

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

**Lista 3 – 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 o diagrama de classes de projeto refinado com as seguintes notações:**

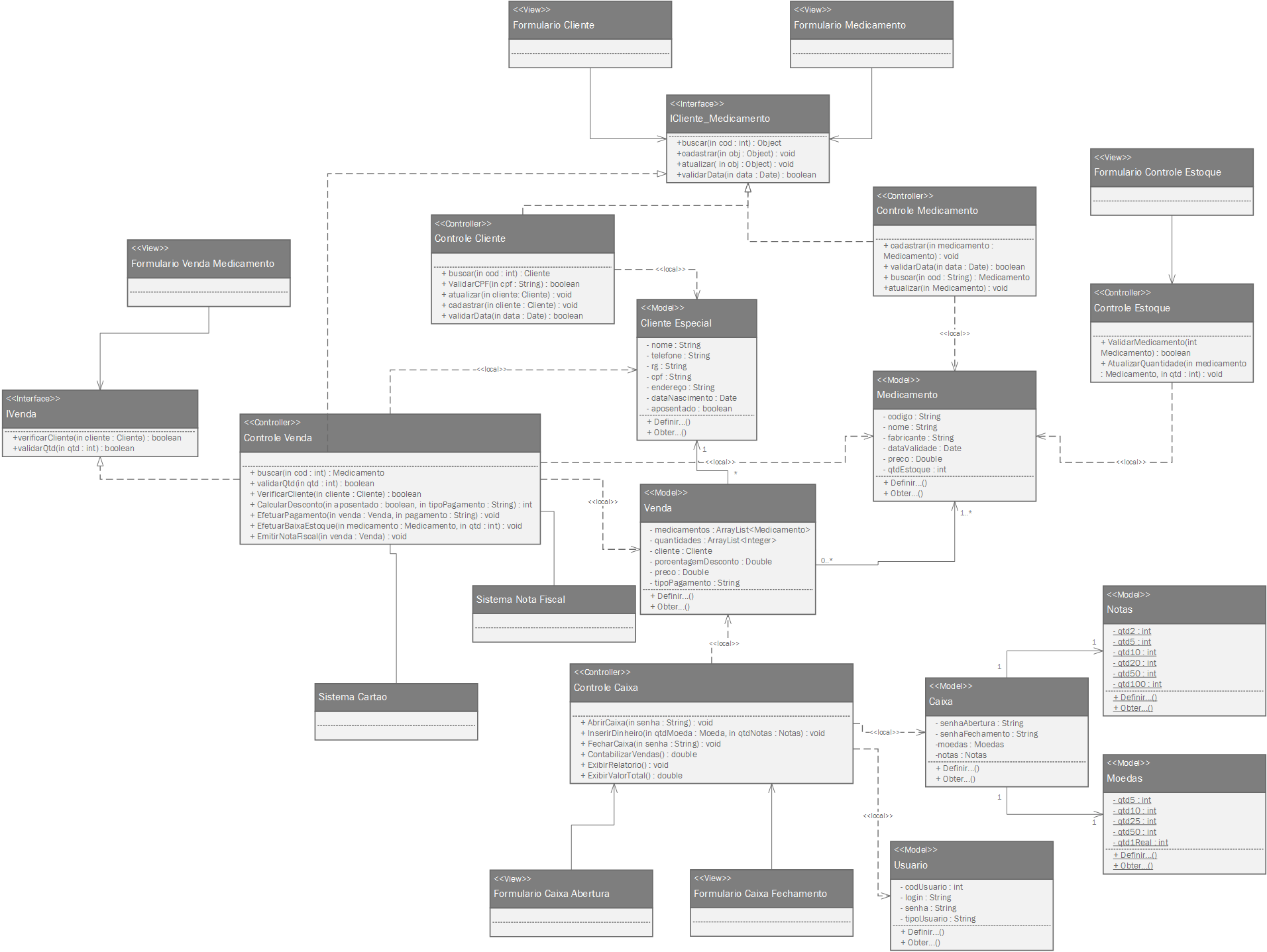
a. Dependências estruturais entre as classes de modelo;

b. Dependências não estruturais por parâmetro ou variável local entre as classes de controle e modelo;

c. Dependências estruturais entre as classes de visão e controle;

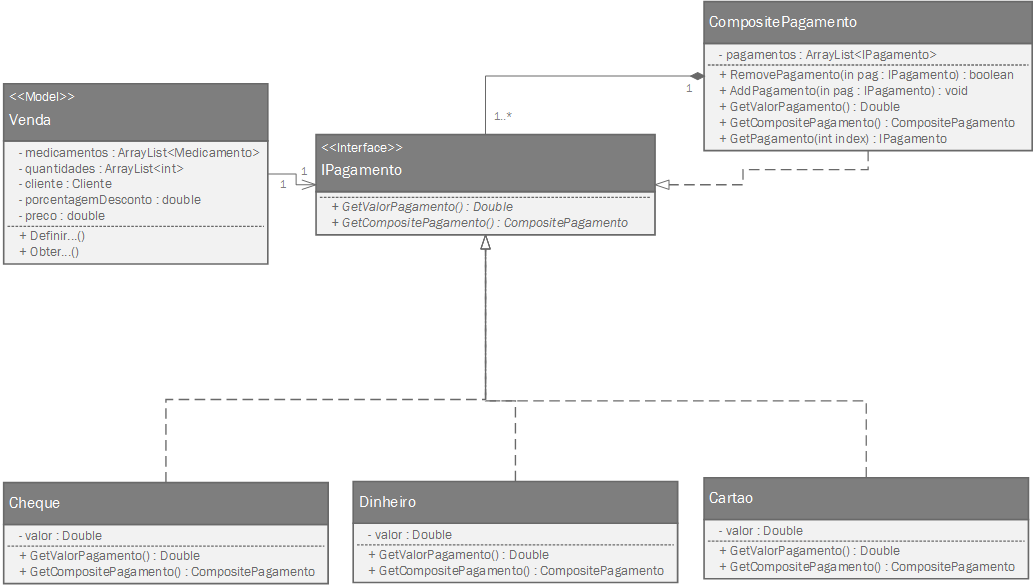
d. Classes parametrizadas com a estrutura <Set> ou <List> para resolver o lado muitos dos relacionamentos;

