

BACK-END

Prof. Bruno Kurzawe



Programação Orientada a Objetos

Programação orientada a objetos (POO, ou OOP segundo as suas siglas em inglês) é um paradigma de **programação** baseado no conceito de "**objetos**", que podem conter dados na forma de campos, também conhecidos como atributos, e códigos, na forma de procedimentos, também conhecidos como métodos.



A Programação Orientada a Objetos (POO) visa trazer um modo de estruturação e organização do código mais próximo à maneira como percebemos o mundo real.









Como eu poderia representar os três objetos de forma generalizada?

Todos são móveis? Correto?



MÓVEL

- Altura
- Peso
- Quantidade de pés
- Largura
- Profundidade
- Material de Fabricação
- Preço de Fabricação
- Preço de Venda

Poderia generalizar todos esse itens como MÓVEL?



Na POO esses objetos serão representados em CLASSES

Em programação orientada a objetos, uma classe é uma estrutura que serve como um plano ou modelo para criar objetos. A classe define o estado e o comportamento que os objetos da classe possuirão.

Estado: É representado por variáveis de instância, também conhecidas como atributos ou propriedades. Por exemplo, em uma classe "Movél", os estados podem incluir coisas como altura, largura, profundidade e material.

Comportamento: É representado por métodos (funções). Estes definem as ações que um objeto da classe pode realizar. Na classe "Móvel", por exemplo, os comportamentos podem incluir métodos como montar e desmontar.

Classe	Estado	Comportamento
Pessoa	Nome, Idade, RG	Falar, Andar, Comer
Cachorro	Nome, Raça	Latir, Correr
Conta Bancária	Saldo, Agência, Número	Creditar, Debitar
Carro	Cor, Marca, Modelo	Acelerar, Frear, Abastecer

CLASSE**OBJETO****OBJETO**

Documento	Documento 1	Documento 2
Foto:	Img.png	Img4.png
Código:	123456	236768
Nome:	Alfredo	Juliana
Data Nascimento:	20/05/1960	20/05/1987

Quatro pilares da Programação Orientada a Objetos



P.O.O.

Programação Orientada a Objetos

Classes e Objetos

Abstração

Herança

Polimorfismo

Encapsulamento

Abstração é um conceito fundamental na ciência da computação e na programação, que descreve a capacidade de simplificar e representar entidades complexas, ideias ou objetos do mundo real de forma mais simples e abstraída, a fim de torná-los mais compreensíveis e gerenciáveis para os seres humanos e os sistemas de computação.

Herança é um conceito fundamental na programação orientada a objetos (POO) que permite que uma classe (chamada de classe derivada ou subclasse) herde os atributos e métodos de outra classe (chamada de classe base ou superclasse). Com a herança, a classe derivada pode reutilizar e estender as funcionalidades da classe base, evitando a duplicação de código e permitindo uma melhor organização e estruturação do código.

Polimorfismo é um conceito importante na programação orientada a objetos (POO) que permite que objetos de diferentes classes sejam tratados de forma uniforme, mesmo que essas classes possuam comportamentos diferentes. É uma das características fundamentais da POO que visa aprimorar a flexibilidade e extensibilidade do código.

Encapsulamento é um dos princípios fundamentais da programação orientada a objetos (POO) e envolve o conceito de ocultar a implementação interna de uma classe e fornecer uma interface externa consistente e controlada para interagir com essa classe. Em outras palavras, o encapsulamento permite que os detalhes internos e a estrutura de uma classe sejam privados e acessíveis somente dentro da própria classe, enquanto os usuários externos da classe interagem com ela apenas através de métodos públicos definidos.

Qual a importância da programação orientada a objetos?

Confiável

Oportuno

Ajustável

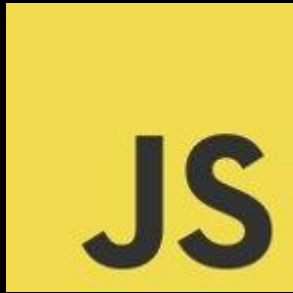
Extensível

Reutilizável

Natural



Exemplos de linguagem que poderiam ser utilizadas para POO:



Qual linguagem de programação que usaremos nessa disciplina?



Características da linguagem Java

Simples: O aprendizado da linguagem de programação Java pode ser feito em um curto período de tempo;

Orientada a objetos: Desde o início do seu desenvolvimento esta linguagem foi projetada para ser orientada a objetos;

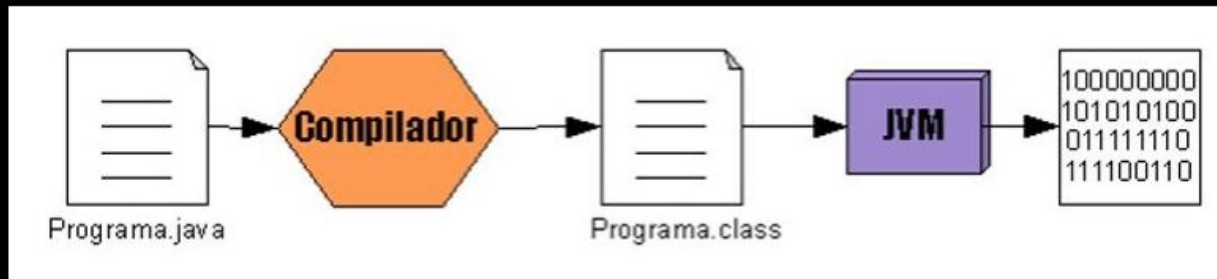
Familiar: A linguagem Java é muito familiar para os programadores C/C++ ;

Robusta: Ela foi pensada para o desenvolvimento de softwares confiáveis, provendo verificações tanto em tempo de execução quanto compilação, o coletor de lixo responsabiliza-se pela limpeza da memória quando houver necessidade;

Segura: Aplicações Java são executadas em ambiente próprio (JRE) o que inviabiliza a intrusão de código malicioso;

Portável: Programas desenvolvidos nesta linguagem podem ser executados em praticamente qualquer máquina desde que esta possua o JRE instalado;

Máquina Virtual JAVA - JVM



Criação do código fonte (Programa.java);

Compilação do código fonte e geração do bytecode (Programa.class);

Interpretação do bytecode pela máquina virtual;

Conversão do bytecode em linguagem de máquina.

