Trabalho Prático: Desenvolvimento de um Sistema Orientado a Objetos
Sistema Bancário

Igor Vinoski

Instituto Federal Sul-rio-grandense - Campus Passo Fundo

Passo Fundo - RS - Brasil

24 de agosto de 2024

Sumário

1 Objetivo **1**

2 Modelagem **2**

3 Implementação **4**

3.1 Domínio Pessoa 5

3.1.1 Classe Pessoa 5

3.1.2 Classe Funcionário 6

3.1.3 Classe Gerente 7

3.1.4 Classe Vendedor 8

3.2 Domínio Conta 9

3.2.1 Classe Conta 9

3.3 Domínio Relatório 10

3.3.1 Classe Relatório 10

1 Objetivo

Este trabalho integra a disciplina de Tecnologia de Orientação a Objetos e tem como objetivo a aplicação prática dos principais conceitos de Orientação a Objetos. Serão abordados, de forma contextualizada os quatro pilares fundamentais desta disciplina, que, de maneira resumida se constitutem em:

- **Herança**: reutilização e especialização de código.

- **Encapsulamento**: proteção de dados e integridade das classes.
- **Polimorfismo**: pode assumir mais de uma forma.
- **Abstração**: simplificação da complexidade através da modelagem de conceitos.

O trabalho consistirá na implementação simplificada de um Sistema Bancário, que visa aplicar os conceitos da disciplina. Incluirá o código, a diagramação[1] de classes e a implementação de testes unitários. O código completo pode ser encontrado em: <https://github.com/IgorVinoski/SistemaBancarioOO>

2 Modelagem

O sistema bancário contém, duas entidades principais. Primeiramente, a entidade PESSOA. Essa primeira entidade está estritamente relacionada com a segunda entidade, CONTA[2].

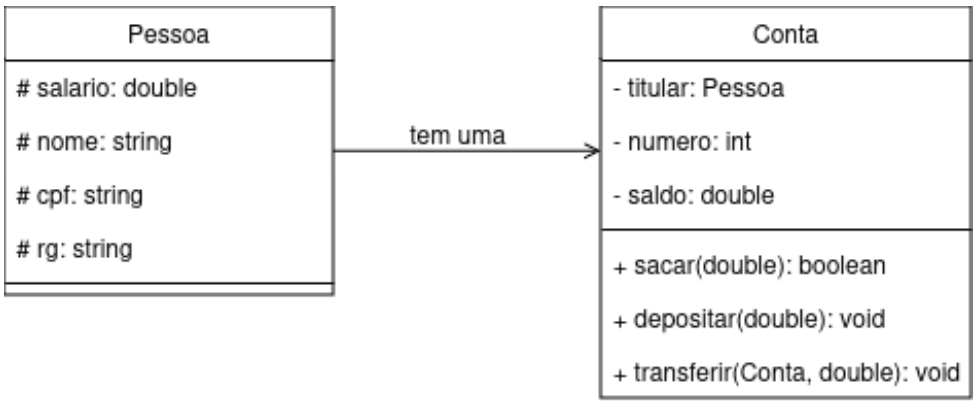

Figura 1: Diagrama das classes Pessoa e Conta.

Além disso, há outras 4 classes. Essas entidades são relacionadas à admnisitração do sistema bancário.

[1] A diagramação não necessariamente representa um diagrama UML Por isso, atente-se ao que ela quer representar, e não aos conceitos de diagramas UML.

[2] *getters e setters sem peculiaridades foram omitidos da modelagem para simplificação do diagrama.