e. Duas interfaces estabelecendo o devido contrato de comportamento entre as classes consumidoras e fornecedoras.



**Parte B**

**2- Com base no diagrama de classes de projeto refinado nesta lista, modele o padrão de projeto Composite. Qual o propósito desse padrão no diagrama?**



O padrão Composite é implementado para que seja possível o uso de várias formas de pagamento diferentes em uma mesma venda, através da composição recursiva dos tipos de pagamento.

**3- Apresente a estrutura básica de código em JAVA, C# ou C++ para implementar o padrão de projeto Composite.**

public class Venda {

private ArrayList<Medicamento> medicamentos;

private ArrayList<int> quantidades;

private Cliente cliente;

private double porcentagemDesconto;

private double preco;

private IPagamento pagamento;

public get...() { ... }

public set...() { ... }

}

public interface IPagamento {

public double GetValorPagamento();

public CompositePagamento GetCompositePagamento();

}

public class CompositePagamento implements IPagamento {

private ArrayList<IPagamento> pagamentos;

public boolean RemovePagamento(IPagamento pag) {

pagamentos.remove(pag);

}

public void AddPagamento(IPagamento pag) {

pagamentos.add(pag);

}

public double GetValorPagamento() {

double valor = 0;

for (IPagamento pag : pagamentos)

valor += pag.GetValorPagamento();

return valor;

}

public CompositePagamento GetCompositePagamento() {

return this;

}

public IPagamento GetPagamento(int index) {

// realiza percurso em pré-ordem e retorna o IPagamento no índice indicado

}

}

public class Cheque implements IPagamento {

private double valor;

public double GetValorPagamento() {

return valor;

}

public CompositePagamento GetCompositePagamento() {

return null;

}

}

public class Dinheiro implements IPagamento {

private double valor;

public double GetValorPagamento() {

return valor;

}

public CompositePagamento GetCompositePagamento() {

return null;

}

}

public class Cartao implements IPagamento {

private double valor;

public double GetValorPagamento() {

return valor;

}

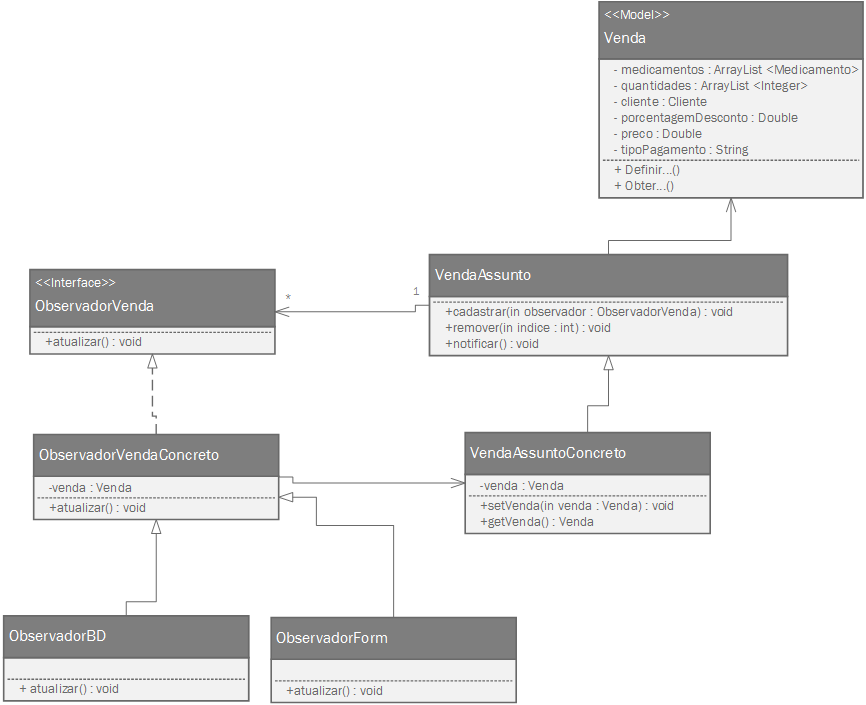
public CompositePagamento GetCompositePagamento() {

return null;

}

}

**4- Com base no diagrama de classes de projeto refinado nesta lista, modele o padrão de projeto Observer. Qual o propósito desse padrão no diagrama?**



Nesse diagrama o Observer tem como propósito manter todos os dependentes de venda notificados e atualizados.