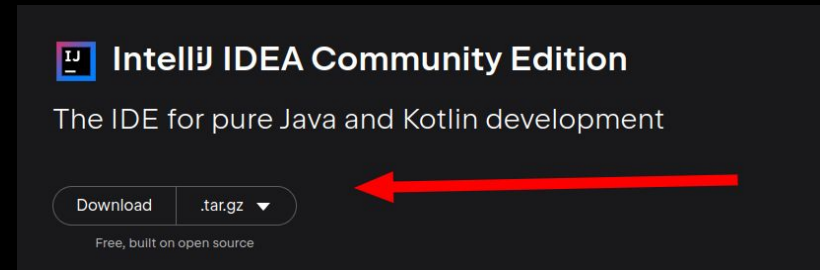
JRE: O Java Runtime Environment contém tudo aquilo que um usuário comum precisa para executar uma aplicação Java (JVM e bibliotecas), como o próprio nome diz é o “Ambiente de execução Java”;

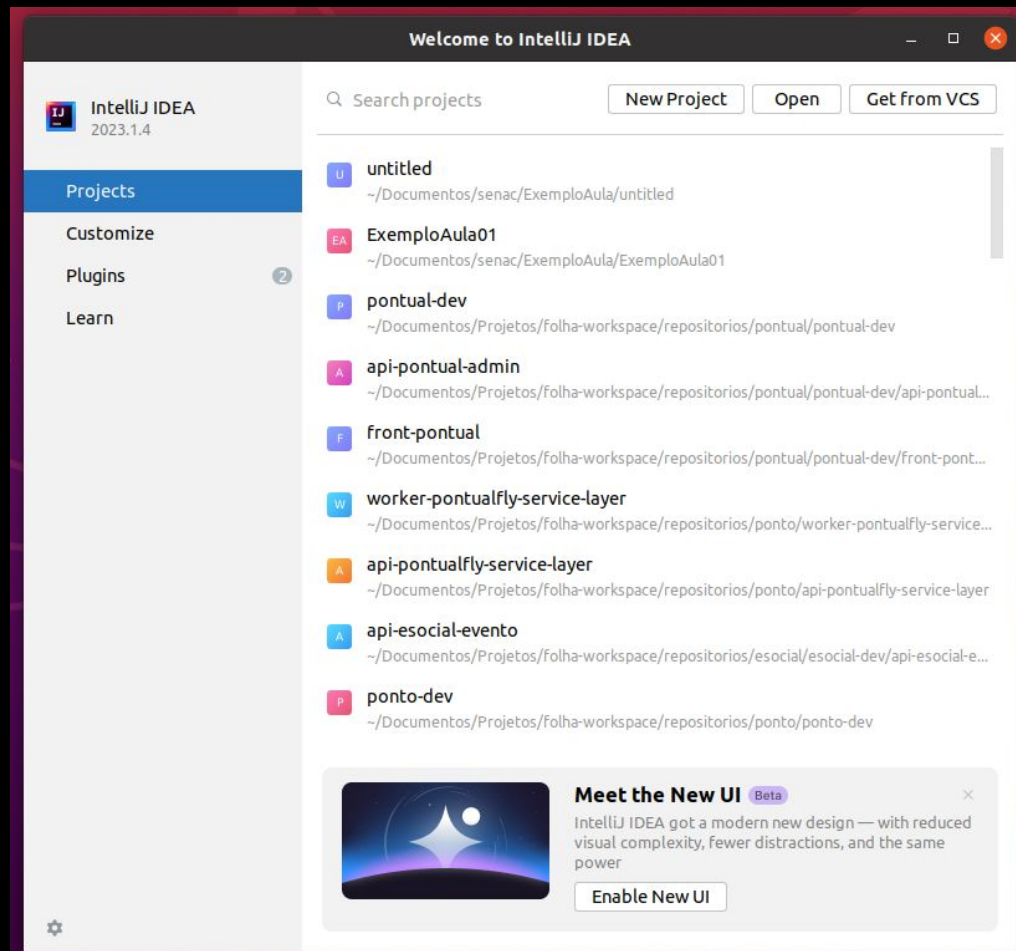
JDK: O Java Development Kit é composto pelo JRE e um conjunto de ferramentas úteis ao desenvolvedor Java.



Como ferramenta de desenvolvimento usaremos o IntelliJ

<https://www.jetbrains.com/pt-br/idea/download/#section=windows>






New Project

to create a general Maven project, go to the [New Project](#) page

Generators


- New Project
- Empty Project
- Maven Archetype**
- JavaFX
- Kotlin Multiplatform
- Compose Multiplatform
- IDE Plugin
- Android


Name:


Location: 

Project will be created in: ~/Documentos...c/ExemploAula/ExemploAula

☐ Create Git repository

JDK:  **corretto-17** Amazon Corretto version 17. ▾

Catalog:  [Manage catalogs...](#)

Archetype: 

Version:

Additional Properties

+ -

No properties

> **Advanced Settings**

New Project

New Project

Empty Project

Generators

Maven Archetype

JavaFX

Kotlin Multiplatform


Compose Multiplatform

IDE Plugin

Android


To create a general Maven project, go to the [New Project](#) page.


Name:


Location: 

Project will be created in: ~/Documentos/senac/ExemploAula/Exemplo0001

☐ Create Git repository

JDK: 

Catalog:  [Manage catalogs...](#)

Archetype: 


Version:

Additional Properties

+ -

No properties

> Advanced Settings



com.dominikcebula.archetypes:java17-basic-archetype

com.dominikcebula.archetypes:java17-cli-pico-archetype


com.dominikcebula.archetypes:java17-spring-boot3-cli-archetype

com.dominikcebula.archetypes:java17-spring-boot3-rest-archetype


New Project



To create a general Maven project, go to the [New Project](#) page.



