

# Engenharia de Softwares Escaláveis

Design Patterns e Domain-Driven Design com Java

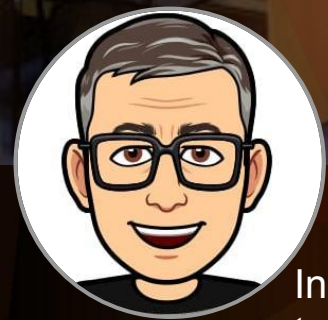
# Agenda

## **Etapas 1:** Design Patterns e Princípios SOLID com Java.

- Apresentações.
- Engenharia de Software.
- APIE - Abstraction, Polymorphism, Inheritance, Encapsulation.



# Apresentações



## Armênio Cardoso

Iniciei minha carreira profissional em **1986** e desde **1990** procuro conciliar o trabalho em **Engenharia de Software** com o de **Professor**.

Particpei em diversos projetos, atuando com modelagem, arquitetura e programação. Fui desenvolvedor Pascal, Clipper, Visual Basic, C/C++, Java, Kotlin, JavaScript, TypeScript.

Em **2002** vim para o Infnet onde dou aulas nos cursos de Graduação, Pós-Graduação e de Extensão.

Desde **2012** trabalho como Sênior Staff Engineer / Tech Leader na DASA - Diagnósticos da América S.A., empresa da área de medicina diagnóstica.



<http://www.linkedin.com/in/armeniocardoso>

## Bloco: Engenharia de Softwares Escaláveis

```
graph TD; A[Bloco: Engenharia de Softwares Escaláveis] --> B[Design Patterns e Domain-Driven Design com Java]; A --> C[Domain-Driven Design e Arquitetura de Softwares Escaláveis com Java];
```

### Design Patterns e Domain-Driven Design com Java

- Desenvolver software aplicando design patterns.
- Projetar softwares de forma estratégica, usando "bounded contexts", subdomínios e linguagem ubíqua.
- Projetar softwares de forma estratégica usando "context maps".
- Projetar softwares usando "aggregates".

### Domain-Driven Design e Arquitetura de Softwares Escaláveis com Java

- Transformar monólitos em microserviços eficazes, aplicando princípios de DDD e técnicas de decomposição.
- Projetar softwares usando "domain events".
- Desenvolver microserviços event-driven e com outros padrões de comunicação assíncrona.
- Implementar testes e observabilidade em microserviços com Zipkin, Spring Cloud Sleuth e ELK Stack.

# Engenharia de Software



O Que é  
Engenharia de  
Software?