**5- Apresente a estrutura básica de código em JAVA, C# ou C++ para implementar o padrão de projeto Observer.**

public class Venda {

private ArrayList<Medicamento> medicamentos;

private ArrayList<int> quantidades;

private Cliente cliente;

private double porcentagemDesconto;

private double preco;

private String tipoPagamento;

public get...() { ... }

public set...() { ... }

}

public class VendaAssunto {

private ArrayList<ObservadorVenda> observadores;

private Venda venda;

public void cadastrar(ObservadorVenda observador) {

observadores.add(observador);

}

public void remover(int indice) {

observadores.remove(indice);

}

public void notificar() {

for (ObservadorVenda obs : observadores)

obs.atualizar();

}

}

public class VendaAssuntoConcreto extends VendaAssunto {

private Venda venda;

public Venda getVenda() { return venda; }

public void setVenda(Venda venda) { this.venda = venda; }

}

public interface ObservadorVenda {

public void atualizar();

}

public class ObservadorVendaConcreto implements ObservadorVenda {

private Venda venda;

private VendaAssuntoConcreto vac;

@Override

public void atualizar() {

venda = vac.getVenda();

}

}

public class ObservadorBD extends ObservadorVendaConcreto {

@Override

public void atualizar() {

// atualiza o BD

}

}

public class ObservadorForm extends ObservadorVendaConcreto {

@Override

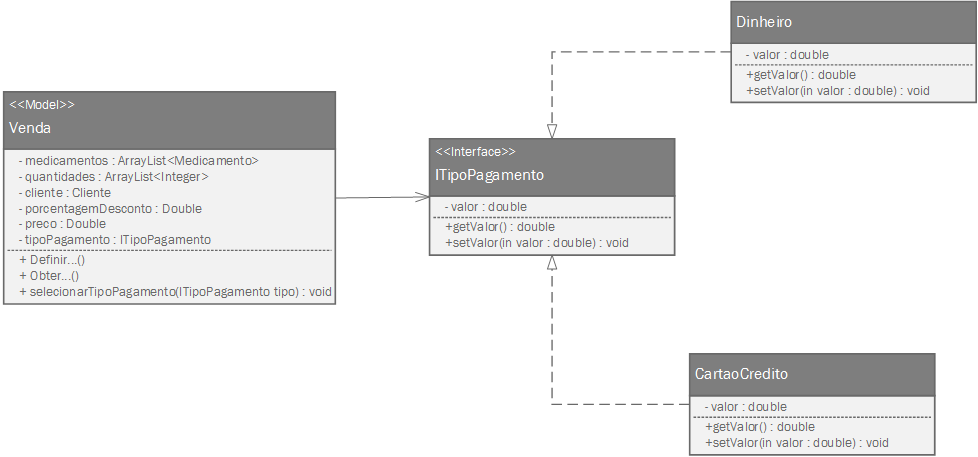
public void atualizar() {

// atualiza o form

}

}

**6- Com base no diagrama de classes de projeto refinado nesta lista, modele o padrão de projeto Strategy. Qual o propósito desse padrão no diagrama?**



Permitir os vários comportamentos possiveis do método selecionarTipoPagamento()

**7- Apresente a estrutura básica de código em JAVA, C# ou C++ para implementar o padrão de projeto Strategy.**

public class Venda {

private ArrayList<Medicamento> medicamentos;

private ArrayList<int> quantidades;

private Cliente cliente;

private double porcentagemDesconto;

private double preco;

private ITipoPagamento tipoPagamento;

public get...() { ... }

public set...() { ... }

public void selecionarTipoPagamento(ITipoPagamento tipo) {

tipoPagamento = tipo;

}

}

public interface ITipoPagamento {

public void selecionarTipoPagamento(ITipoPagamento tipo);

}

public class Dinheiro implements ITipoPagamento {

private double valor;

public double getValor() { return valor; }

public void setValor(double valor) { this.valor = valor; }

}

public class CartaoCredito implements ITipoPagamento {

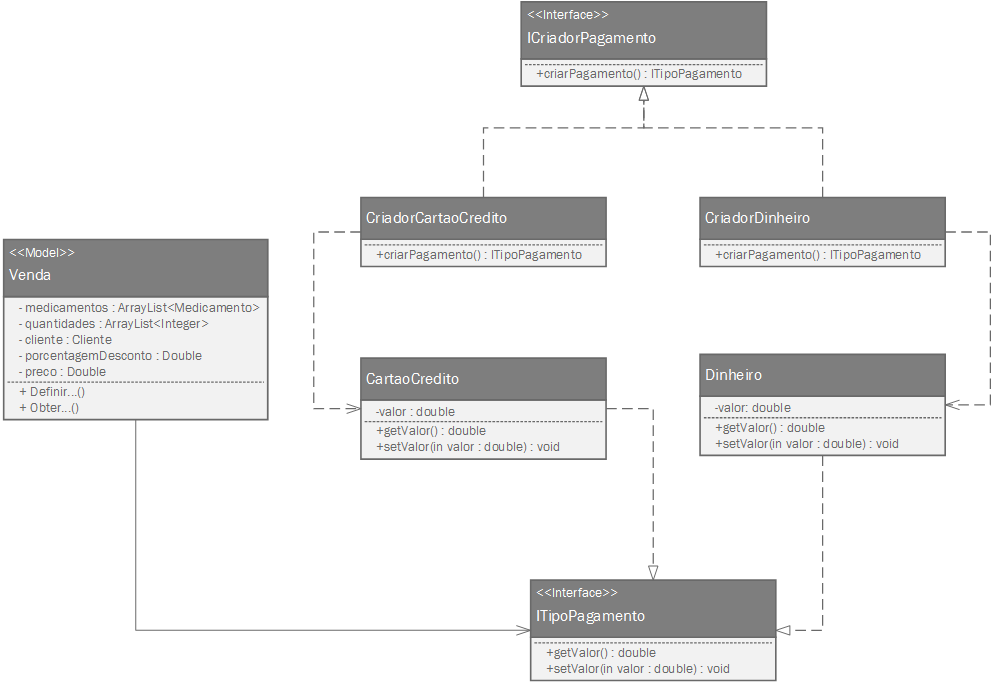
private double valor;