Name:


Location: 
Project will be created in: ~/Documentos...c/ExemploAula/ExemploAula001

☐ Create Git repository

JDK: 

Catalog:   [Manage catalogs...](#)

Archetype:  


Version: 


Additional Properties

+ -

No properties

> Advanced Settings





Exemplo0001 - pom.xml (Exemplo0001)

File Edit View Navigate Code Refactor Build Run Tools VCS Window Help

Exemplo0001 pom.xml

Project

- Exemplo0001 - ~/Documentos/senac/Exemplo0001
 - .idea
 - src
 - .gitignore
 - Exemplo0001 pom.xml
 - External Libraries
 - Scratches and Consoles

Run pom.xml (Exemplo0001)

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
4     <modelVersion>4.0.0</modelVersion>
5
6     <groupId>org.example</groupId>
7     <artifactId>Exemplo0001</artifactId>
8     <version>1.0-SNAPSHOT</version>
9
10    <properties>
11        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
12    </properties>
```

Property
project.build.sourceEncoding: UTF-8
Exemplo0001

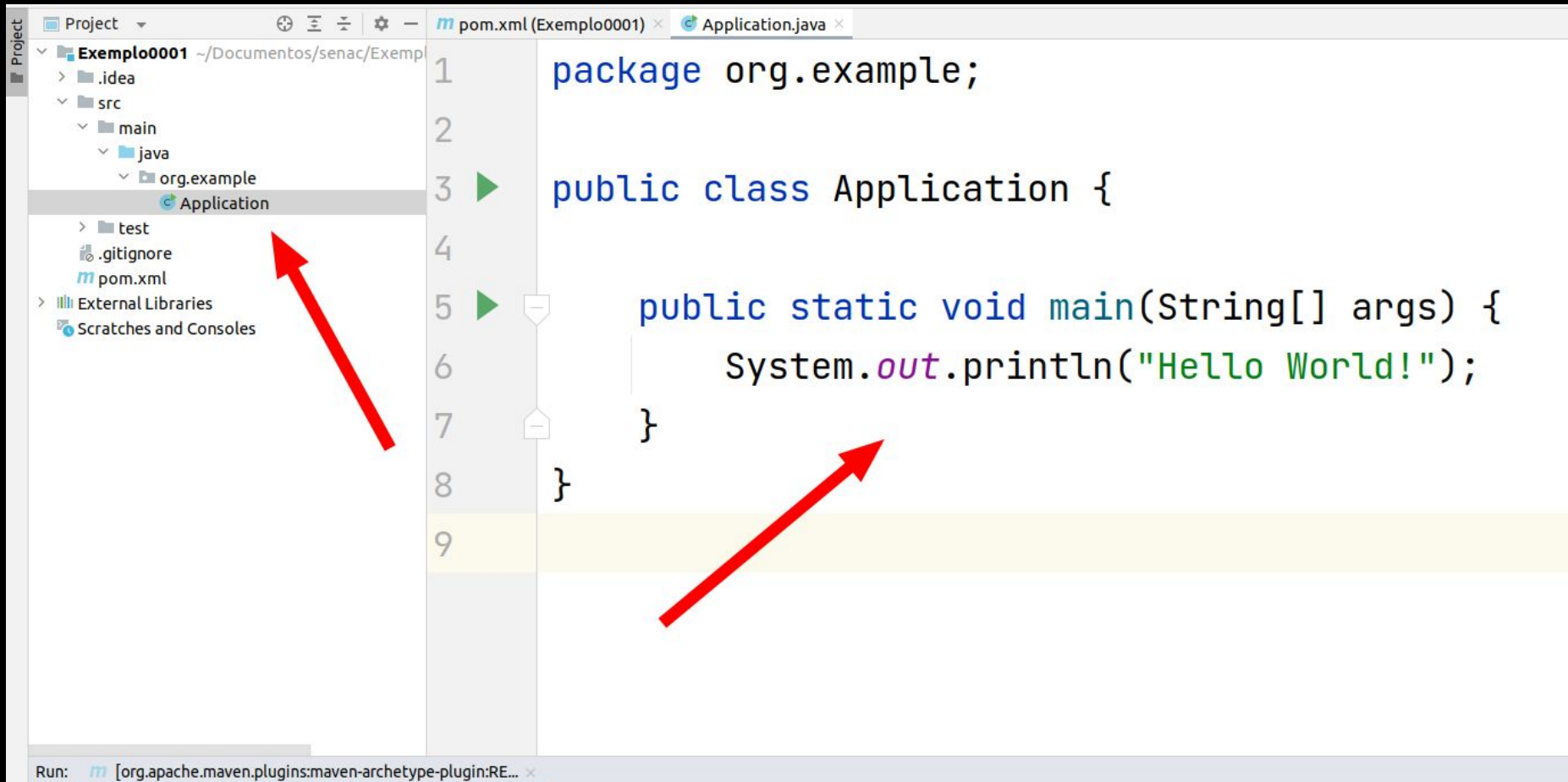
Run: [org.apache.maven.plugins:maven-archetype-plugin:RE...

[INFO] BUILD SUCCESS

[INFO]

Version Control Run TODO Problems Terminal Services Build Dependencies

1:1 LF UTF-8 4spaces



Este projeto já vem pré configurado com MAVEN, mas para frente abordaremos essa tema.

O **main** é o método que inicia as aplicações Java, quando solicitamos ao interpretador que execute uma determinada classe compilada ele procura o método main, se este método não existir irá ser gerada uma exceção informando que o método não foi localizado.



Palavras reservadas

abstract	boolean	break	byte	case	catch
char	class	const	continue	default	do
double	else	extends	final	finally	float
for	goto	if	implements	import	instanceof
int	interface	long	native	new	package
private	protected	public	return	short	static
strictfp	super	switch	synchronized	this	throw
throws	transient	try	void	volatile	while
assert					

Agora vamos desenvolver um sistema orientado a objetos juntos?

Para desenvolver um sistema, primeiro precisamos de um tema!

Loja de venda e locação de equipamentos informática

Que funcionalidades poderíamos ter no nosso sistema?

Gestão de Produtos: O sistema deve ser capaz de adicionar, editar, excluir e visualizar equipamentos de informática. Cada equipamento tem atributos como código, nome, descrição, preço de venda, preço de locação, quantidade em estoque, e status (disponível, alugado, vendido).

Gestão de Clientes: O sistema deve ser capaz de adicionar, editar, excluir e visualizar clientes. Cada cliente tem atributos como ID, nome, endereço, telefone e email.

