

**ANÁLISE E DESENVOLVIMENTO DE SISTEMAS – 4º SEMESTRE MATUTINO – 2016**

**Lista 2 – Engenharia de Software III**

**NOMES:**

**Caio Larroza de Oliveira 1680481511006**

**Giovanni Armane 1680481511016**

**Leonardo Costa 1680481512015**

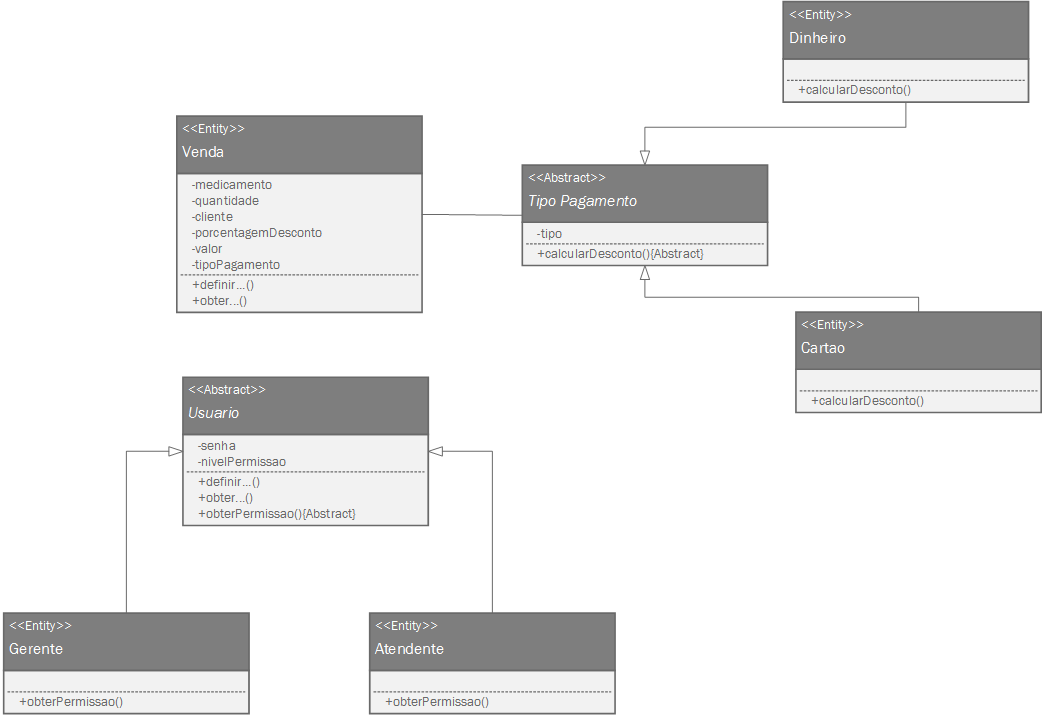
**Matheus dos Santos 1680481511044**

**SÃO CAETANO DO SUL**

**2016**

**Parte A**

**1 - Apresente a modelagem das relações de gen/espec do exercício 19 da Lista.**



**2 – As relações de gen/espec violam o Princípio de Liskov? Justifique a tua resposta.**

Nos dois casos, não fere o Princípio de Liskov pois, apesar das operações serem diferentes, os tipos de entrada e saídas são os mesmos entre as classes de mãe em comum.

**3 – Apresente a estrutura básica de código em JAVA, C# ou C++ para implementar as relações de gen/espec e as operações polimórficas.**

//TIPO PAGAMENTO//

public abstract class Tipo\_Pagamento{

private String tipo;

//Getter e Setter//

public abstract BigDecimal calcularDesconto(BigDecimal valor);

}

public class Dinheiro extends Tipo\_Pagamento{

@Override

public BigDecimal calcularDesconto(BigDecimal valor){

//CODIGO

}

}

public class Cartao extends Tipo\_Pagamento{

@Override

public BigDecimal calcularDesconto(BigDecimal valor){

//CODIGO

}

}

//USUÁRIO//

public abstract class Usuario{

private String tipo;

//Getter e Setter//

public abstract void obterPermissao();

}

public class Gerente extends Usuario{

@Override

public void obterPermissao(){

//CODIGO

}

}

public class Atendente extends Usuario{

@Override

public void obterPermissao(){

//CODIGO

}

}

**4 – As relações de gen/espec apresentam problema de Classificação Dinâmica? Justifique a tua resposta.**

No caso dos usuários, há uma violação pois um atendente pode vir a porventura se tornar um gerente.

No caso das formas de pagamento, não, pois não existe transição entre os tipos de pagamento. Ou ele é em cartão, recebendo um valor específico de desconto, ou é em dinheiro, recebendo outro valor de desconto.