public double getValor() { return valor; }

public void setValor(double valor) { this.valor = valor; }

}

**8- Com base no diagrama de classes de projeto refinado nesta lista, modele o padrão de projeto Factory Method. Qual o propósito desse padrão no diagrama?**



O padrão Factory Method foi implementado para que a instanciação do tipo de pagamento de uma venda seja repassada para uma subclasse, reduzindo o acoplamento.

**9- Apresente a estrutura básica de código em JAVA, C# ou C++ para implementar o padrão de projeto Factory Method**

public class Venda {

private ArrayList<Medicamento> medicamentos;

private ArrayList<int> quantidades;

private Cliente cliente;

private double porcentagemDesconto;

private double preco;

private ITipoPagamento tipoPagamento;

public get...() { ... }

public set...() { ... }

}

public interface ICriadorPagamento {

public ITipoPagamento criarPagamento();

}

public class CriadorCartaoCredito implements ICriadorPagamento {

public ITipoPagamento criarPagamento() {

return new CartaoCredito();

}

}

public class CriadorDinheiro implements ICriadorPagamento {

public ITipoPagamento criarPagamento() {

return new Dinheiro();

}

}

public interface ITipoPagamento {

public double getValor();

public void setValor(double valor);

}

public class CartaoCredito implements ITipoPagamento {

private double valor;

public double getValor() { return valor \* 0.95; }

public void setValor(double valor) { this.valor = valor; }

}

public class Dinheiro implements ITipoPagamento {

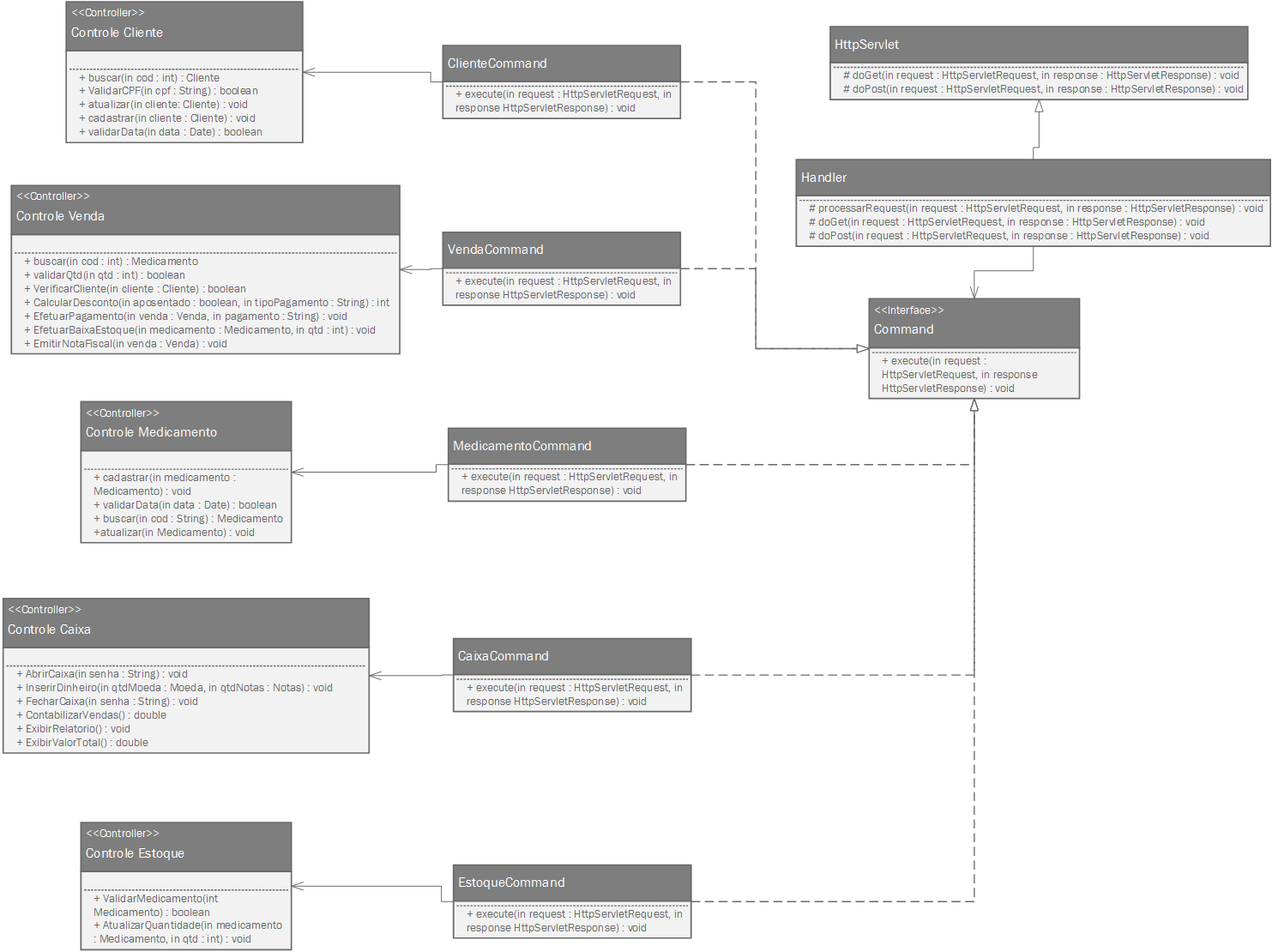
private double valor;