Vendas: O sistema deve ser capaz de registrar a venda de equipamentos para clientes, atualizar a quantidade de estoque de equipamentos e gerar uma fatura de venda.

Locações: O sistema deve ser capaz de registrar a locação de equipamentos para clientes, marcar o equipamento como alugado, definir uma data de devolução e gerar uma fatura de locação.

Devoluções: O sistema deve ser capaz de registrar a devolução de equipamentos alugados, marcar o equipamento como disponível novamente e calcular quaisquer taxas de atraso devidas.

Gestão de Fornecedores: O sistema deve ser capaz de adicionar, editar, excluir e visualizar fornecedores. Cada fornecedor tem atributos como ID, nome, endereço, telefone, email e produtos fornecidos.

Compras: O sistema deve ser capaz de registrar a compra de equipamentos de fornecedores, adicionar os equipamentos adquiridos ao estoque e gerar uma ordem de compra.

Gestão de Estoque: O sistema deve rastrear a quantidade de cada equipamento em estoque. Deve atualizar a quantidade em estoque quando equipamentos são comprados de um fornecedor, vendidos a um cliente ou alugados. Deve também alertar quando a quantidade em estoque de um equipamento cair abaixo de um certo limite.

Agora que temos requisitos, vamos iniciar o desenvolvimento.

Começaremos representando criando nossas classes de modelo, que de fato começaram a representar os objetos do nosso sistema.

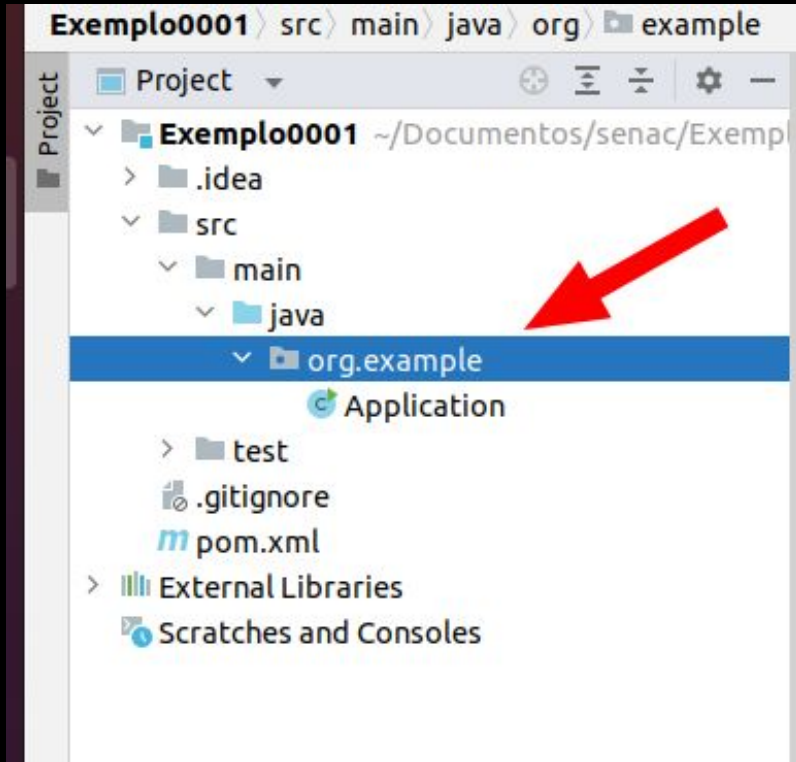
Que classes poderíamos ter no nosso projeto?

- Produto
- Cliente
- Venda
- Locação
- Fornecedor
- Compra
- Estoque

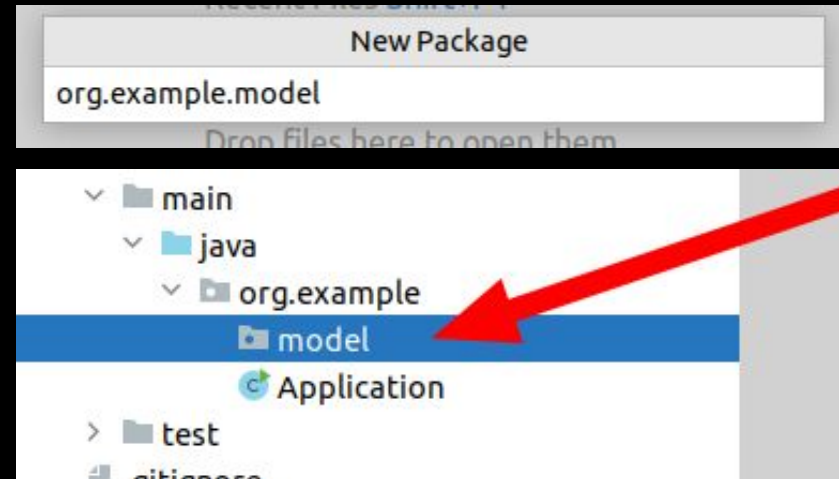
Quais atributos
poderíamos ter na
nossas classes?

Vamos abrir nossa IDE e começar de fato a desenvolver.

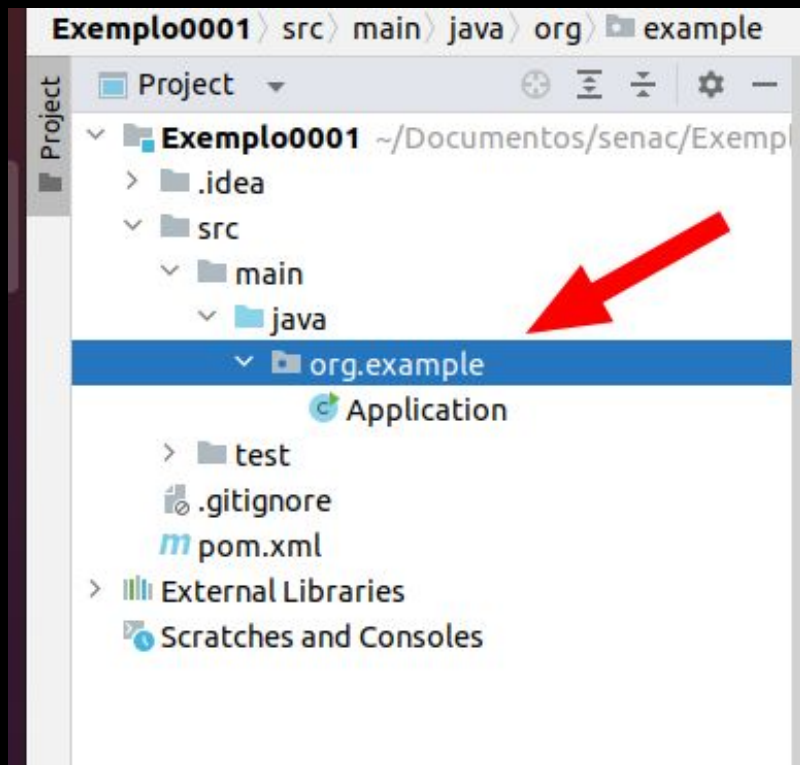
Criando pacotes



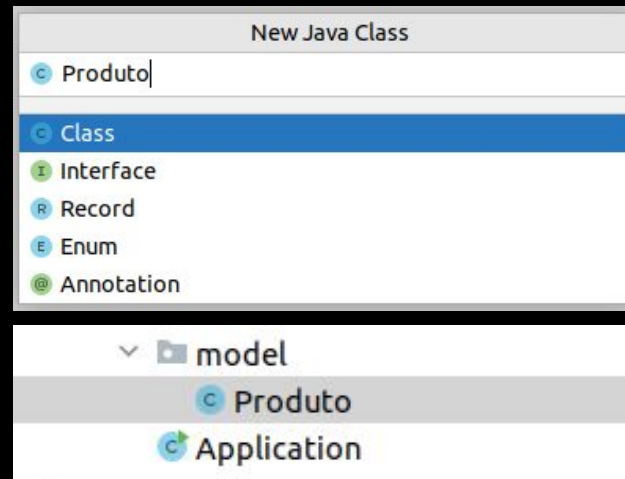
1. Clique com o botão direito
2. Clique em new
3. Package



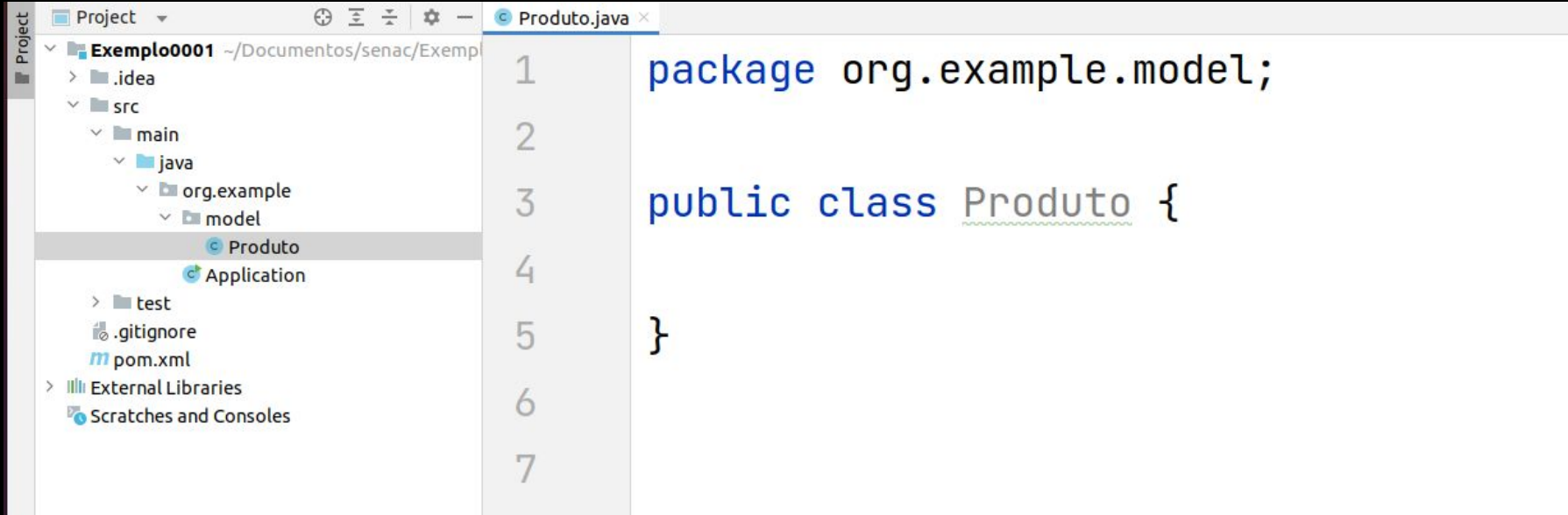
Criando uma classe



1. Clique com o botão direito
2. Clique em new
3. Java Class



Vamos criar nossa primeira classe Produto



The screenshot shows an IDE window with a project named 'Exemplo0001' located at '~/Documentos/senac/Exemplo'. The project structure is visible in the left sidebar, showing a hierarchy of folders: .idea, src, main, java, org.example, and model. The 'Produto' class is highlighted in the 'model' folder. The main editor window displays the code for 'Produto.java' with the following content:

```
1 package org.example.model;  
2  
3 public class Produto {  
4  
5 }  
6  
7
```

Vamos criar nossa primeira classe Produto

Precisaremos ter um campo ID

Precisaremos ter um nome

Precisaremos de uma descrição



Precisaremos de um preço de venda

Precisaremos de um preço de compra

Precisaremos de uma data de validade ou prazo

Precisaremos de um status (Disponível ou Alugado)

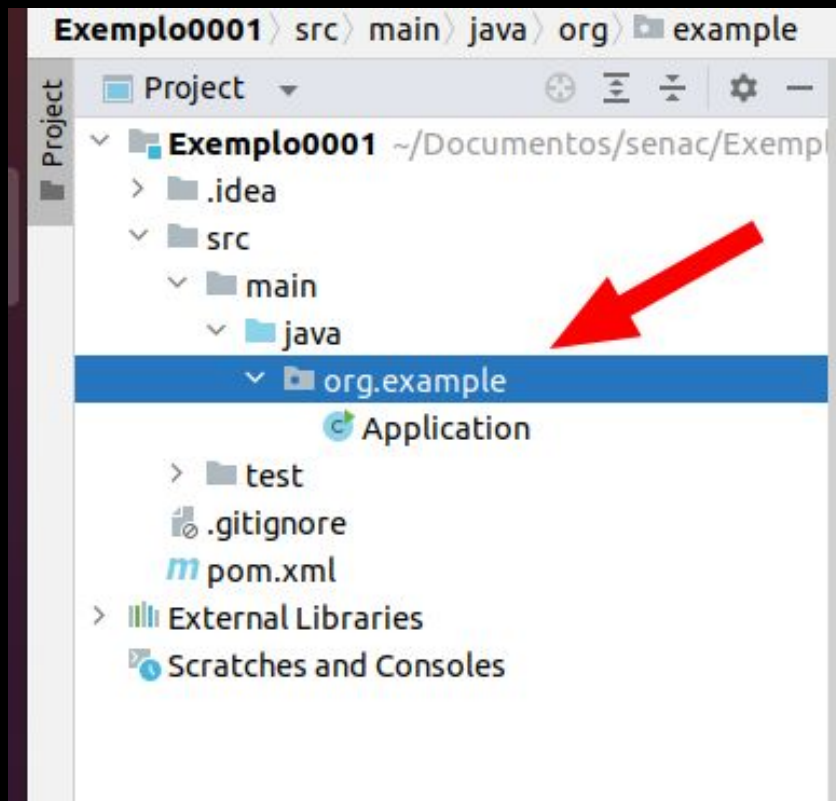
Declarando variáveis em JAVA