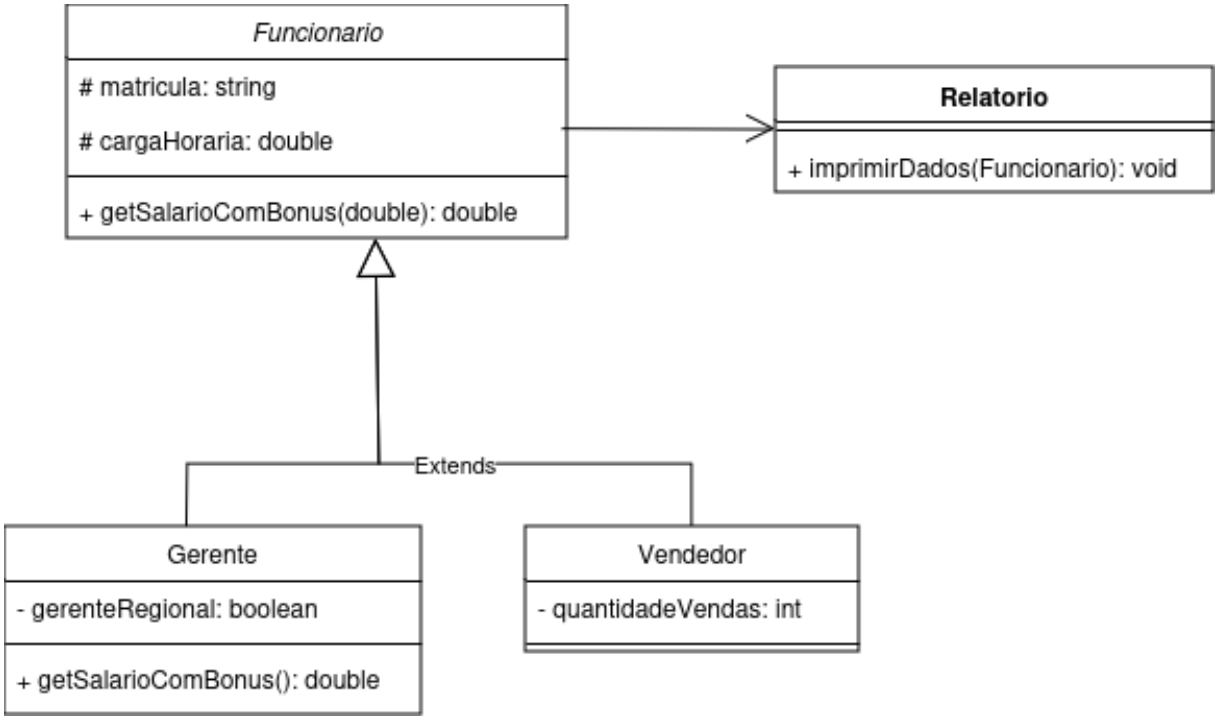

Figura 2: Diagrama das classses Funcionário, Relatório, Gerente e Vendedor.

Consequentemente, o modelo final do sistema é composto pela união entre Pessoa e Funcionário, resultando no seguinte diagrama:

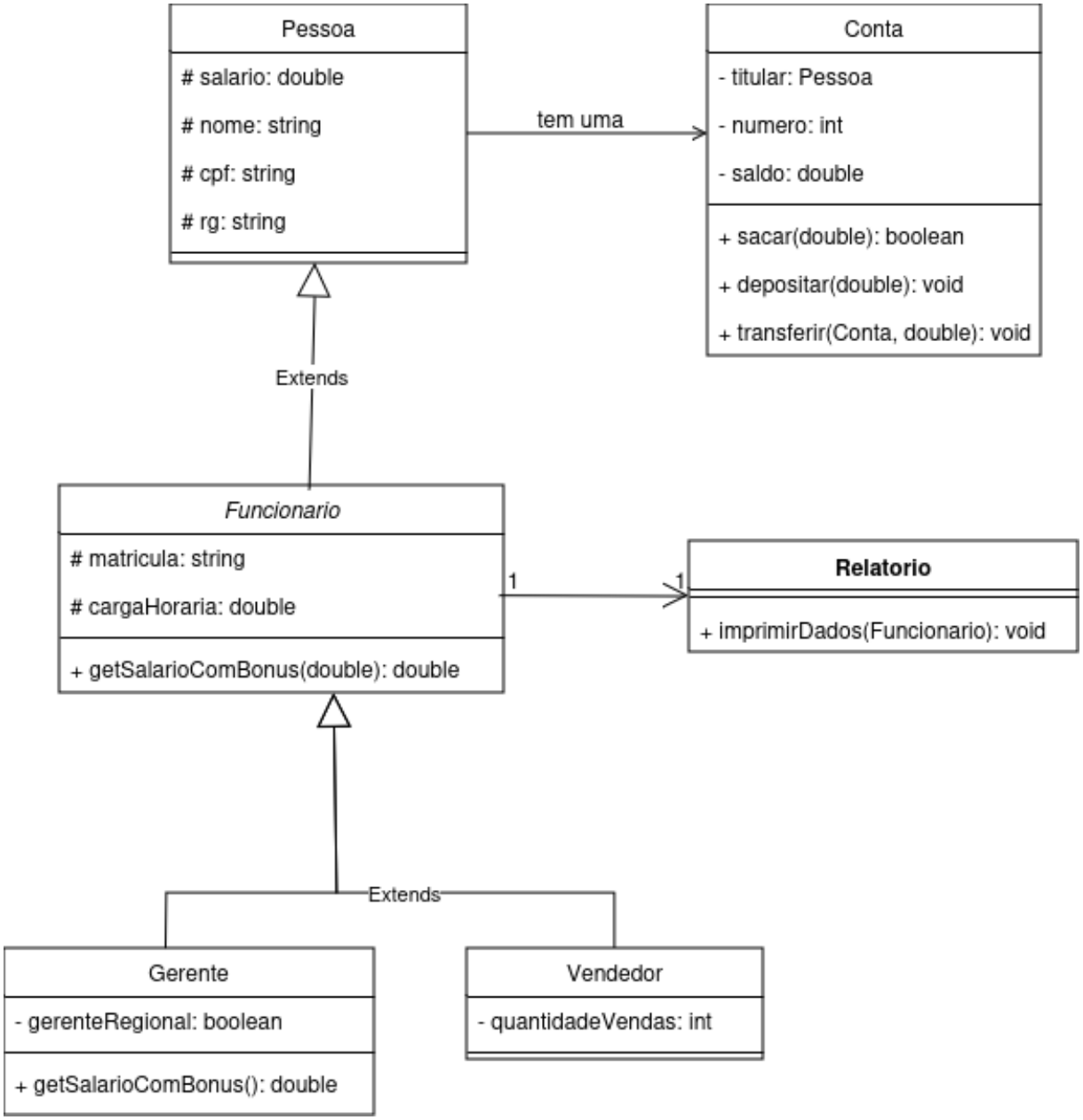

Figura 3: Diagrama do Sistema Bancário

3 Implementação

O sistema será dividido em 3 domínios (*packges*). Sendo elas: pessoa, conta e relatório.

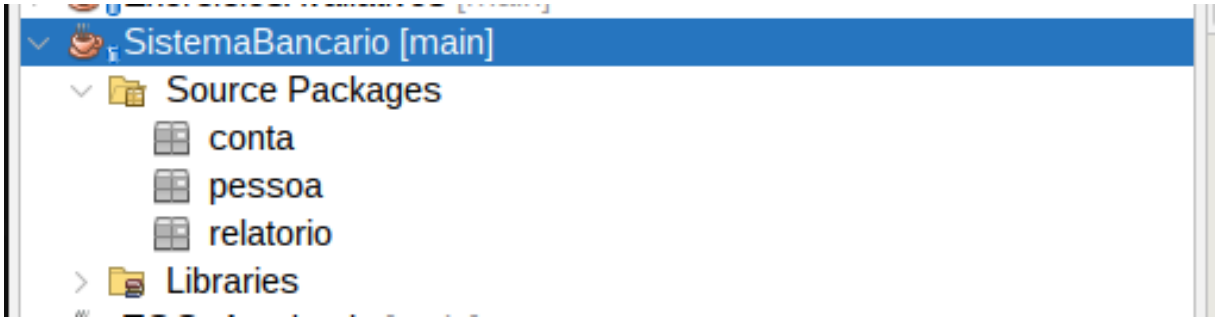

Figura 4: *Packges* do projeto.

3.1 Domínio Pessoa

A *packge* pessoa conterá 4 entidades. Sendo as classes: Pessoa, Funcionario (abstrata), Gerente e Vendedor.

3.1.1 Classe Pessoa

A classe pessoa encapusla os seus atributos com *protected*. Portanto, sem ser através dos métodos *getters* e *setters*, só é possível acessá-los através de herança.

```
package pessoa;

public class Pessoa {
    protected String nome;
    protected String cpf;
    protected String rg;
    protected double salario;


    public Pessoa(String nome, String cpf, String rg) {
        this.nome = nome;
        this.cpf = cpf;
        this.rg = rg;
    }

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }
```

```
    public String getCpf() {
        return cpf;
    }

    public void setCpf(String cpf) {
        this.cpf = cpf;
    }

    public String getRg() {
        return rg;
    }

    public void setRg(String rg) {
        this.rg = rg;
    }

    public double getSalario() {
        return salario;
    }

    public void setSalario(double salario) {
        this.salario = salario;
    }
}
```

