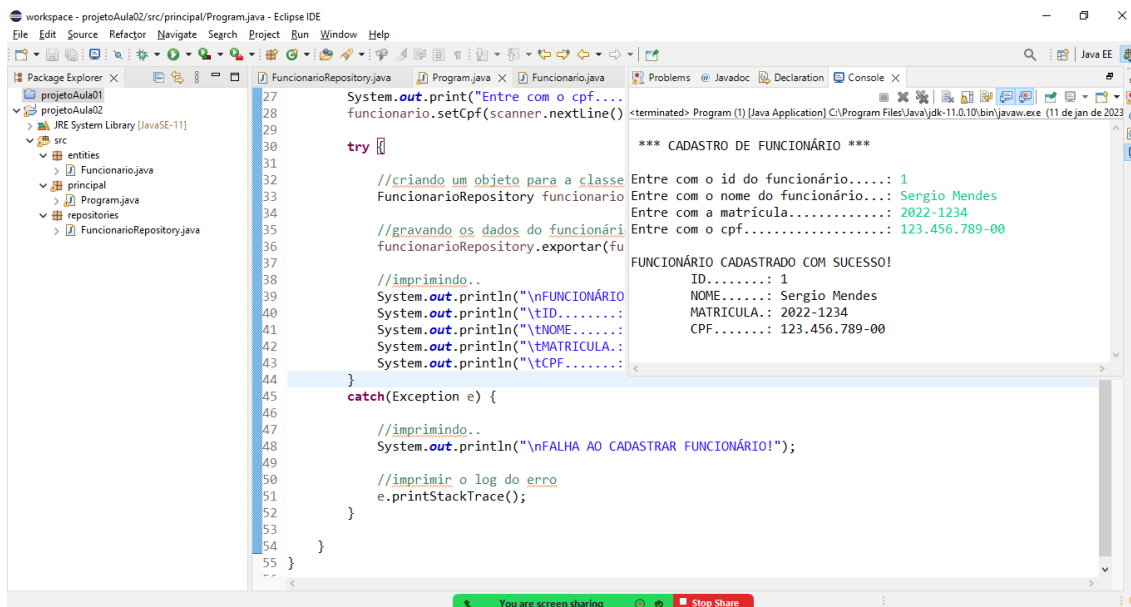
        //imprimindo..
        System.out.println
            ("\nFUNCIONÁRIO CADASTRADO COM SUCESSO!");

        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:



```

workspace - projetoAula02/src/principal/Program.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer X
projetoAula02
  JRE System Library [JavaSE-11]
  src
    entidades
      Funcionario.java
    principal
      Program.java
    repositories
      FuncionarioRepository.java
27 System.out.print("Entre com o cpf...");
28 funcionario.setCpf(scanner.nextLine());
29
30 try {
31
32     //criando um objeto para a classe
33     FuncionarioRepository funcionario;
34
35     //gravando os dados do funcionário
36     funcionarioRepository.exportar(fu
37
38     //imprimindo..
39     System.out.println("\nFUNCIONÁRIO
40     System.out.println("\tID.....:
41     System.out.println("\tNOME.....:
42     System.out.println("\tMATRICULA.:
43     System.out.println("\tCPF.....:
44 }
45 catch(Exception e) {
46
47     //imprimindo..
48     System.out.println("\nFALHA AO CADASTRAR FUNCIONÁRIO!");
49
50     //imprimir o log do erro
51     e.printStackTrace();
52 }
53
54 }
55 }
*** 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
  