```
public class Produto {  
    Integer id;  NOME DA VARIÁVEL  
     TIPO DO DADO  
    String nome;  
  
    String descricao;  
    Double precoVenda;  
    Double precoCompra;  
    LocalDate dataValidade;  
    LocalDate dataPrazo;  
}
```

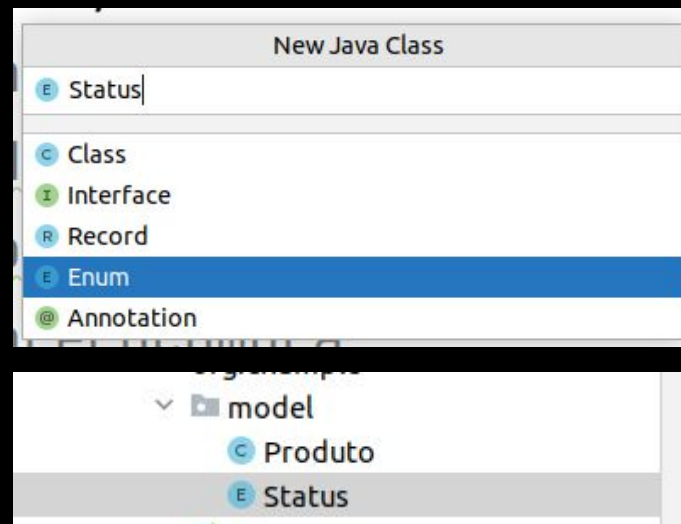
Definimos que nossa classe produto deveria ter um campo Status, podemos criar como String? Sim! É usual, não!

Vamos criar uma Enum!

Criando uma Enum



1. Clique com o botão direito
2. Clique em new
3. Java Class



Definindo dados de uma Enum

```
package org.example.model;  
  
public enum Status {  
    DISPONIVEL,  
    ALUGADO  
}
```


Usando uma Enum em uma classe

```
public class Produto {  
    Integer id;  
    String nome;  
    String descricao;  
    Double precoVenda;  
    Double precoCompra;  
    LocalDate dataValidade;  
    LocalDate dataPrazo;  
    Status status;  
}
```

Vamos alterar todos os nossos atributos : public

```
public class Produto {  
    public Integer id;  
    public String nome;  
    public String descricao;  
    public Double precoVenda;  
    public Double precoCompra;  
    public LocalDate dataValidade;  
    public LocalDate dataPrazo;  
    public Status status;  
}
```

Beleza! Criamos nossa primeira Classe.

Vamos utilizar nossa classe

```
public class Application {  
  
    public static void main(String[] args) {  
  
        Produto produto = new Produto();  
  
    }  
}
```

Como podemos definir os valores para nossos atributos?

Definindo valores dos atributos

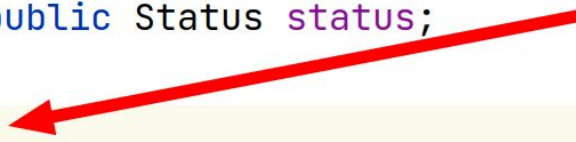
```
public class Application {  
    public static void main(String[] args) {  
        Produto produto = new Produto();  
        produto.nome = "Impressora 3D HP";  
        produto.descricao = "Impressora 3D filamento XYZ";  
        produto.dataPrazo = LocalDate.of(2023, 01, 15);  
        produto.precoCompra = 1200.00;  
        produto.precoVenda = 1500.00;  
        produto.status = Status.DISPONIVEL;  
    }  
}
```

Não é uma boa prática acessarmos nossos atributos diretamente, para resolver isso vamos falar de Métodos.

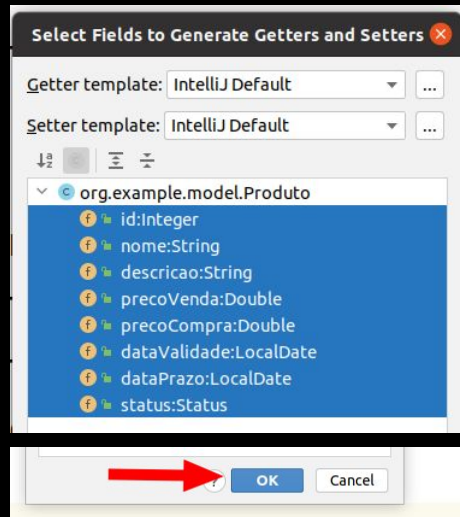
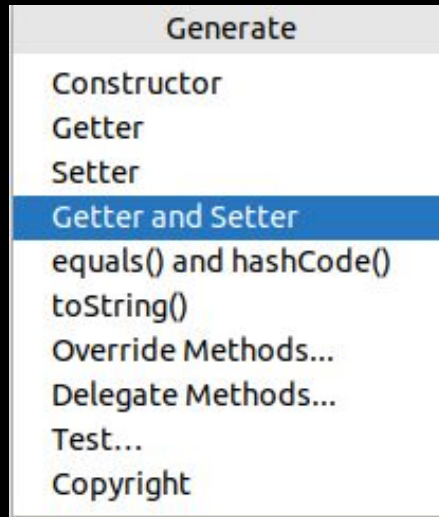
Em programação, um método é um bloco de código que realiza uma tarefa específica. Os métodos são usados para dividir programas complexos em partes menores e mais gerenciáveis, o que facilita o desenvolvimento, a depuração e a manutenção do código.

Criando um métodos