Tech Manager

Tech Leader

Treinamento

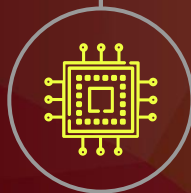


**Pessoas**

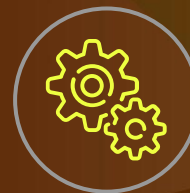
Infra

Tecnologia

Tools



**Ferramentas**



**Processos**

Agilidade

GMUD

DOC / GI



**Eficiência**

**Qualidade**

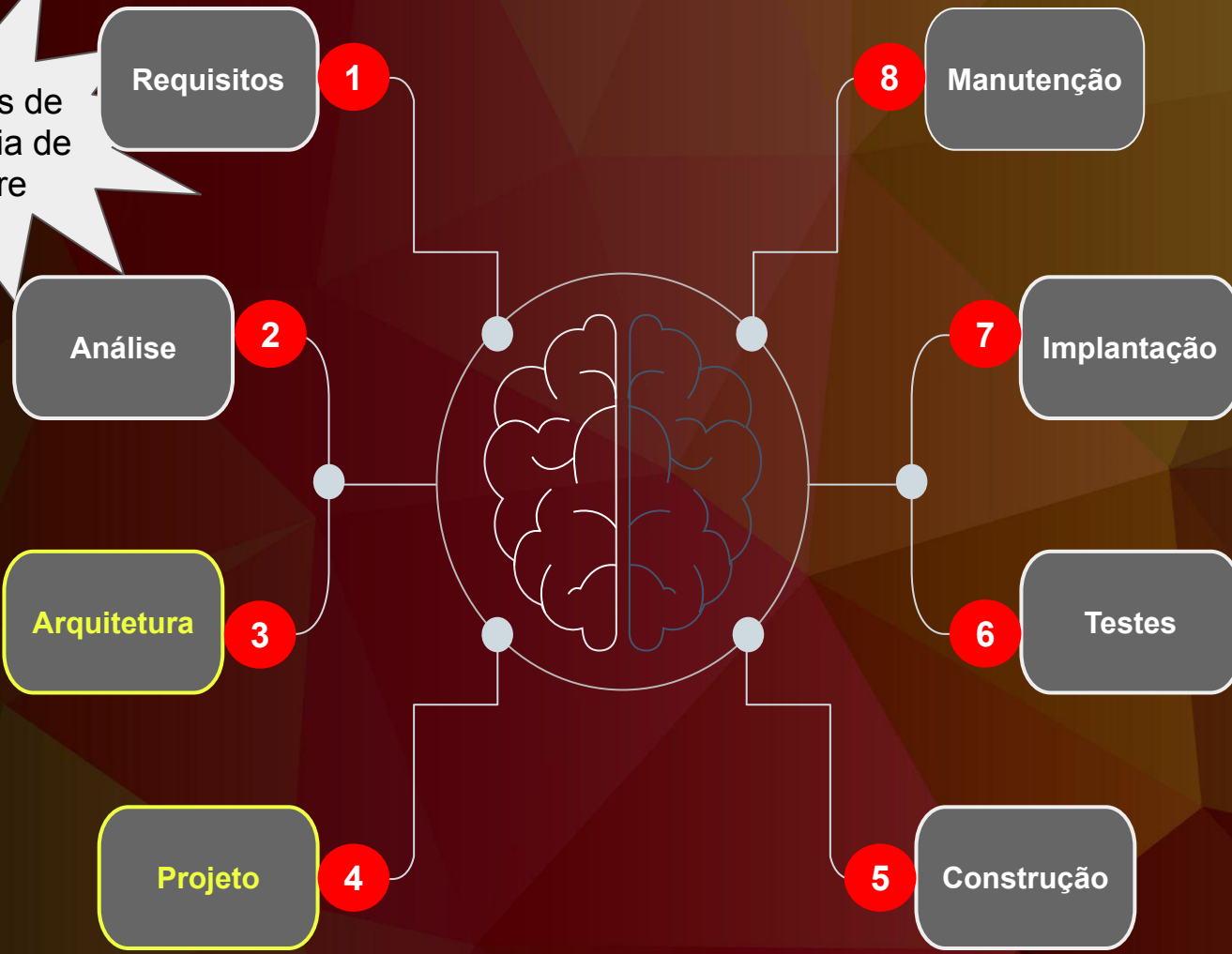
## O Que Faz o Engenheiro de Software?

Viabilizam **projetos de construção de softwares**, empregando **processos**, **pessoas** e **ferramentas**, de forma que atendam aos requisitos definidos pelos usuários, **buscando eficiência e qualidade**.

1. Gerenciar projetos de desenvolvimento de software.
2. Elaborar a **arquitetura e projetar soluções de software**.
3. Prospectar novas ferramentas e processos.
4. Programar módulos dos sistemas.
5. Liderar equipes de desenvolvimento e testes.



# Atividades de Engenharia de Software



## Objetivos do Engenheiro de Software

**Produzir uma solução potencialmente consumível.**

Aproximar-se de uma versão implantável.

**Comprovar a arquitetura antecipadamente.**

**Cumprir a missão do projeto.**



Aumente as habilidades dos membros da equipe.

Atender às necessidades em constante mudança das partes interessadas.

**Manter ou melhorar os níveis de qualidade existentes.**

Melhorar a infraestrutura existente.



# APIE - Abstraction, Polymorphism, Inheritance, Encapsulation

Projetar sistemas bem estruturados, reutilizáveis e fáceis de manter.

Trabalhar com frameworks e linguagens modernas que usam intensamente POO, como Java, C#, Python, etc.



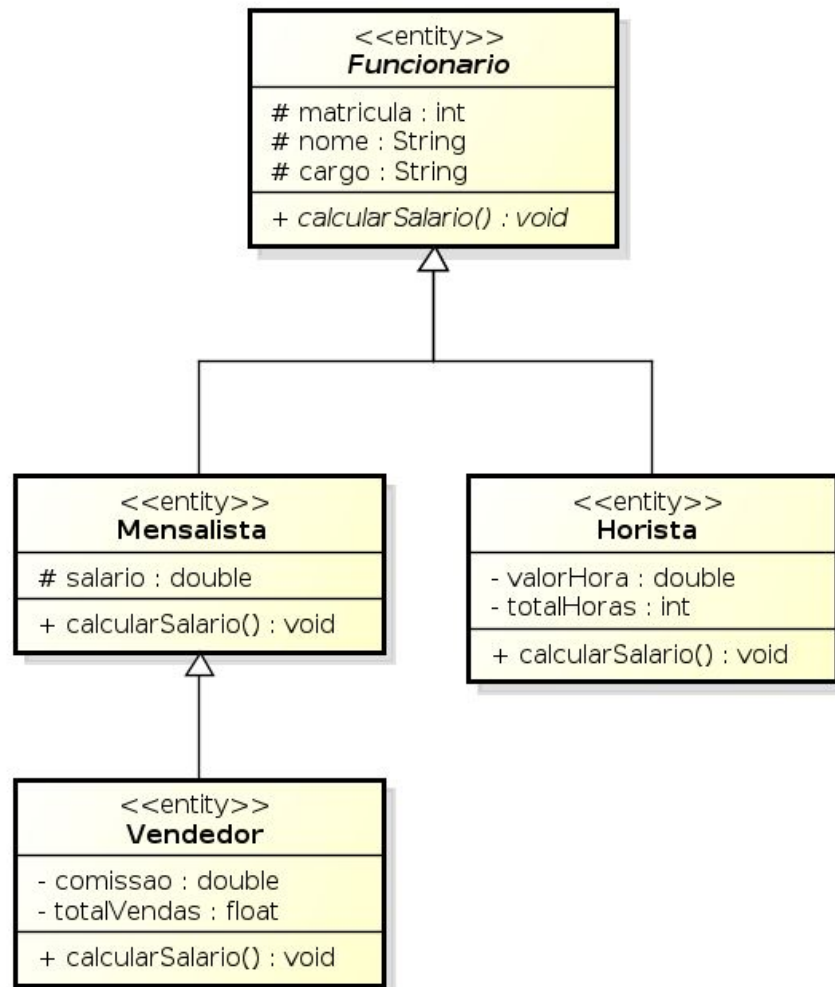
Aplicar boas práticas de desenvolvimento (como SOLID).