```

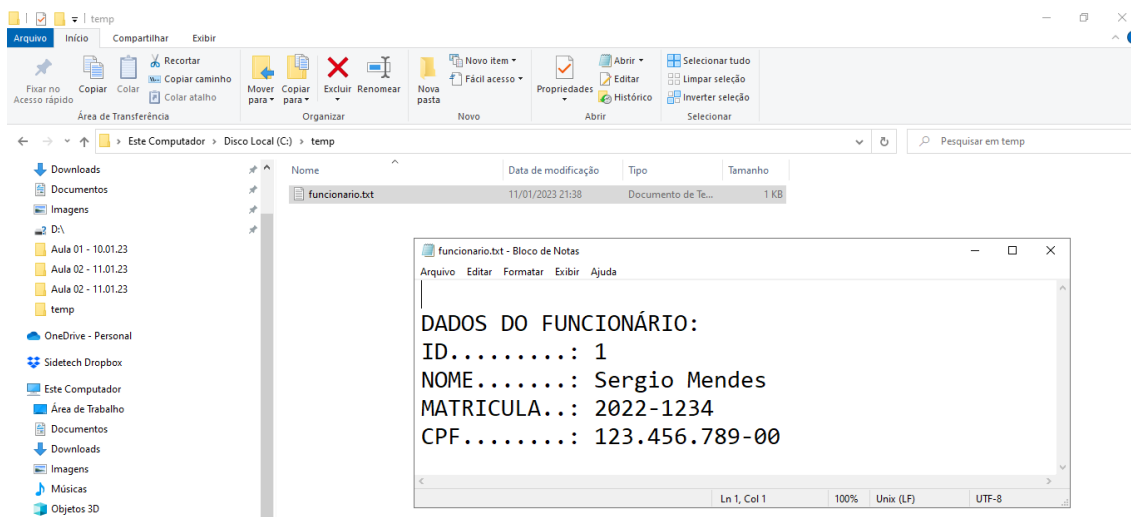
\*\*\* CADASTRO DE FUNCIONÁRIO \*\*\*

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:



## 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:

```
class Test {  
  
    public static void main(String[] args) {  
  
        Impressao impressao = new Impressao();  
  
        impressao.imprimir()  
    }  
}
```

- imprimir() : void - Impressao
- imprimir(String nome) : void - Impressao
- imprimir(String nome, String sobrenome) : void - Impressao

```
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 conteúdo do arquivo
        printWriter.write("\nDADOS DO FUNCIONÁRIO:");

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

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

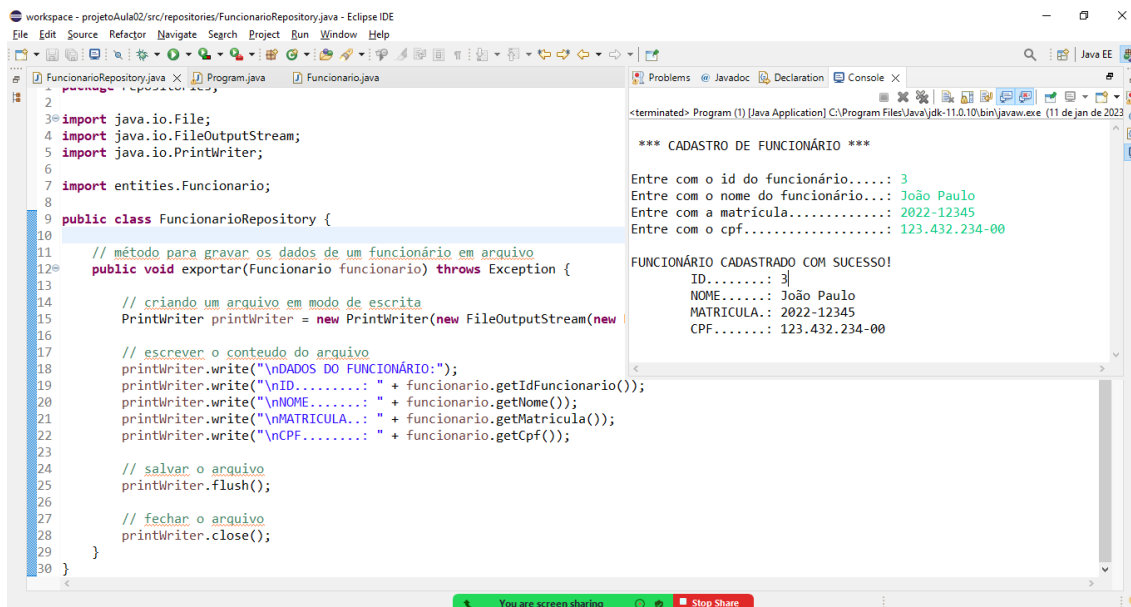
        printWriter.write("\nMATRICULA..: "
            + funcionario.getMatricula());

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

        // salvar o arquivo
        printWriter.flush();

        // fechar o arquivo
        printWriter.close();
    }
}
```