**Parte B**

**5 – Apresente o diagrama de classes de projeto do exercício 18 da Lista 1.**



**6 – Transforme todos os relacionamentos de associação ou agregação entre classes de modelo para dependências estruturais. Explique a vantagem e desvantagem desse tipo de dependência.**



Dependência por atributo permite melhora no desempenho em tempo de execução, porém diminui o encapsulamento.

**7 – Apresente a estrutura básica de código em JAVA, C# ou C++ para implementar as dependências estruturais.**

//PRIMEIRO

public class Cliente\_Especial {

private String nome, rg, endereco;

private int telefone;

private long cpf;

private Date dataNascimento;

private List<Venda> vendas;

// Getters e Setters//

}

public class Medicamento {

private String nome, fabricante;

private Date dataValidade;

private BigDecimal preco;

private int qtdEstoque;

// Getters e Setters //

}

public class Venda {

private List<Medicamento> medicamento;

private List<int> quantidade;

private Cliente\_Especial cliente;

private double porcentagemDesconto;

private BigDecimal valor;

private Tipo\_Pagamento tipoPagamento;

// Getters e Setters//

}

//SEGUNDO

public class Caixa {

private String senhaAbertura, senhaFechamento;

private Notas notas;

private Moedas moedas;

// Getters e Setters//

}

public class Notas {

private int qtdDoisReais, qtdCincoReais, qtdDezReais,

qtdVinteReais, qtdCinquentaReais, qtdCemReais;

// Getters e Setters//

}

public class Moedas {

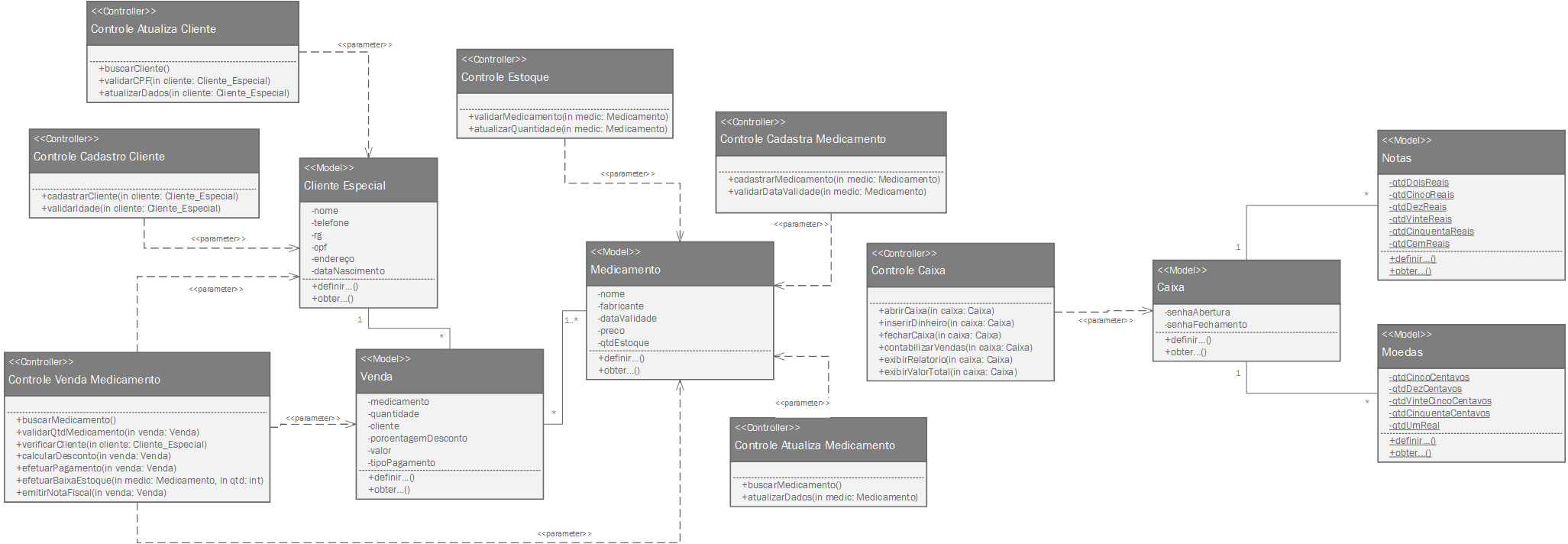
private int qtdCincoCentavos, qtdDezCentavos,

qtdVinteCincoCentavos, qtdCinquentaCentavos, qtdUmReal;

// Getters e Setters//

}

**8 – Transforme todos os relacionamentos de associação entre as classes de controle e modelo para dependências não estruturais por parâmetro. Explique a vantagem e desvantagem desse tipo de dependência.**



Dependência não estrutural por parâmetro tem por vantagens aumentar o encapsulamento e diminuir o acoplamento, ao passo que diminui o desempenho, sendo esta sua desvantagem.