## Abstração

A chave para este conceito é a remoção constante de itens específicos ou detalhes individuais para alcançar a generalização da finalidade do objeto.

Com esse conhecimento, criamos uma abstração adequada, uma classe abstrata que pode ser herdada posteriormente ao construir uma classe modelo específica.

Podemos implementar **Abstração** através de classes abstratas ou por meio de interfaces.



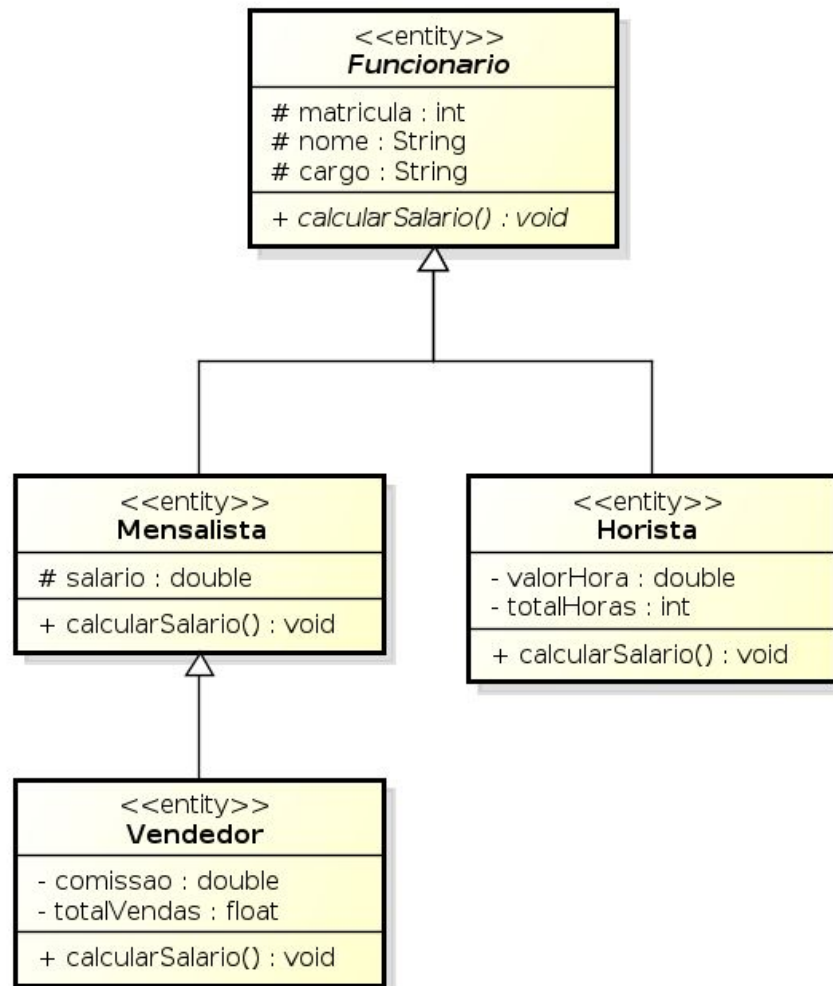
```
public abstract class Produto {  
    protected String nome;  
    protected double precoBase;  
  
    public Produto(String nome, double precoBase) {  
        this.nome = nome;  
        this.precoBase = precoBase;  
    }  
  
    public String getNome() {  
        return nome;  
    }  
  
    public double getPrecoBase() {  
        return precoBase;  
    }  
  
    public void setPrecoBase(double precoBase) {  
        if (precoBase > 0) {  
            this.precoBase = precoBase;  
        }  
    }  
  
    public abstract double calcularPrecoFinal();  
}
```



## Herança

Todas as subclasses vão herdar as propriedades que não são *private*, pois as mesmas são comuns para todas.