```
public class Produto {  
    public Integer id;  
    public String nome;  
    public String descricao;  
    public Double precoVenda;  
    public Double precoCompra;  
    public LocalDate dataValidade;  
    public LocalDate dataPrazo;  
    public Status status;  
}
```

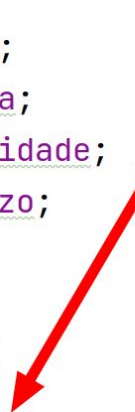


1. Clique com o botão direito
2. Clique em generate (alt + insert)



Criando um métodos

```
public class Produto {  
    public Integer id;  
    public String nome;  
    public String descricao;  
    public Double precoVenda;  
    public Double precoCompra;  
    public LocalDate dataValidade;  
    public LocalDate dataPrazo;  
    public Status status;  
  
    public Integer getId() {  
        return id;  
    }  
  
    public void setId(Integer id) {  
        this.id = id;  
    }  
}
```



```
public String getNome() {  
    return nome;  
}  
  
public void setNome(String nome) {  
    this.nome = nome;  
}  
  
public String getDescricao() {  
    return descricao;  
}  
  
public void setDescricao(String descricao) {  
    this.descricao = descricao;  
}
```

Podemos utilizar esses métodos para atribuímos os valores aos nossos objetos?

Definindo valores dos atributos via método

```
public class Application {  
    public static void main(String[] args) {  
        Produto produto = new Produto();  
        produto.setNome("Impressora 3D HP");  
        produto.setDescricao("Impressora 3D filamento XYZ");  
        produto.setDataPrazo(LocalDate.of(2023, 01, 15));  
        produto.setPrecoCompra(1200.00);  
        produto.setPrecoVenda(1500.00);  
        produto.setStatus(Status.DISPONIVEL);  
    }  
}
```

Métodos são apenas para setar valores? Não!

Vamos criar o método calculaMargemDeLucro

```
public Double calculaMargemDeLucro() {  
    double lucro = precoVenda - precoCompra;  
    double margemLucro = (lucro / precoVenda) * 100;  
    return margemLucro;  
}
```

Vamos usar esse método que acabamos de criar

```
public class Application {  
    public static void main(String[] args) {  
        Produto produto = new Produto();  
        produto.setNome("Impressora 3D HP");  
        produto.setDescricao("Impressora 3D filamento XYZ");  
        produto.setDataPrazo(LocalDate.of(2023, 01, 15));  
        produto.setPrecoCompra(1200.00);  
        produto.setPrecoVenda(1500.00);  
        produto.setStatus(Status.DISPONIVEL);  
        System.out.println("Margem: " + produto.calculaMargemDeLucro() + "%");  
    }  
}
```

Esses métodos seriam a única maneira correta de atribuírmos valores a um objeto?

Construtores

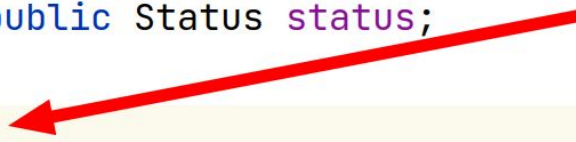
Em programação orientada a objetos, um construtor é um bloco de código especial que é chamado quando um objeto é criado a partir de uma classe. Ele permite que você inicialize o estado de um objeto quando ele é criado.

Isso é um construtor

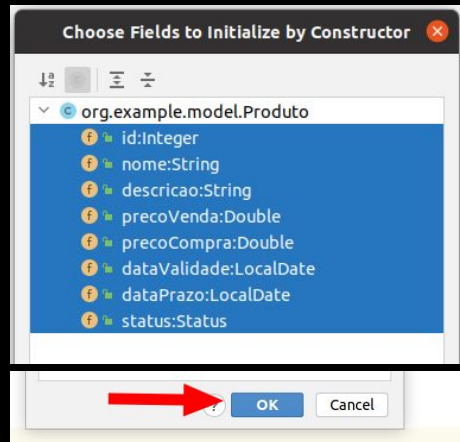
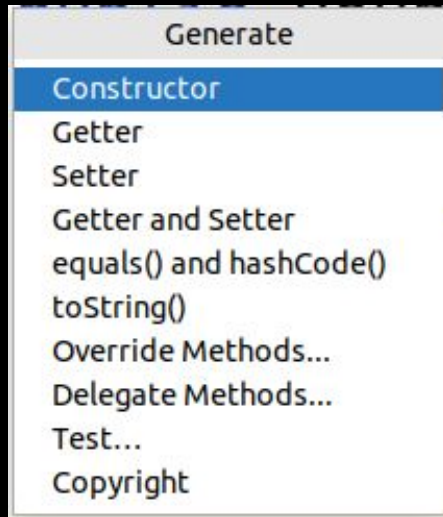
```
public static void main(String[] args) {  
    Produto produto = new Produto();  
  
    produto.setNome("Impressora 3D HP");  
    produto.setDescricao("Impressora 3D filamento XYZ");  
    produto.setDataPrazo(LocalDate.of(2023, 01, 15));  
    produto.setPrecoCompra(1200.00);  
    produto.setPrecoVenda(1500.00);  
    produto.setStatus(Status.DISPONIVEL);  
    System.out.println("Margem: " + produto.calculaMargemDeLucro() + "%");  
}
```

Criando um construtor