**9 – Apresente a estrutura básica de código em JAVA, C# ou C++ para implementar as dependências não estruturais por parâmetro.**

public class Controle\_Atualiza\_Cliente {

public Cliente\_Especial buscarCliente() {

// CÓDIGO

}

public boolean validarCPF(Cliente\_Especial cliente) {

// CÓDIGO

}

public void atualizaDados(Cliente\_Especial cliente) {

// CÓDIGO

}

}

public class Controle\_Cadastro\_Cliente {

public boolean validarIdade(Cliente\_Especial cliente) {

// CÓDIGO

}

public void cadastrarCliente(Cliente\_Especial cliente) {

// CÓDIGO

}

}

public class Controle\_Estoque {

public boolean validarMedicamento(Medicamento medic) {

// CÓDIGO

}

public void atualizarQuantidade(Medicamento medic) {

// CÓDIGO

}

}

public class Controle\_Cadastro\_Medicamento {

public void cadastrarMedicamento(Medicamento medic) {

// CÓDIGO

}

public boolean validarDadaValidade(Medicamento medic) {

// CÓDIGO

}

}

public class Controle\_Caixa {

public Caixa abrirCaixa(Caixa caixa) {

// CÓDIGO

}

public void inserirDinheiro(Caixa caixa) {

// CÓDIGO

}

public void fecharCaixa(Caixa caixa) {

// CÓDIGO

}

public double contabilizarVendas(Caixa caixa) {

// CÓDIGO

}

public void exibirRelatorio(Caixa caixa) {

// CÓDIGO

}

public void exibirValorTotal(Caixa caixa) {

// CÓDIGO

}

}

public class Controle\_Atualiza\_Medicamento {

public Medicamento buscarMedicamento() {

// CÓDIGO

}

public void atualizaDados(Medicamento medic) {

// CÓDIGO

}

}

public class Controle\_Venda\_Medicamento {

public Medicamento buscarMedicamento() {

// CÓDIGO

}

public boolean validarQtdMedicamento(Venda venda) {

// CÓDIGO

}

public boolean verificarCliente(Cliente\_Especial cliente) {

// CÓDIGO

}

public double calcularDesconto(Venda venda) {

// CÓDIGO

}

public void efetuarPagamento(Venda venda) {

// CÓDIGO

}

public void efetuarBaixaEstoque(Medicamento medic, int qtd) {

// CÓDIGO

}

public void emitirNotaFiscal(Venda venda) {

// CÓDIGO

}

}

public class Cliente\_Especial {

private String nome, rg, endereco;

private int telefone;

private long cpf;

private Date dataNascimento;

private List<Venda> vendas;

// Getters e Setters//

}

public class Medicamento {

private String nome, fabricante;

private Date dataValidade;

private BigDecimal preco;

private int qtdEstoque;

// Getters e Setters //

}

public class Venda {

private List<Medicamento> medicamento;

private List<int> quantidade;

private Cliente\_Especial cliente;

private double porcentagemDesconto;

private BigDecimal valor;

private Tipo\_Pagamento tipoPagamento;

// Getters e Setters//

}

public class Caixa {

private String senhaAbertura, senhaFechamento;

private Notas notas;

private Moedas moedas;

// Getters e Setters//

}

public class Notas {

private int qtdDoisReais, qtdCincoReais, qtdDezReais,

qtdVinteReais, qtdCinquentaReais, qtdCemReais;