Aqui vemos a sobrescrita onde é possível adaptar uma funcionalidade para a realidade específica da classe.



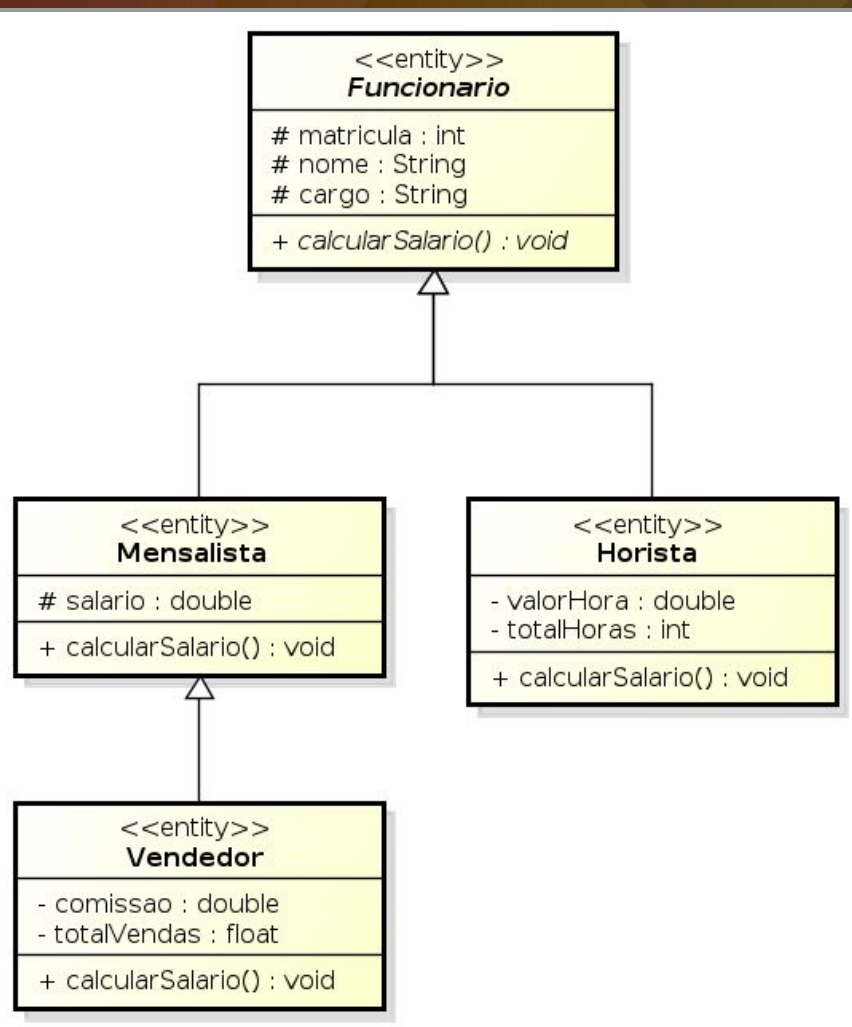


```
public class ProdutoFisico extends Produto {  
    private double frete;  
  
    public ProdutoFisico(String nome, double  
precoBase, double frete) {  
        super(nome, precoBase);  
        this.frete = frete;  
    }  
  
    @Override  
    public double calcularPrecoFinal() {  
        return getPrecoBase() + frete;  
    }  
}
```

```
public class ProdutoDigital extends Produto {  
    private double desconto;  
  
    public ProdutoDigital(String nome, double  
precoBase, double desconto) {  
        super(nome, precoBase);  
        this.desconto = desconto;  
    }  
  
    @Override  
    public double calcularPrecoFinal() {  
        return getPrecoBase() - desconto;  
    }  
}
```

**Polimorfismo Estático** é o uso de sobrecarga de métodos, onde temos métodos com mesmo nome, mas parâmetros diferentes. O método correto é resolvido durante a compilação do programa.

**Polimorfismo Dinâmico** é o uso de sobrescrita de métodos, onde o método é conhecido em tempo de execução. O método substituído é chamado por meio de referência à instância do objeto ao qual ele pertence.