public double getValor() { return valor; }

public void setValor(double valor) { this.valor = valor; }

}

**10- Com base no diagrama de classes de projeto refinado nesta lista, modele o padrão de projeto Front Controller. Qual o propósito desse padrão no diagrama?**



O padrão Front Controller foi implementado para tratar as solicitações feitas ao sistema web através de um controlador único, que fará a delegação das requisições para outro controlador e método mais adequado.

**11- Apresente a estrutura básica de código em JAVA, C# ou C++ para implementar o padrão de projeto Front Controller.**

public class Handler extends HttpServlet {

protected void processarRequest(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

Command comando = null;

try {

comando = (Command)Class.forName("commands."+request.getParameter("command")).newInstance();

} catch (InstantiationException e) {

e.printStackTrace();

} catch (IllegalAccessException e) {

e.printStackTrace();

} catch (ClassNotFoundException e) {

e.printStackTrace();

}

comando.execute(request, response);

}

@Override

protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

processRequest(request, response);

}

@Override

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

processRequest(request, response);

}

}

public interface Command {

public void execute(HttpServletRequest request, HttpServletResponse response);

}

public class ClienteCommand implements Command {

ControleCliente control = new ControleCliente();

@Override

public void execute(HttpServletRequest request, HttpServletResponse response) {

String metodo = request.getParameter("metodo");

switch (metodo) {

case "buscar":

response.getWriter().write(control.buscar((int)request.getParameter("cod")));

case "validarCPF":

response.getWriter().write(control.validarCPF(request.getParameter("cpf")));

case "atualizar":

control.atualizar((Cliente)request.getParameter("cliente"));

case "cadastrar":

control.cadastrar((Cliente)request.getParameter("cliente"));

case "validarData":

response.getWriter().write(control.validarData((Date)request.getParameter("data")));

}

}

}

public class VendaCommand implements Command {

ControleVenda control = new ControleVenda();

@Override

public void execute(HttpServletRequest request, HttpServletResponse response) {

// processa a request...

}

}

public class MedicamentoCommand implements Command {

MedicamentoControle control = new MedicamentoControle();

@Override

public void execute(HttpServletRequest request, HttpServletResponse response) {

// processa a request...

}

}

public class CaixaCommand implements Command {

CaixaControle control = new CaixaControle();

@Override

public void execute(HttpServletRequest request, HttpServletResponse response) {

// processa a request...

}

}

public class EstoqueCommand implements Command {

EstoqueControle control = new EstoqueControle();

@Override

public void execute(HttpServletRequest request, HttpServletResponse response) {

// processa a request...