// Getters e Setters//

}

public class Moedas {

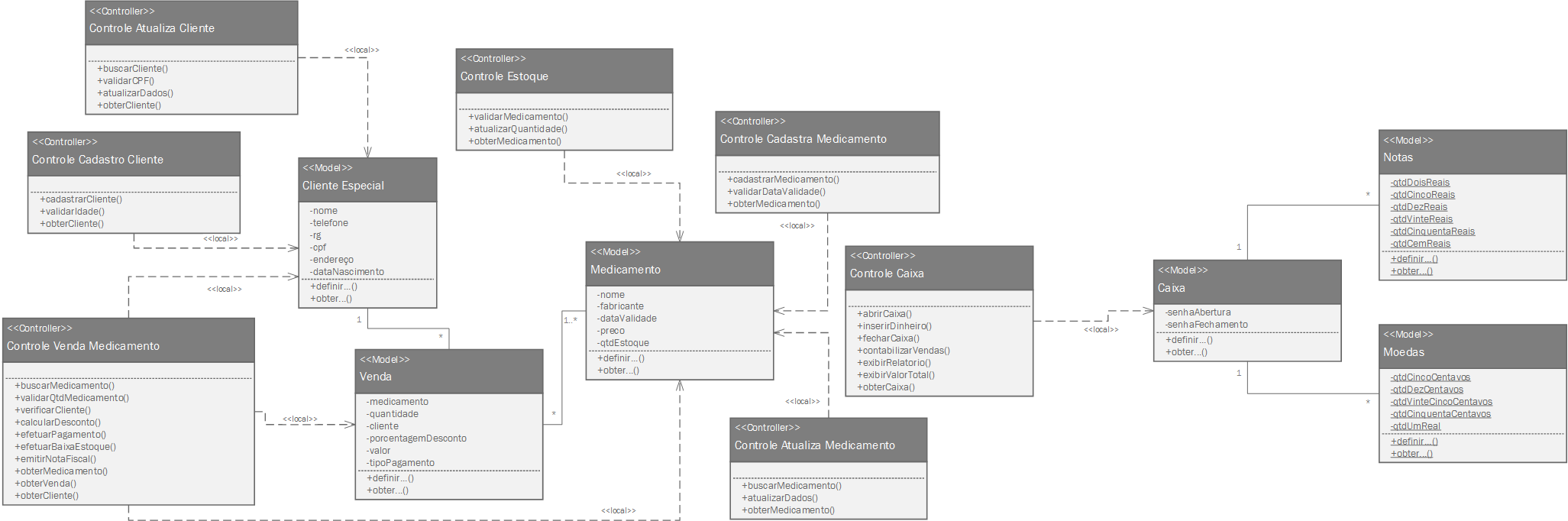
private int qtdCincoCentavos, qtdDezCentavos,

qtdVinteCincoCentavos, qtdCinquentaCentavos, qtdUmReal;

// Getters e Setters//

}

**10 – Transforme todos os relacionamentos de associação entre as classes de controle e modelo para dependências não estruturais por variável local. Explique a vantagem e desvantagem desse tipo de dependência.**



Este tipo de dependência permite um menor acoplamento e maior encapsulamento. Em pontos negativos, o desempenho em tempo de execução é menor.

**11 – Apresente a estrutura básica de código em JAVA, C# ou C++ para implementar as dependências não estruturais por variável local.**

public class Controle\_Atualiza\_Cliente {

public Cliente\_Especial buscarCliente() {

Cliente\_Especial cliente = ObterCliente();

// CÓDIGO

}

public boolean validarCPF() {

Cliente\_Especial cliente = ObterCliente();

// CÓDIGO

}

public void atualizaDados() {

Cliente\_Especial cliente = ObterCliente();

// CÓDIGO

}

public Cliente\_Especial obterCliente() {

Cliente\_Especial cliente = new Cliente\_Especial();

// Código para obter dados do cliente

return cliente;

}

}

public class Controle\_Cadastro\_Cliente {

public boolean validarIdade() {

Cliente\_Especial cliente = ObterCliente();

// CÓDIGO

}

public void cadastrarCliente() {

Cliente\_Especial cliente = ObterCliente();

// CÓDIGO

}

public Cliente\_Especial obterCliente() {

Cliente\_Especial cliente = new Cliente\_Especial();

// Código para obter dados do cliente

return cliente;

}

}

public class Controle\_Estoque {

public boolean validarMedicamento() {

Medicamento medic = ObterMedicamento();;

// CÓDIGO

}

public void atualizarQuantidade() {

Medicamento medic = ObterMedicamento();;

// CÓDIGO

}

public Medicamento obterMedicamento() {

Medicamento medic = new Medicamento();

// Código para obter dados do medicamento

return medic;

}

}

public class Controle\_Cadastro\_Medicamento {

public void cadastrarMedicamento() {

Medicamento medic = ObterMedicamento();;

// CÓDIGO

}

public boolean validarDadaValidade() {

Medicamento medic = ObterMedicamento();;

// CÓDIGO

}

public Medicamento obterMedicamento() {

Medicamento medic = new Medicamento();

// Código para obter dados do medicamento

return medic;

}

}

public class Controle\_Caixa {

public Caixa abrirCaixa() {

Caixa caixa = ObterCaixa();;

// CÓDIGO

}

public void inserirDinheiro() {

Caixa caixa = ObterCaixa();;

// CÓDIGO

}

public void fecharCaixa() {

Caixa caixa = ObterCaixa();;

// CÓDIGO

}

public double contabilizarVendas() {

Caixa caixa = ObterCaixa();;

// CÓDIGO

}

public void exibirRelatorio() {

Caixa caixa = ObterCaixa();;