3.1.2 Classe Funcionário

A classe Funcionário herda os comportamentos da classe Pessoa, ela também é uma classe abstrata. Portanto, assina os métodos e atributos que todos os funcionários devem ter. Mesmo que, para diferentes tipos de funcionários, a lógica interna desses métodos mude, a sua assinatura continuará a mesma. Essa implementação evidencia os conceitos de Abstração e Polimorfismo. Além disso, no método $getSalarioComBonus()$ é possível observar as diferentas formas que um método pode assumir. POr fim, também pela visibilidade de seus atributos, a classe mantém o pilar de Encapsulamento.

```
package pessoa;

public abstract class Funcionario extends Pessoa {
  public static final double PERCENTUAL_PADRAO_BONUS = 10;
```

```
  public static final int CARGA_HORARIA_PADRAO = 220;
    private final double CEM = 100;
    protected String matricula;
    protected double cargaHoraria;

    public Funcionario(String nome, String cpf, String rg, String matricula ) {
        super(nome, cpf, rg);
        this.matricula=matricula;
        this.cargaHoraria = CARGA_HORARIA_PADRAO;
    }

    public double getSalarioComBonus() {
        return this.salario + (salario * PERCENTUAL_PADRAO_BONUS /CEM);
    }
    public double getSalarioComBonus(double percentualBonus) {
        return this.salario + (salario * percentualBonus /CEM);
    }
}
```

3.1.3 Classe Gerente

Essa classe herda de Funcionário. Portanto, todo Gerente é um funcionário, contém em sua base, os mesmos atributos e métodos. Essa classe sobrescreve o método *getSalarioComBonus*() da classe abstrata que herda.

```
package pessoa;

public class Gerente extends Funcionario{
    public static final double PERCENTUAL_PADRAO_BONUS_GERENTE_REGIONAL= 20;

    private boolean gerenteRegional;

    public Gerente(String nome, String cpf, String rg, String matricula) {
        super(nome, cpf, rg, matricula);
    }
    public Gerente(String nome, String cpf, String rg,
    String matricula, boolean isGerenteRegional) {
        super(nome, cpf, rg, matricula);
        this.gerenteRegional=isGerenteRegional;
    }
```

```
    @Override
    public double getSalarioComBonus(){

        if(isGerenteRegional()){
return this.salario + (this.salario + PERCENTUAL_PADRAO_BONUS_GERENTE_REGIONAL /100);
        }
        return super.getSalarioComBonus();

    }
    public boolean isGerenteRegional() {
        return gerenteRegional;
    }

    public void setGerenteRegional(boolean gerenteRegional) {
        this.gerenteRegional = gerenteRegional;
    }
}
```

3.1.4 Classe Vendedor

Essa classe simplesmente contém *getter* e *setter* e herda os comportamentos de Funcionário.

```
package pessoa;

public class Vendedor extends Funcionario {
    private int quantidadeVendas;

    public Vendedor(String nome, String cpf, String rg, String matricula) {
        super(nome, cpf, rg, matricula);
    }

    public int getQuantidadeVendas() {
        return quantidadeVendas;
    }

    public void setQuantidadeVendas(int quantidadeVendas) {
        this.quantidadeVendas = quantidadeVendas;
    }
```

```
}
```

3.2 Domínio Conta

3.2.1 Classe Conta

A classe Conta contém a lógica de negocio da aplaicação bancária. Sua implementação encapsula os seus comportamentos, seguindo o padrão da implementação.

```
package conta;

import pessoa.Pessoa;

public class Conta {
    private Pessoa titular;
    private int numero;
    private double saldo;
    public Conta(int numero, Pessoa titular){
        this.numero=numero;
        this.titular=titular;
    }

    public Conta(int numero, Pessoa titular, double saldoInicial){
        this.numero=numero;
        this.depositar(saldoInicial);
    }
    public boolean sacar(double valorASacar){
        if(this.saldo >= valorASacar){
            this.saldo -= valorASacar;
            return true;
        }
        return false;
    }

    public void depositar(double saldoADepositar){
        this.saldo +=saldoADepositar;
        System.out.println("SALDO: " + this.saldo);

    }
```

```
    public void transferir(Conta contaDestino, double valorATransferir) {
        sacar(valorATransferir);
        contaDestino.depositar(valorATransferir);

    }
    public double getSaldo(){
        return this.saldo;
    }

    public int getNumero() {
        return numero;
    }

    public Pessoa getTitular() {
        return titular;
    }

    public void setTitular(Pessoa titular) {
        this.titular = titular;
    }
}
```

3.3 Domínio Relatório

3.3.1 Classe Relatório

Essa classe contém apenas um método, que recebe um Funcionário e imprime algumas informações. Assim, gerando um relatório. Esse método também evidencia o Polimorfismo e a Abstração, uma vez que, independetemente de qual o tipo de Funcionário passado para o método, ele será capaz de retornar a bonificação levando em consideração se é um Gerente ou Vendedor.

```
package relatorio;
import pessoa.Funcionario;
public class Relatorio {
    public void imprimirDados(Funcionario funcionario) {
System.out.println("Nome:" + funcionario.getNome()
+ " | Salrio: " + funcionario.getSalario()
+ " | Salrio com bonificao " + funcionario.getSalarioComBonus());
```

```
    }
}
```