```
public class Produto {  
    public Integer id;  
    public String nome;  
    public String descricao;  
    public Double precoVenda;  
    public Double precoCompra;  
    public LocalDate dataValidade;  
    public LocalDate dataPrazo;  
    public Status status;  
}
```



1. Clique com o botão direito
2. Clique em generate (alt + insert)



Criando um construtor

```
public Produto(Integer id, String nome, String descricao, Double precoVenda,
                Double precoCompra, LocalDate dataValidade,
                LocalDate dataPrazo, Status status) {
    this.id = id;
    this.nome = nome;
    this.descricao = descricao;
    this.precoVenda = precoVenda;
    this.precoCompra = precoCompra;
    this.dataValidade = dataValidade;
    this.dataPrazo = dataPrazo;
    this.status = status;
}
```

Utilizando um construtor

```
public class Application {  
    public static void main(String[] args) {  
        Produto produto = new Produto(10, "Impressora 3D HP",  
            "Impressora 3D filamento XYZ", 1200.00, 1500.00,  
            LocalDate.of(2023, 01, 15),  
            LocalDate.of(2023, 01, 15), Status.DISPONIVEL);  
  
        // produto.setNome("Impressora 3D HP");  
        // produto.setDescricao("Impressora 3D filamento XYZ");  
        // produto.setDataPrazo(LocalDate.of(2023, 01, 15));  
        // produto.setPrecoCompra(1200.00);  
        // produto.setPrecoVenda(1500.00);  
        // produto.setStatus(Status.DISPONIVEL);  
        System.out.println("Margem: " + produto.calculaMargemDeLucro() + "%");  
    }  
}
```

Outros exemplos de construtores

```
public Produto(String nome, String descricao) {  
    this.nome = nome;  
    this.descricao = descricao;  
}
```

```
public Produto(String nome, Double precoVenda, Double precoCompra, Status status) {  
    this.nome = nome;  
    this.precoVenda = precoVenda;  
    this.precoCompra = precoCompra;  
    this.status = status;  
}
```

Encapsulamento

Imagine que nosso cliente pediu adicionássemos uma regra no nosso sistema para que nunca pudessem ter uma margem de lucro menor que 20%.

Onde poderíamos adicionar essa regra?

Vamos começar alterando nosso setPrecoVenda

```
public void setPrecoVenda(Double precoVenda) {  
    this.precoVenda = precoVenda;  
    if (this.calculaMargemDeLucro() < 20.0) {  
        System.out.println("A Margem de lucro deve ser sempre maior ou igual a 20%");  
    }  
}
```

Vamos começar alterando nosso setPrecoVenda

```
public class Application {  
    public static void main(String[] args) {  
        Produto produto = new Produto();  
        produto.setNome("Impressora 3D HP");  
        produto.setDescricao("Impressora 3D filamento XYZ");  
        produto.setDataPrazo(LocalDate.of(2023, 01, 15));  
        produto.setPrecoCompra(1200.00);  
        produto.setPrecoVenda(1400.00);  
        produto.setStatus(Status.DISPONIVEL);  
        System.out.println("Margem: " + produto.calculaMargemDeLucro() + "%");  
    }  
}
```

Vamos executar nosso programa

```
Application x

```

Beleza, a regra foi apresentada! Porém se eu fizesse isso?

```
public class Application {  
    public static void main(String[] args) {  
        Produto produto = new Produto();  
        produto.setNome("Impressora 3D HP");  
        produto.setDescricao("Impressora 3D filamento XYZ");  
        produto.setDataPrazo(LocalDate.of(2023, 01, 15));  
        produto.setPrecoCompra(1200.00);  
        produto.precoVenda = 1400.00;  
        produto.setStatus(Status.DISPONIVEL);  
        System.out.println("Margem: " + produto.calculaMargemDeLucro() + "%");  
    }  
}
```

```
n: Application x
/home/bruno.kurzawe/Documentos/senac/ExemploAula
.Application
Margem: 14.285714285714285%

Process finished with exit code 0
```


Burlamos a regra!! Deveríamos deixar essa regra ser burlada assim facilmente?

Para resolver isso vamos usar o encapsulamento

O encapsulamento é um dos quatro princípios fundamentais da programação orientada a objetos (POO), juntamente com a herança, a abstração e o polimorfismo. O propósito do encapsulamento é controlar como as variáveis e métodos de uma classe podem ser acessados ou modificados.

Campos/variáveis: Geralmente são declarados como `private` para restringir o acesso direto. Em vez disso, métodos `get` e `set` (conhecidos como "getters" e "setters") são usados para ler e modificar os valores desses campos, respectivamente.


Métodos: Podem ser declarados como public se quisermos que sejam acessíveis de fora da classe, ou private se eles são apenas para uso interno dentro da classe.

- private
- public
- protected

Vamos alterar todos os nossos campos : private

```
public class Produto {  
    private Integer id;  
    private String nome;  
    private String descricao;  
    private Double precoVenda;  
    private Double precoCompra;  
    private LocalDate dataValidade;  
    private LocalDate dataPrazo;  
    private Status status;
```

Automaticamente a IDE já reclama

```
public class Application {  
    public static void main(String[] args) {  
        Produto produto = new Produto();  
        produto.setNome("Impressora 3D HP");  
        produto.setDescricao("Impressora 3D filamenta XY7");  
        produto.setDataPrazo(LocalDate.of(2023, 1, 1));  
        produto.setPrecoCompra(1200.00);  
        produto.precoVenda = 1400.00;   
        produto.setStatus(Status.DISPONIVEL);  
        System.out.println("Margem: " + produto.calculaMargemDeLucro() + "%");  
    }  
}
```

'precoVenda' has private access in 'org.example.model.Produto'

org.example.model.Produto

private Double precoVenda

Exemplo0001

Para funcionar

```
public class Application {  
    public static void main(String[] args) {  
        Produto produto = new Produto();  
        produto.setNome("Impressora 3D HP");  
        produto.setDescricao("Impressora 3D filamento XYZ");  
        produto.setDataPrazo(LocalDate.of(2023, 01, 15));  
        produto.setPrecoCompra(1200.00);  
        produto.setPrecoVenda(1400.00);  
        produto.setStatus(Status.DISPONIVEL);  
        System.out.println("Margem: " + produto.calculaMargemDeLucro() + "%");  
    }  
}
```

Fim da aula 02...