// CÓDIGO

}

public void exibirValorTotal() {

Caixa caixa = ObterCaixa();;

// CÓDIGO

}

public Caixa obterCaixa() {

Caixa caixa = new Caixa();

// Código para obter dados do caixa

return caixa;

}

}

public class Controle\_Atualiza\_Medicamento {

public Medicamento buscarMedicamento() {

Medicamento medic = ObterMedicamento();;

// CÓDIGO

}

public void atualizaDados() {

Medicamento medic = ObterMedicamento();;

// CÓDIGO

}

public Medicamento obterMedicamento() {

Medicamento medic = new Medicamento();

// Código para obter dados do medicamento

return medic;

}

}

public class Controle\_Venda\_Medicamento {

public Medicamento buscarMedicamento() {

Medicamento medic = ObterMedicamento();;

// CÓDIGO

}

public boolean validarQtdMedicamento() {

Venda venda = ObterVenda();;

// CÓDIGO

}

public boolean verificarCliente() {

Cliente\_Especial cliente = ObterCliente();

// CÓDIGO

}

public double calcularDesconto() {

Venda venda = ObterVenda();;

// CÓDIGO

}

public void efetuarPagamento() {

Venda venda = ObterVenda();;

// CÓDIGO

}

public void efetuarBaixaEstoque() {

Medicamento medic = ObterMedicamento();;

// CÓDIGO

}

public void emitirNotaFiscal() {

Venda venda = ObterVenda();;

// CÓDIGO

}

public Cliente\_Especial obterCliente() {

Cliente\_Especial cliente = new Cliente\_Especial();

// Código para obter dados do cliente

return cliente;

}

public Medicamento obterMedicamento() {

Medicamento medic = new Medicamento();

// Código para obter dados do medicamento

return medic;

}

public Caixa obterCaixa() {

Caixa caixa = new Caixa();

// Código para obter dados do caixa

return caixa;

}

}

public class Cliente\_Especial {

private String nome, rg, endereco;

private int telefone;

private long cpf;

private Date dataNascimento;

private List<Venda> vendas;

// Getters e Setters//

}

public class Medicamento {

private String nome, fabricante;

private Date dataValidade;

private BigDecimal preco;

private int qtdEstoque;

// Getters e Setters //

}

public class Venda {

private List<Medicamento> medicamento;

private List<int> quantidade;

private Cliente\_Especial cliente;

private double porcentagemDesconto;

private BigDecimal valor;

private Tipo\_Pagamento tipoPagamento;

// Getters e Setters//

}

public class Caixa {

private String senhaAbertura, senhaFechamento;

private Notas notas;

private Moedas moedas;

// Getters e Setters//

}

public class Notas {

private int qtdDoisReais, qtdCincoReais, qtdDezReais,

qtdVinteReais, qtdCinquentaReais, qtdCemReais;

// Getters e Setters//

}

public class Moedas {

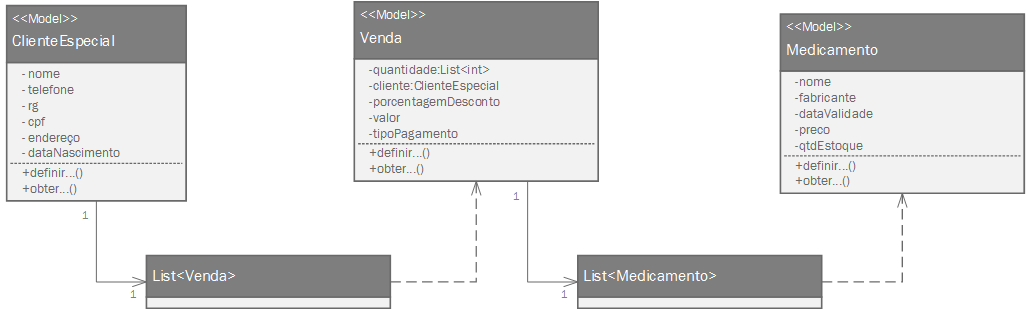
private int qtdCincoCentavos, qtdDezCentavos,

qtdVinteCincoCentavos, qtdCinquentaCentavos, qtdUmReal;

// Getters e Setters//

}

**12 – Modele as classes parametrizadas com a estrutura <List> para resolver o lado muitos dos relacionamentos. Por que tais classes foram modeladas?**



**13 – Apresente a estrutura básica de código em JAVA, C# ou C++ para implementar as classes parametrizadas com a estrutura <List>.**

public class Cliente\_Especial {

private String nome, rg, endereco;

private int telefone;

private long cpf;

private Date dataNascimento;

private List<Venda> vendas;