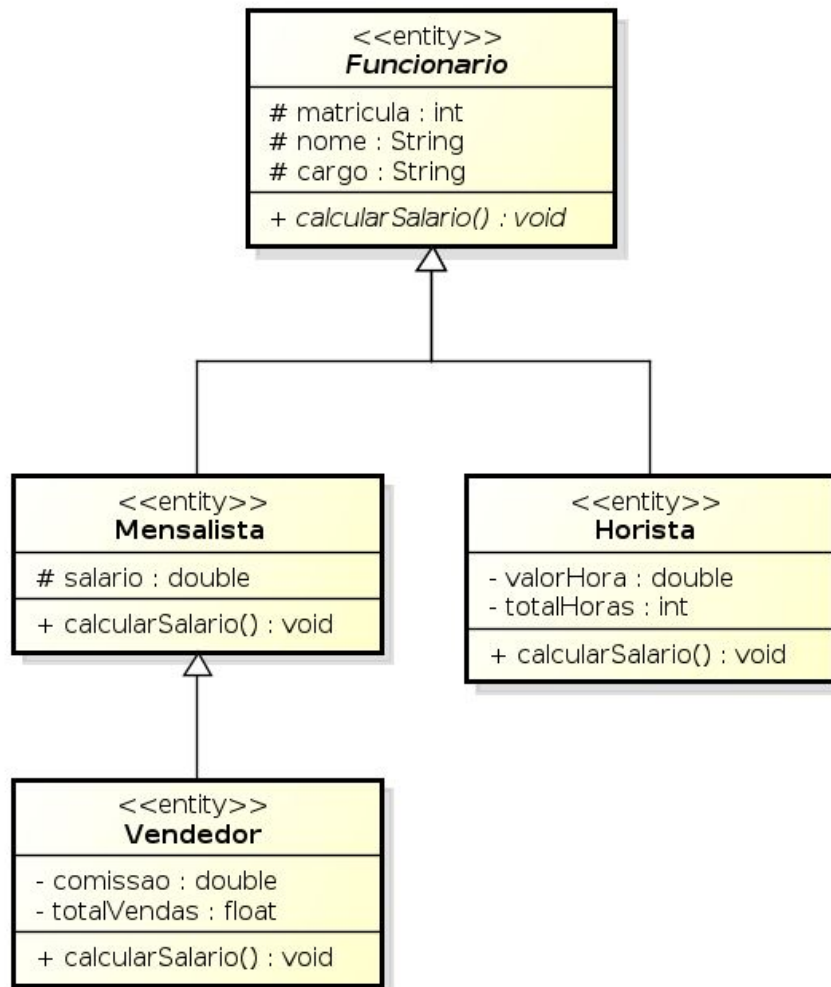


```
public class Loja {  
    public static void main(String[] args) {  
        Produto livro = new ProdutoFisico("Livro de Java", 50.0, 10.0);  
        Produto curso = new ProdutoDigital("Curso Online de Java", 100.0, 30.0);  
  
        imprimirPreco(livro);  
        imprimirPreco(curso);  
    }  
  
    public static void imprimirPreco(Produto produto) {  
        System.out.printf("%s: R$ %.2f\n", produto.getNome(), produto.calcularPrecoFinal());  
    }  
}
```

## Encapsulamento

O estado de uma instância pode ser alterado ou atualizado por meio de métodos ou campos expostos; todo o resto está oculto do mundo exterior.

Todos os elementos da classe estão ocultos e somente **calcularSalario** está disponível para ser usado.