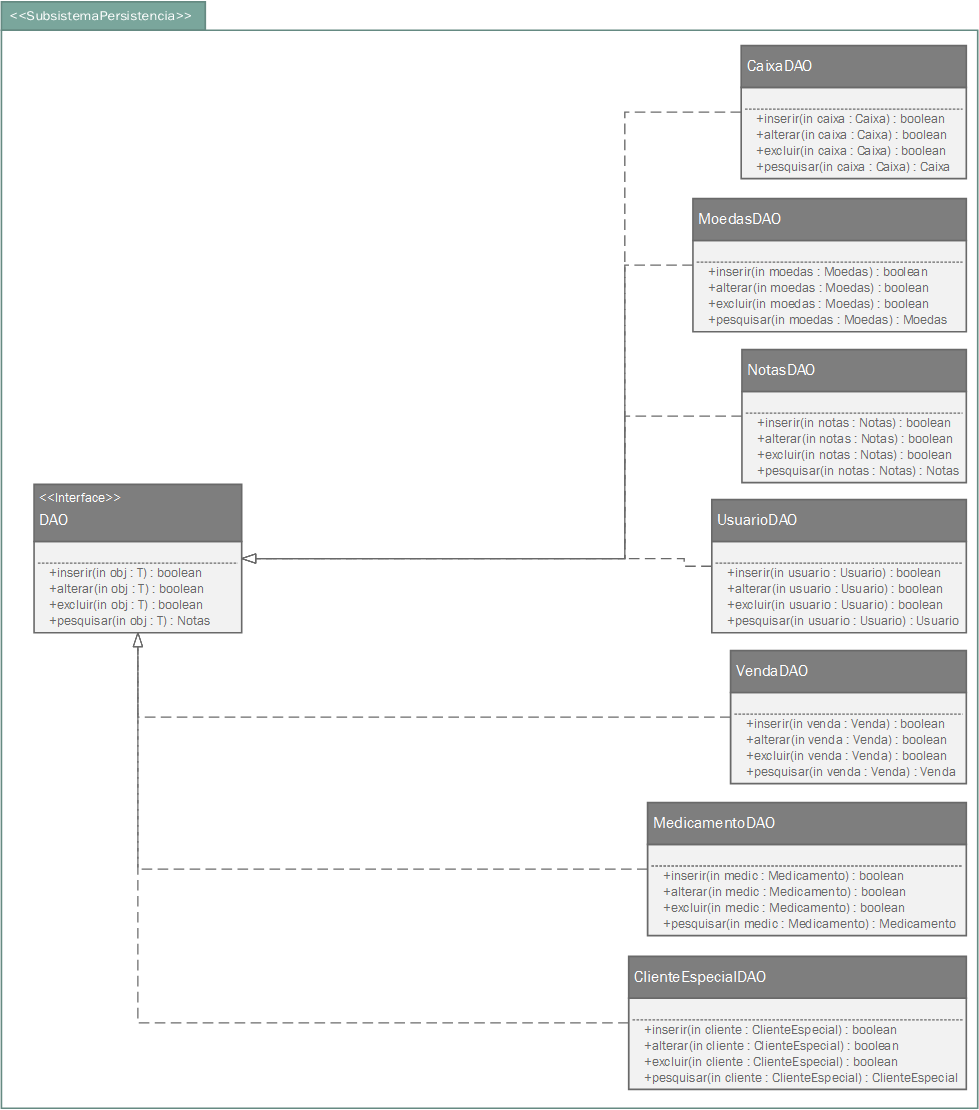
}

}

**12- Com base no diagrama de classes de projeto refinado nesta lista,, modele os pacotes (subsistemas) e faça a alocação das classes em cada pacote. Cada pacote deve mostrar as classes detalhadas com atributos e métodos. Neste exercício, deve constar um pacote de classes de visão, um pacote de classes de controle, no mínimo três pacotes de classes de modelo e um pacote de classes enumeradas.**



**13- Construa o pacote de Persistência e faça a alocação das classes DAO no pacote. Este pacote deve mostrar as classes detalhadas com métodos**



**14- Apresente a estrutura básica de código para implementar o pacote de Persistência (DAO).**

package Persistencia

import java.util.list;

public interface DAO<T> {

public boolean inserir(T object);

public boolean alterar(T object);

public boolean excluir(T object);

public T pesquisar(T object);

public List<T> listar(String filtro);