// Getters e Setters//

}

public class Medicamento {

private String nome, fabricante;

private Date dataValidade;

private BigDecimal preco;

private int qtdEstoque;

// Getters e Setters //

}

public class Venda {

private List<Medicamento> medicamento;

private List<int> quantidade;

private Cliente\_Especial cliente;

private double porcentagemDesconto;

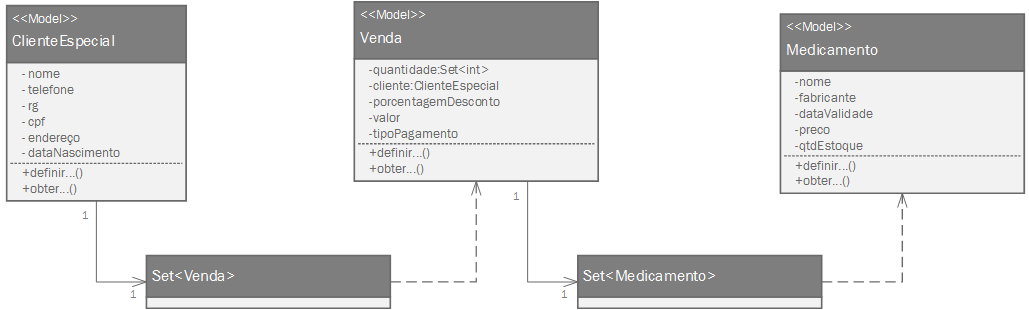
private BigDecimal valor;

private Tipo\_Pagamento tipoPagamento;

// Getters e Setters//

}

**14 - Modele as classes parametrizadas com a estrutura <Set> para resolver o lado muitos dos relacionamentos. Por que tais classes foram modeladas?**



**15 – Apresente a estrutura básica de código em JAVA, C# ou C++ para implementar as classes parametrizadas com a estrutura <Set>.**

public class Cliente\_Especial {

private String nome, rg, endereco;

private int telefone;

private long cpf;

private Date dataNascimento;

private Set<Venda> vendas;

// Getters e Setters//

}

public class Medicamento {

private String nome, fabricante;

private Date dataValidade;

private BigDecimal preco;

private int qtdEstoque;

// Getters e Setters //

}

public class Venda {

private Set<Medicamento> medicamento;

private Set<int> quantidade;

private Cliente\_Especial cliente;

private double porcentagemDesconto;

private BigDecimal valor;

private Tipo\_Pagamento tipoPagamento;