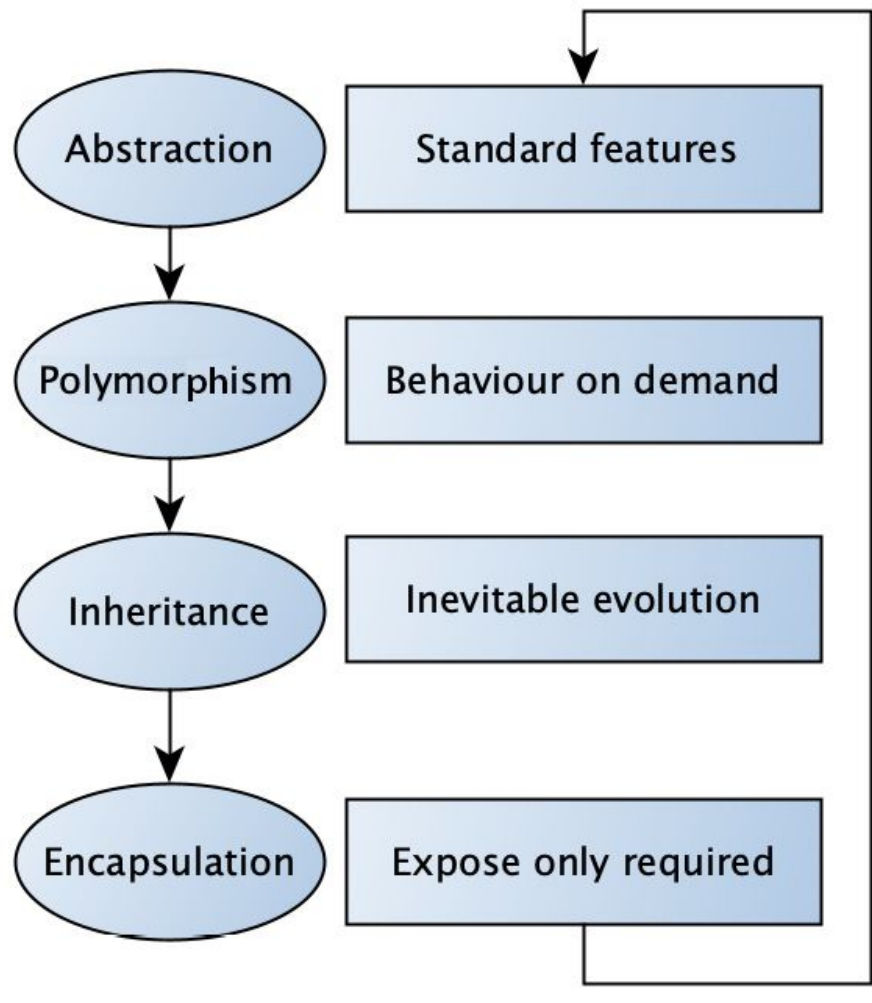


## APIE

A motivação de cada um dos pilares mencionados nas seções anteriores é introduzir estrutura no código por meio de um determinado conjunto de conceitos.

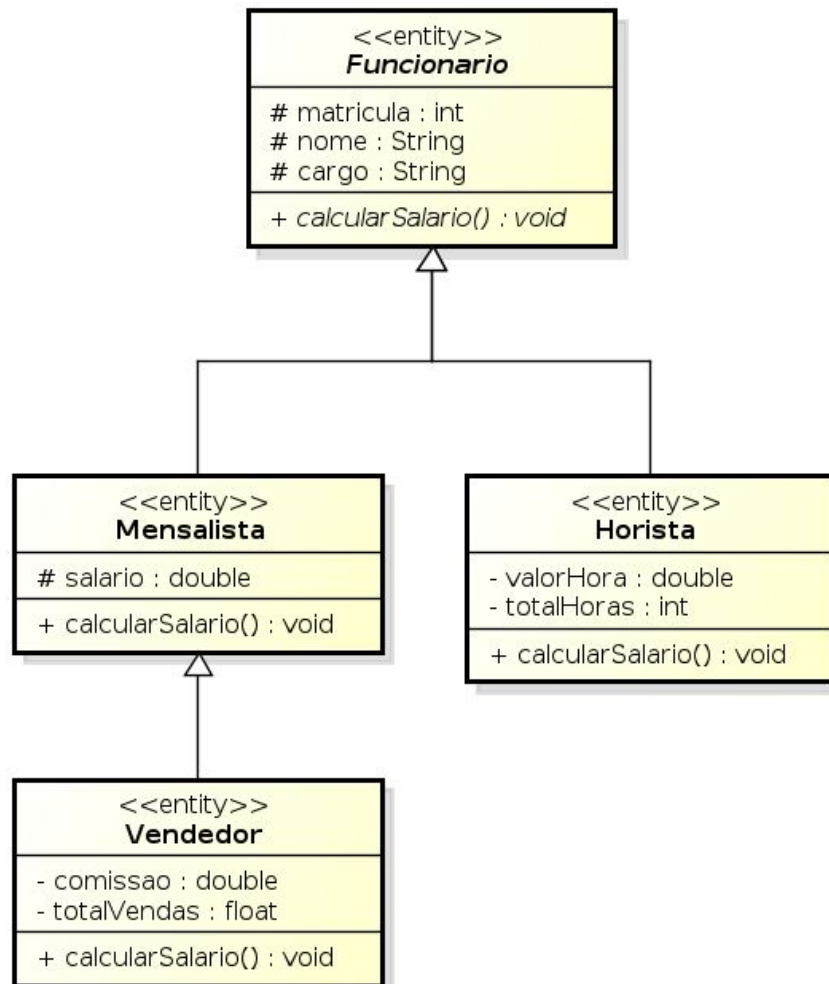
Os pilares são definidos e complementares.

Embora estes quatro pilares pareçam triviais, é incrivelmente difícil segui-los.