## Executando:



```

workspace - projetoAula02/src/repositories/FuncionarioRepository.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help

FuncionarioRepository.java
1
2
3 import java.io.File;
4 import java.io.FileOutputStream;
5 import java.io.PrintWriter;
6
7 import entities.Funcionario;
8
9 public class FuncionarioRepository {
10
11 // método para gravar os dados de um funcionário em arquivo
12 public void exportar(Funcionario funcionario) throws Exception {
13
14 // criando um arquivo em modo de escrita
15 PrintWriter printWriter = new PrintWriter(new FileOutputStream(new
16
17 // escrever o conteúdo do arquivo
18 printWriter.write("\nDADOS DO FUNCIONÁRIO:");
19 printWriter.write("\nID.....: " + funcionario.getIdFuncionario());
20 printWriter.write("\nNOME.....: " + funcionario.getNome());
21 printWriter.write("\nMATRICULA..: " + funcionario.getMatricula());
22 printWriter.write("\nCPF.....: " + funcionario.getCpf());
23
24 // salvar o arquivo
25 printWriter.flush();
26
27 // fechar o arquivo
28 printWriter.close();
29
30 }
}

*** 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
  
```

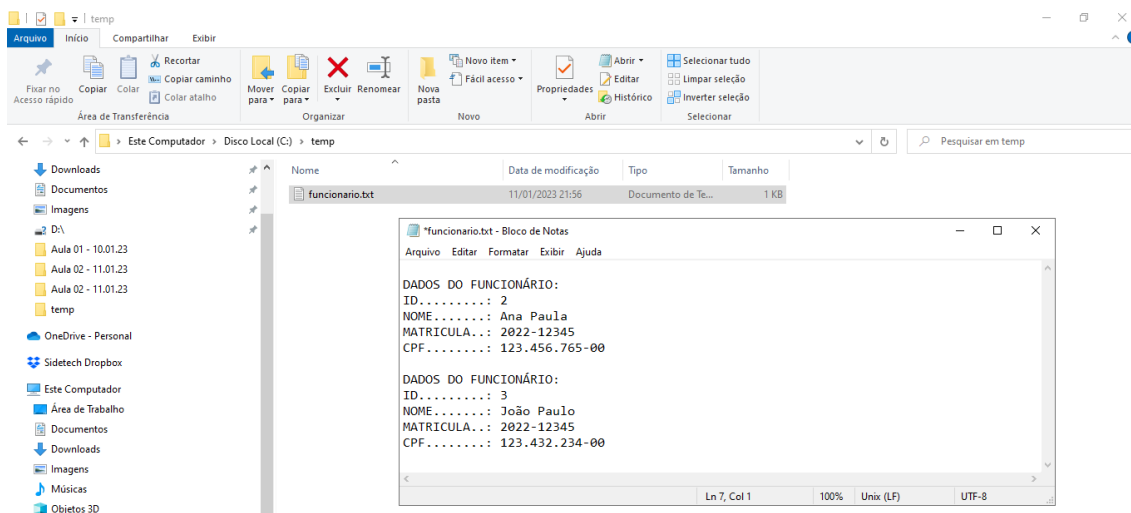
\*\*\* CADASTRO DE FUNCIONÁRIO \*\*\*

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

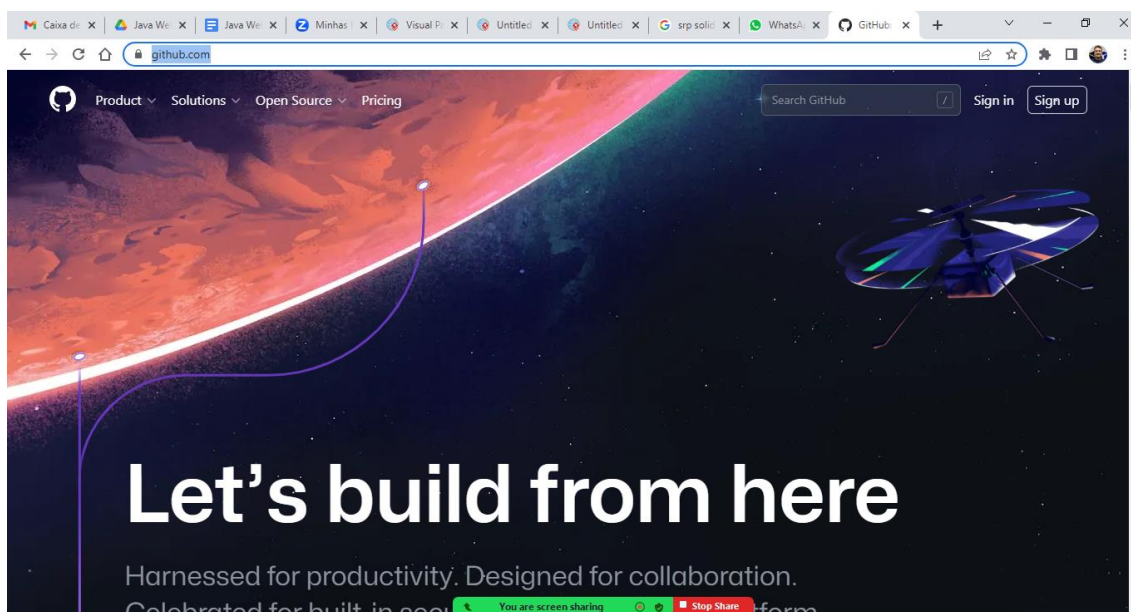
## Arquivo gerado com sucesso:



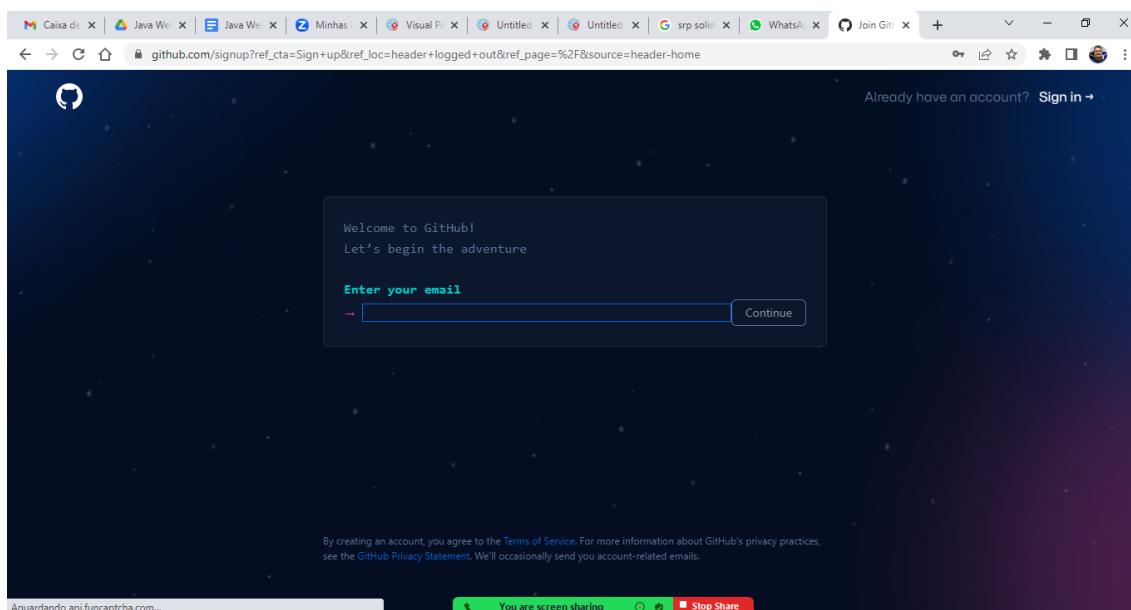
## GIT HUB

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/>



[https://github.com/signup?ref\\_cta=Sign+up&ref\\_loc=header+logged+out&ref\\_page=%2F&source=header-home](https://github.com/signup?ref_cta=Sign+up&ref_loc=header+logged+out&ref_page=%2F&source=header-home)



## 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.