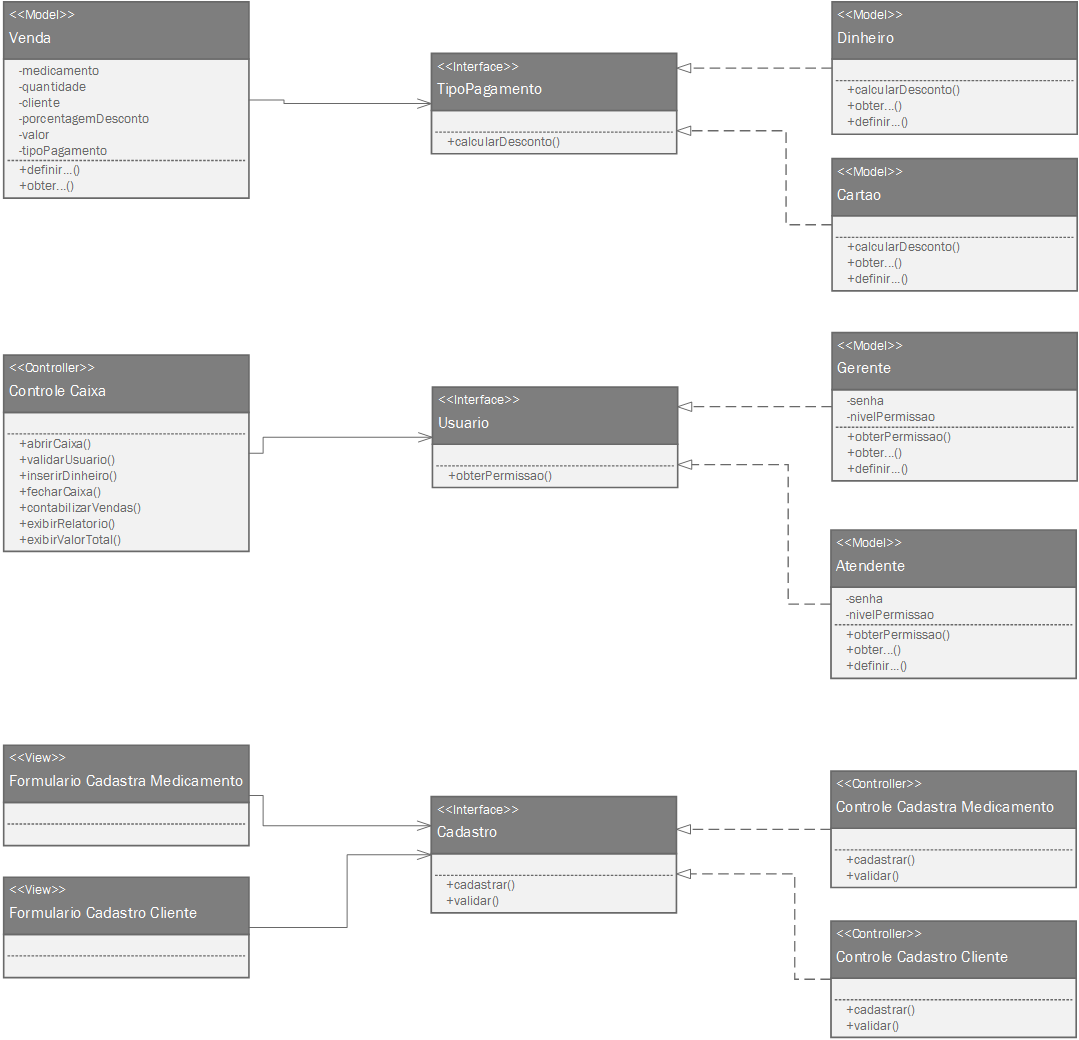
// Getters e Setters//

}

**16 – Qual a diferença entre classe parametrizada, multiobjetos e collection?**

Classe parametrizada é uma classe utilizada para descrever outras classes. Uma collection é uma coleção de classes parametrizadas. “Multiobjects” é o nome dado pela UML para uma coleção de objetos de uma mesma classe.

**17 – Modele três interfaces estabelecendo o devido contrato de comportamento entre as classes consumidoras e fornecedoras e declarando as operações nas interfaces a serem implementadas pelas classes fornecedoras. Justifique a razão de existência de cada uma das interfaces.**



**Interface TipoPagamento:** implementação de calcularDesconto() difere entre as classes de Dinheiro e Cartão.

**Interface Usuário:** uma interface entre as classes possibilita diferentes níveis de segurança no uso da senha.

**Interface Cadastro:** as classes de controle para cadastro de cliente e medicamento tem funcionamento similar, porém implementação distinta. Uma interface entre elas garantiria melhor organização do código, não forçaria relação entre elas e também garantiria reuso das assinaturas caso no futuro seja necessário implementar novas classes de cadastro.

**18 – Apresente a estrutura básica de código em JAVA, C# ou C++ para implementar as relações de interface.**

public interface TipoPagamento {

public BigDecimal calcularDesconto();

}

public interface Usuario {

public boolean obterPermissao();

}

public interface Cadastro {

public void cadastrar();

public boolean validar();

}

public class Venda {

private List<Medicamento> medicamento;

private List<int> quantidade;

private Cliente\_Especial cliente;

private double porcentagemDesconto;

private BigDecimal valor;

private TipoPagamento tipoPagamento;

// Getters e Setters//

}

public class Dinheiro implements TipoPagamento {

public BigDecimal calcularDesconto() {

}

// Getters e Setters//

}

public class Cartao implements TipoPagamento {

public BigDecimal calcularDesconto() {

}

// Getters e Setters//

}

public class Gerente implements Usuario {

String senha;

int nivelPermissao;

public boolean obterPermissao() {

}

// Getters e Setters//

}

public class Atendente {

String senha;

int nivelPermissao;

public boolean obterPermissao() {

}

// Getters e Setters//

}

public class Controle\_Caixa {

Usuario usuario;

public Caixa abrirCaixa(Caixa caixa) {

// CÓDIGO

}

public boolean validarUsuario() {

// CÓDIGO

}

public void inserirDinheiro(Caixa caixa) {

// CÓDIGO

}

public void fecharCaixa(Caixa caixa) {

// CÓDIGO

}

public double contabilizarVendas(Caixa caixa) {

// CÓDIGO

}

public void exibirRelatorio(Caixa caixa) {

// CÓDIGO

}

public void exibirValorTotal(Caixa caixa) {

// CÓDIGO

}

}

public class Controle\_Cadastro\_Medicamento implements Cadastro {

public void cadastrar() {

// CÓDIGO

}

public boolean validar() {

// CÓDIGO

}

}

public class Controle\_Cadastro\_Cliente implements Cadastro {

public boolean validar() {

// CÓDIGO

}

public void cadastrar() {

// CÓDIGO

}

}

public class Formulario\_CadastraMedicamento {

private Cadastro cadastro;