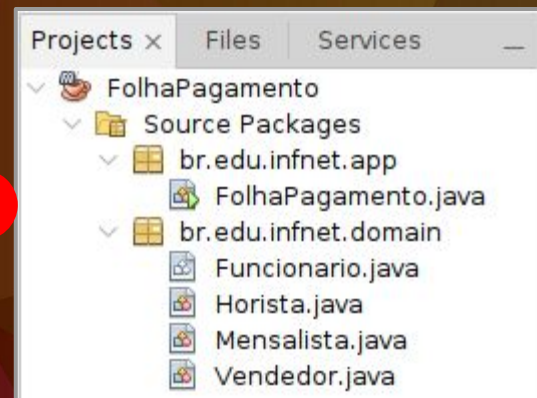




Demonstração

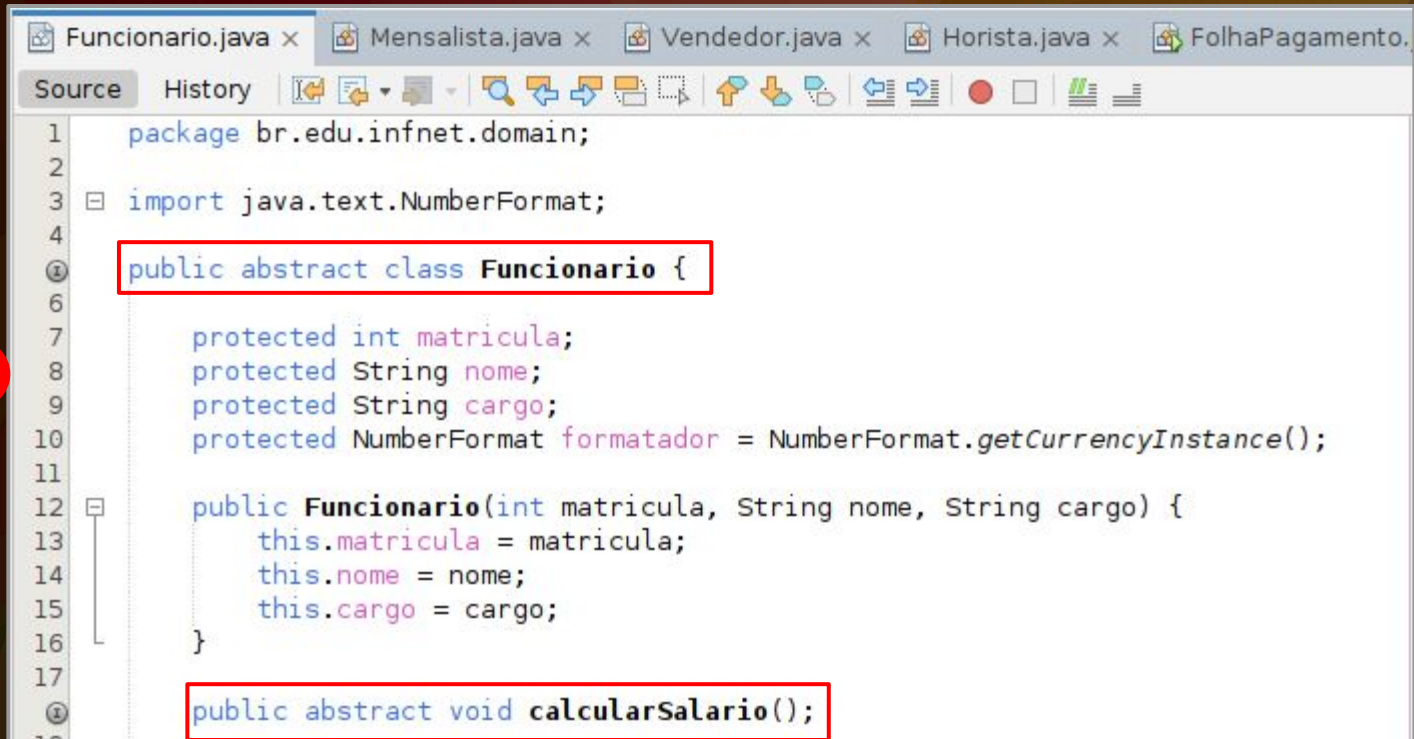


1





2



```
1 package br.edu.infnet.domain;
2
3 import java.text.NumberFormat;
4
5 public abstract class Funcionario {
6
7     protected int matricula;
8     protected String nome;
9     protected String cargo;
10    protected NumberFormat formatador = NumberFormat.getCurrencyInstance();
11
12    public Funcionario(int matricula, String nome, String cargo) {
13        this.matricula = matricula;
14        this.nome = nome;
15        this.cargo = cargo;
16    }
17
18    public abstract void calcularSalario();
19 }
```

3

```
Funcionario.java x Mensalista.java x Vendedor.java x Horista.java x
Source History
19 public abstract void calcularSalario();
20
21 public void mostrarCabecalho() {
22     System.out.println("----- Contracheque -----");
23     System.out.println("Nome....." + nome);
24     System.out.println("Cargo....." + cargo);
25 }
26
27 public double calcularImposto(double valor) {
28     double imposto = 0;
29     if (valor > 2000) {
30         imposto = valor * 0.15;
31     } else if (valor > 3000) {
32         imposto = valor * 0.25;
33     } else if (valor > 5000) {
34         imposto = valor * 0.30;
35     }
36     return imposto;
37 }
```