}

public class ClienteEspecialDAO implements DAO<ClienteEspecial> {

@Override

public boolean inserir(ClienteEspecial cliente) {

// Código

}

@Override

public boolean alterar(ClienteEspecial cliente) {

// Código

}

@Override

public boolean excluir(ClienteEspecial cliente) {

// Código

}

@Override

public ClienteEspecial pesquisar(CLienteEspecial cliente) {

// Código

}

}

public class MedicamentoDAO implements DAO<Medicamento> {

@Override

public boolean inserir(Medicamento medic) {

// Código

}

@Override

public boolean alterar(Medicamento medic) {

// Código

}

@Override

public boolean excluir(Medicamento medic) {

// Código

}

@Override

public Medicamento pesquisar(Medicamento medic) {

// Código

}

}

public class VendaDAO implements DAO<Venda> {

@Override

public boolean inserir(Venda venda) {

// Código

}

@Override

public boolean alterar(Venda venda) {

// Código

}

@Override

public boolean excluir(Venda venda) {

// Código

}

@Override

public Venda pesquisar(Venda venda) {

// Código

}

}

public class UsuarioDAO implements DAO<Usuario> {

public boolean inserir(Usuario usuario) {

// Código

}

@Override

public boolean alterar(Usuario usuario) {

// Código

}

@Override

public boolean excluir(Usuario usuario) {

// Código

}

@Override

public Usuario pesquisar(Usuario usuario) {

// Código

}

}

public class CaixaDAO implements DAO<Caixa> {

public boolean inserir(Caixa caixa) {

// Código

}

@Override

public boolean alterar(Caixa caixa) {

// Código

}

@Override

public boolean excluir(Caixa caixa) {

// Código

}

@Override

public Caixa pesquisar(Caixa caixa) {

// Código

}

}

public class MoedasDAO implements DAO<Moedas> {

public boolean inserir(Moedas moedas) {

// Código

}

@Override

public boolean alterar(Moedas moedas) {

// Código

}

@Override

public boolean excluir(Moedas moedas) {

// Código

}

@Override

public Moedas pesquisar(Moedas moedas) {

// Código

}

}

public class NotasDAO implements DAO<Notas> {

@Override

public boolean inserir(Notas notas) {

// Código

}

@Override

public boolean alterar(Notas notas) {

// Código

}

@Override

public boolean excluir(Notas notas) {

// Código

}

@Override

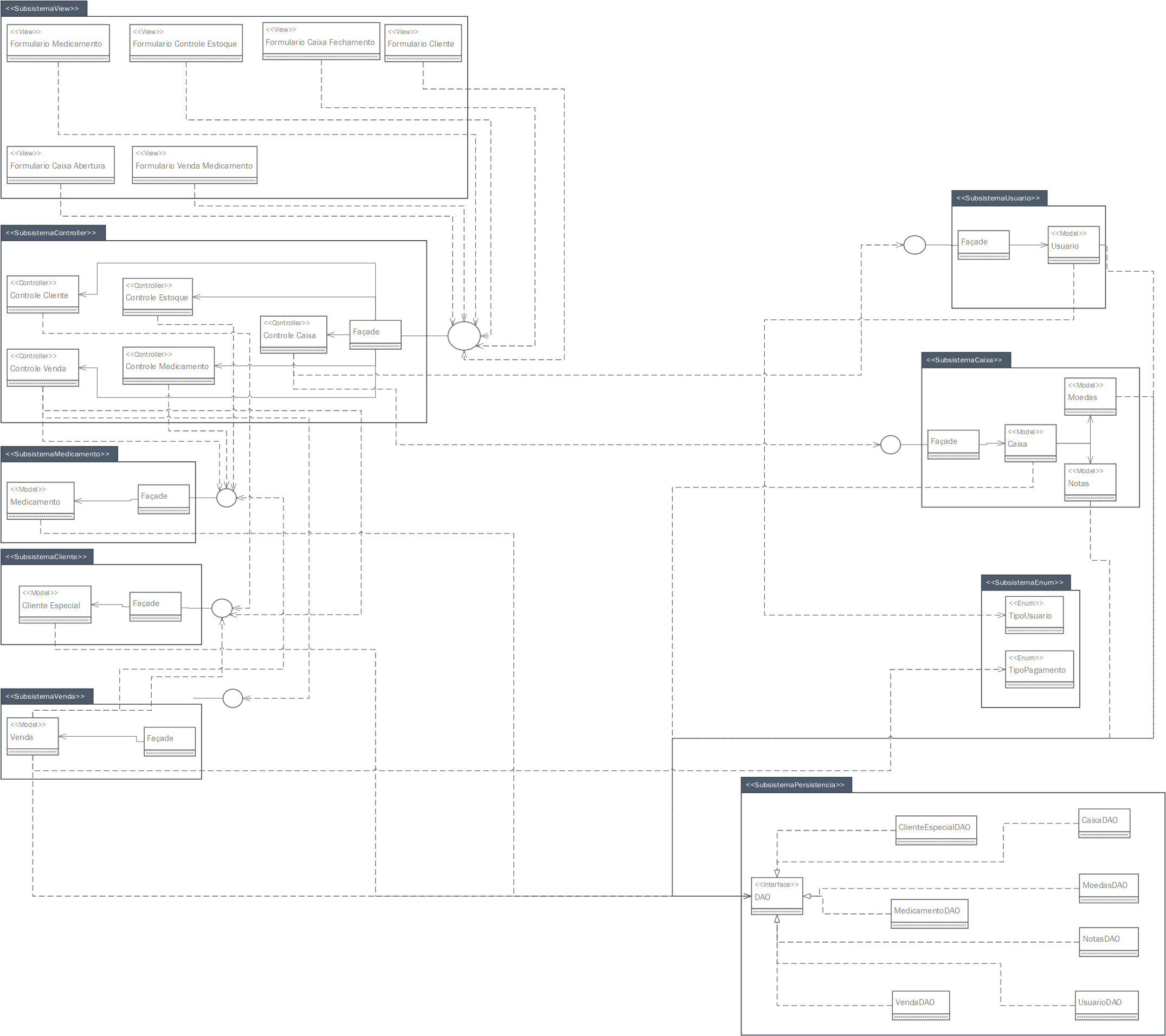
public Notas pesquisar(Notas notas) {

// Código

}

}

**15- Após a identificação dos pacotes (subsistemas) e alocação das classes, modele um diagrama de pacotes com os devidos relacionamentos, aplicando o padrão de projeto Façade no pacote de controle e nos pacotes de modelo. Neste diagrama, os pacotes devem mostrar somente os nomes das classes, sem a necessidade de apresentar os detalhes (atributos e/ou métodos) das classes.**



**16- Apresente a estrutura básica de código em JAVA, C# ou C++ para implementar o padrão de projeto Façade.**

// Pacote Enum

package Enum;

public enum TipoPagamento {

// Code

}

public enum TipoUsuario {

// Code

}

// Pacote Caixa

package caixa;

public class Caixa {

// Code

}

public class Facade implements IFacade{

private Caixa caixa;

private Moedas moedas;

private Notas notas;

// Code

}

public interface IFacade {

// Code

}

public class Moedas {

// Code

}

public class Notas {

// Code

}

// Pacote Cliente

package cliente;

public class ClienteEspecial {

// Code

}

public class Facade implements IFacade {

private ClienteEspecial clienteEspecial;

// Code

}

public interface IFacade {

// Code

}

// Pacote Controller

package controller;

public class ControleCaixa {

// Code

}

public class ControleCliente {

// Code

}

public class ControleEstoque {

// Code

}

public class ControleMedicamento {

// Code

}

public class ControleVenda {

// Code

}

public class Facade implements IFacade {

private ControleCaixa controleCaixa;

private ControleCliente controleCliente;

private ControleEstoque controleEstoque;

private ControleMedicamento controleMedicamento;

private ControleVenda controleVenda;

// Code

}

public interface IFacade {

// Code

}

// Pacote Medicamento

package medicamento;

public class Facade implements IFacade {

private Medicamento medicamento;

// Code

}

public interface IFacade {

// Code

}

public class Medicamento {

// Code

}

// Pacote Persistencia

package persistencia;

public class CaixaDAO implements DAO<caixa.Caixa> {

// Code

}

public class ClienteEspecialDAO implements DAO<cliente.ClienteEspecial> {

//Code

}

public interface DAO<T> {

// Code

}

public class MedicamentoDAO implements DAO<medicamento.Medicamento> {

// Code

}

public class MoedasDAO implements DAO<caixa.Moedas> {

// Code

}

public class NotasDAO implements DAO<caixa.Notas> {

// Code

}

public class UsuarioDAO implements DAO<usuario.Usuario> {

// Code

}

public class VendaDAO implements DAO<venda.Venda> {

// Code

}

// Pacote Usuario

package usuario;

public class Facade implements IFacade {

private Usuario usuario;