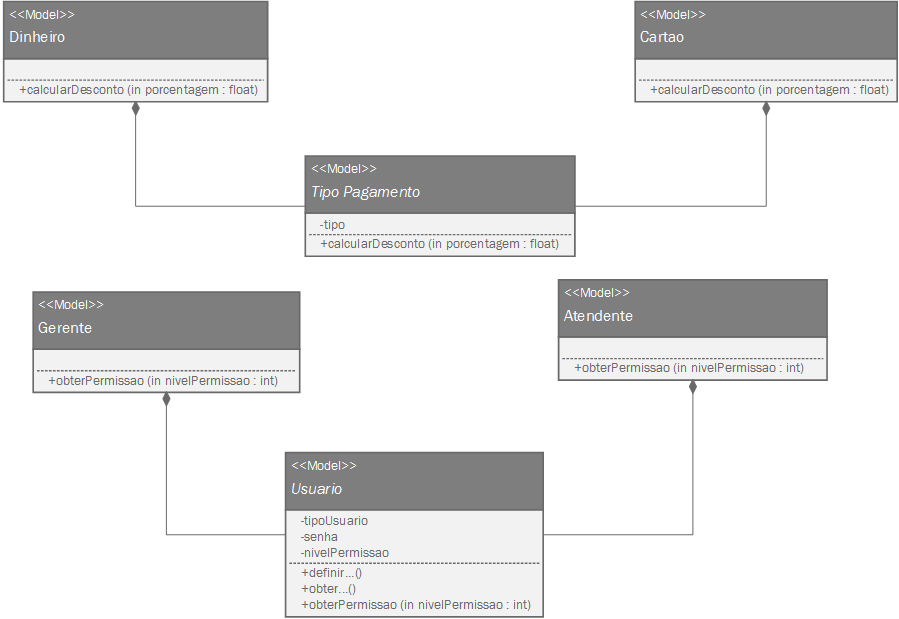
}

public class Formulario\_CadastroCliente {

private Cadastro cadastro;

}

**19 – Modele duas relações de delegação, utilizando classes diferentes para cada uma. Justifique a razão de existência de cada uma das relações de delegação.**



**20 – Apresente a estrutura básica de código em JAVA, C# ou C++ para implementar as relações de delegação.**

public class Gerente implements Usuario {

String senha;

int nivelPermissao;

Usuario usuario;

public boolean obterPermissao() {

return usuario.obterPermissao(nivelPermissao);

}

// Getters e Setters//

}

public class Atendente {

String senha;

int nivelPermissao;

Usuario usuario;

public boolean obterPermissao() {

return usuario.obterPermissao(nivelPermissao);

}

// Getters e Setters//

}

public class Dinheiro implements TipoPagamento {

private TipoPagamento tipoPagamento;

public BigDecimal calcularDesconto() {

tipoPagamento.calcularDesconto(0.05f);

}

// Getters e Setters//

}

public class Cartao implements TipoPagamento {

private TipoPagamento tipoPagamento;

public BigDecimal calcularDesconto() {

tipoPagamento.calcularDesconto(0f);

}

// Getters e Setters//

}

public class TipoPagamento {

public BigDecimal calcularDesconto(float porcentagem) {

// CÓDIGO

}

}

public class Usuario {

public boolean obterPermissao(int nivelPermissao) {

// CÓDIGO

}

}

**21 - Faça um quadro comparativo entre generalização, realização e delegação, apresentando no mínimo duas vantagens e duas desvantagens para cada um desses conceitos.**

|  |  |  |  |
| --- | --- | --- | --- |
| Reuso | Generalização | Delegação | Realização |
| Vantagens | * Fácil de implementar, utilizando conceito de herança simples. * Superclasse consegue esconder seus dados, não permitindo que os mesmos sejam alterados pelas classes derivadas. | * Permite que um objeto reutilize o comportamento de outro sem necessariamente ter que ser subtipo deste. * Pode ser realizado em tempo de execução. | * Melhor coesão e menor acoplamento da classe. * Captura semelhanças entre classes sem forçar relação entre elas. |
| Desvantagens | * Expõe detalhes da superclasse as subclasses, violando princípio do encapsulamento. * Possível violação do Princípio de Liskov, ou Regra da Substituição. | * Diminui desempenho, pois implica cruzar a fronteira de outro objeto para realização de uma função. * Não pode ser utilizado em situação em que uma classe parcialmente abstrata está envolvida. | * Não reutiliza código. * Interfaces devem ser utilizadas várias vezes, caso contrário não existe motivo delas serem utilizadas. |