4

```
Funcionario.java x Mensalista.java x Vendedor.java x Horista.java x FolhaPagamento.java
Source History
1 package br.edu.infnet.domain;
2
3
4 public class Mensalista extends Funcionario {
5
6     protected double salario;
7
8     public Mensalista(int matricula, String nome, String cargo, double salario) {
9         super(matricula, nome, cargo);
10        this.salario = salario;
11    }
12
13    @Override
14    public void calcularSalario() {
15        super.mostrarCabecalho();
16        System.out.println("Salário Bruto..." + formatador.format(salario));
17        double imposto = calcularImposto(this.salario);
18        double salarioLiq = this.salario - imposto;
19        System.out.println("Imposto....." + formatador.format(imposto));
20        System.out.println("Salário Líquido." + formatador.format(salarioLiq));
21    }
22 }
```

5

Funcionario.java x Mensalista.java x Vendedor.java x Horista.java x FolhaPagamento.java x

Source

History



```
1 package br.edu.infnet.domain;
```

```
2  
3 public class Vendedor extends Mensalista {
```

```
4  
5     private final double comissao;
```

```
6     private final double totalVendas;
```

```
7  
8     public Vendedor(int matricula, String nome, double salario, double comissao, float totalVendas) {
```

```
9         super(matricula, nome, "Vendedor", salario);
```

```
10        this.comissao = comissao;
```

```
11        this.totalVendas = totalVendas;
```

```
12    }
```

```
13  
14    @Override
```

```
15    public void calcularSalario() {
```

```
16        super.mostrarCabecalho();
```

```
17        System.out.println("Salário Bruto..." + formatador.format(salario));
```

```
18        double comissoes = totalVendas * comissao;
```

```
19        System.out.println("Comissões....." + formatador.format(comissoes));
```

```
20        double imposto = calcularImposto(this.salario + comissoes);
```

```
21        double salarioLiq = this.salario + comissoes - imposto;
```

```
22        System.out.println("Imposto....." + formatador.format(imposto));
```

```
23        System.out.println("Salário Líquido." + formatador.format(salarioLiq));
```

```
24    }
```

```
25 }
```



6

```
Funcionario.java x Mensalista.java x Vendedor.java x Horista.java x FolhaPagamento.java x
Source History
1 package br.edu.infnet.domain;
2
3 public class Horista extends Funcionario {
4
5     private final double valorHora;
6     private final int totalHoras;
7
8     public Horista(int matricula, String nome, String cargo, double valorHora, int totalHoras) {
9         super(matricula, nome, cargo);
10        this.valorHora = valorHora;
11        this.totalHoras = totalHoras;
12    }
13
14    @Override
15    public void calcularSalario() {
16        super.mostrarCabecalho();
17        double salBruto = valorHora * totalHoras;
18        System.out.println("Salário Bruto..." + formatador.format(salBruto));
19        double imposto = calcularImposto(salBruto);
20        double salarioLiq = salBruto - imposto;
21        System.out.println("Imposto....." + formatador.format(imposto));
22        System.out.println("Salário Líquido." + formatador.format(salarioLiq));
23    }
24 }
```

```
1 package br.edu.infnet.app;
2
3 import br.edu.infnet.domain.Funcionario;
4 import br.edu.infnet.domain.Horista;
5 import br.edu.infnet.domain.Mensalista;
6 import br.edu.infnet.domain.Vendedor;
7
8 public class FolhaPagamento {
9
10     public static void main(String[] args) {
11
12         Funcionario [] funcionarios = {
13             new Mensalista(123, "Machado de Assis", "Repórter", 3245.67),
14             new Vendedor(456, "Rachel de Queiroz", 2000, 0.05, 25000),
15             new Horista(789, "Clarice Lispector", "Professor", 30, 120)
16         };
17
18         System.out.println("==== Folha de Pagamento =====");
19         for (Funcionario funcionario : funcionarios) {
20             funcionario.calcularSalario();
21         }
22     }
23 }
```

Output - Run (FolhaPagamento) x

```
===== Folha de Pagamento =====
----- Contracheque -----
Nome.....Machado de Assis
Cargo.....Repórter
Salário Bruto...R$ 3.245,67
Imposto.....R$ 486,85
Salário Líquido.R$ 2.758,82
----- Contracheque -----
Nome.....Rachel de Queiroz
Cargo.....Vendedor
Salário Bruto...R$ 2.000,00
Comissões.....R$ 1.250,00
Imposto.....R$ 487,50
Salário Líquido.R$ 2.762,50
----- Contracheque -----
Nome.....Clarice Lispector
Cargo.....Professor
Salário Bruto...R$ 3.600,00
Imposto.....R$ 540,00
Salário Líquido.R$ 3.060,00
-----
```