// Code

}

package usuario;

public interface IFacade {

// Code

}

package usuario;

public class Usuario {

// Code

}

// Pacote Venda

package venda;

public class Facade implements IFacade {

private Venda venda;

// Code

}

public interface IFacade {

// Code

}

public class Venda {

// Code

}

// Pacote View

package view;

public interface FormularioCaixaAbertura {

// Code

}

public interface FormularioCaixaFechamento {

// Code

}

public interface FormularioCliente {

// Code

}

public interface FormularioControleEstoque {

// Code

}

public interface FormularioMedicamento {

// Code

}

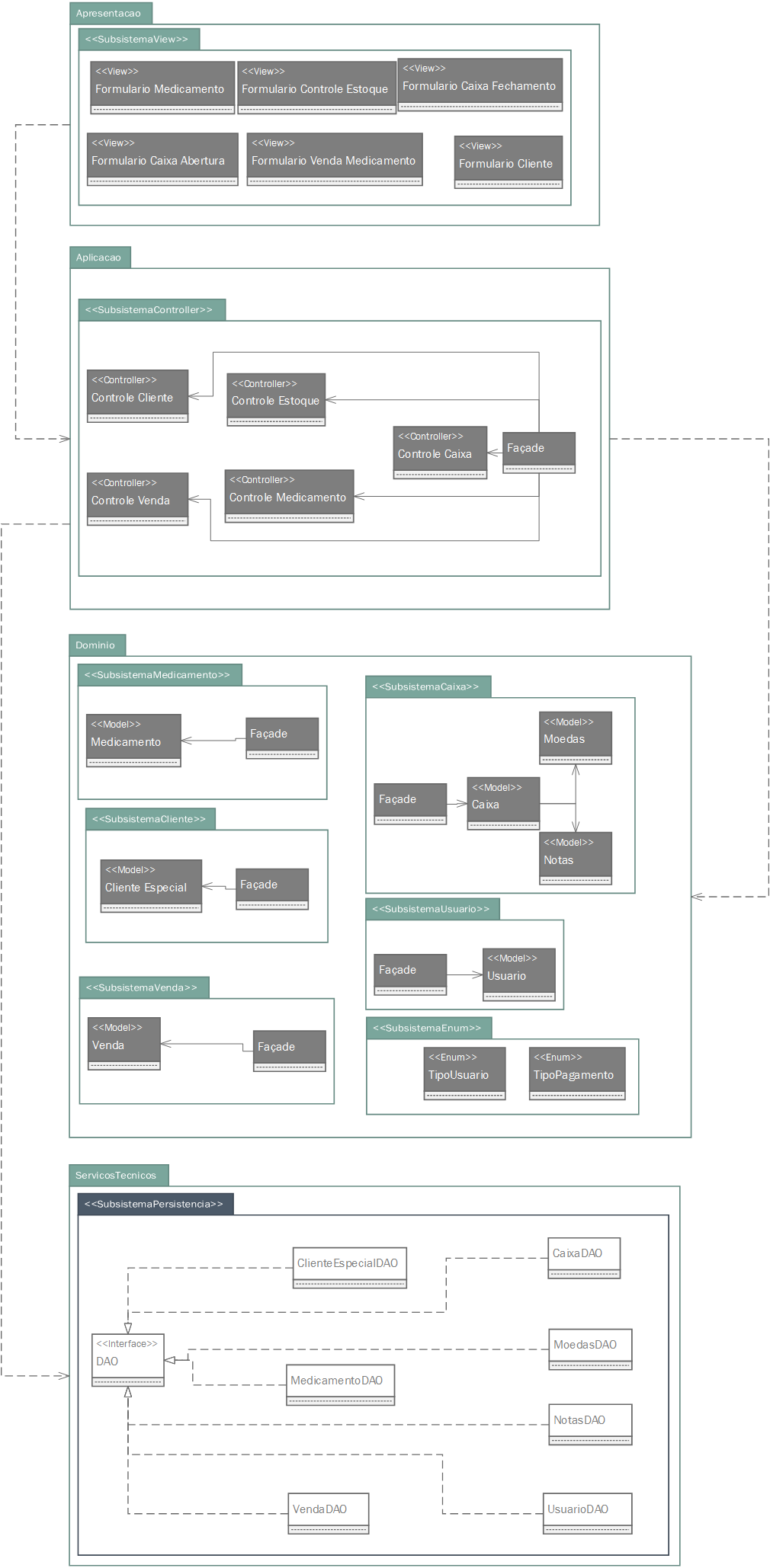
public interface FormularioVendaMedicamento {

// Code

}

**Parte C**

**17- Faça a alocação dos pacotes (subsistemas) nas camadas de software apresentadas em aula. As camadas devem ser representadas no sentido vertical e com arquitetura aberta.**



**Parte D**

**21- Abstraia o Mapa Mundi e modele um diagrama de pacotes com os devidos relacionamentos. Somente o nome de cada classe alocada no devido pacote é suficiente para este exercício.**

