

**ANÁLISE E DESENVOLVIMENTO DE SISTEMAS – MA4**

Engenharia de Software III – Profº Wilson Vendramel

LISTA 3 - VALIDAÇÃO

Anderson Marcondes Santana RA: 141681205 .

Eduardo Andrade RA: 1680481512006

Gabriel Viana Bueno Vieira RA: 1680481511036

Giovanni Armane RA: 1680481511016

Hugo de Melo Rodrigues RA: 141682211 .

Ronaldo Francisco Alves da Silva RA: 1680481511032

Yuri Cabral RA: 141681030 .

São Caetano do Sul

2016

PARTE A

**1- Apresente o diagrama de classes de projeto refinado com as seguintes notações:**

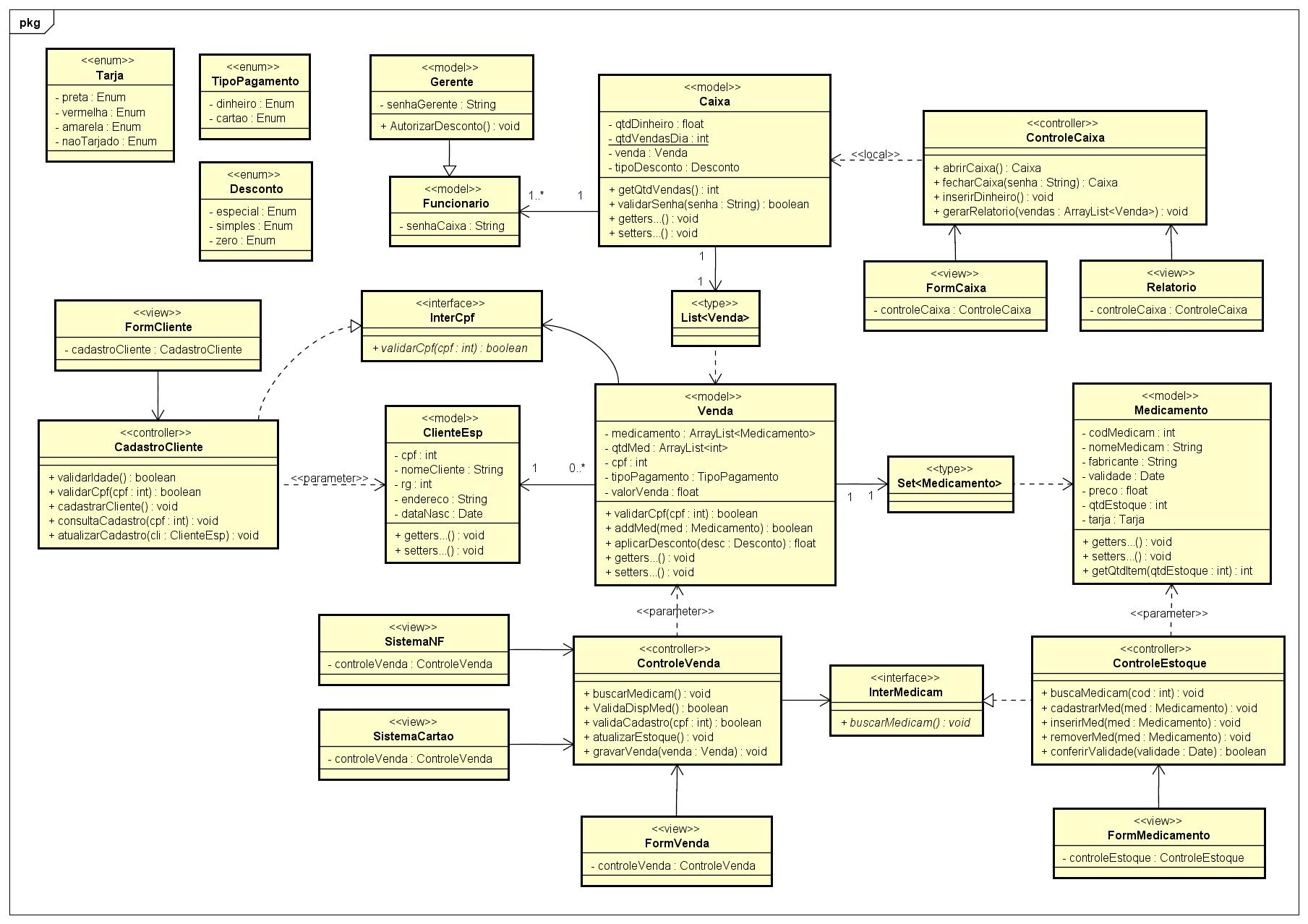
a. Dependências estruturais entre as classes de modelo;

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

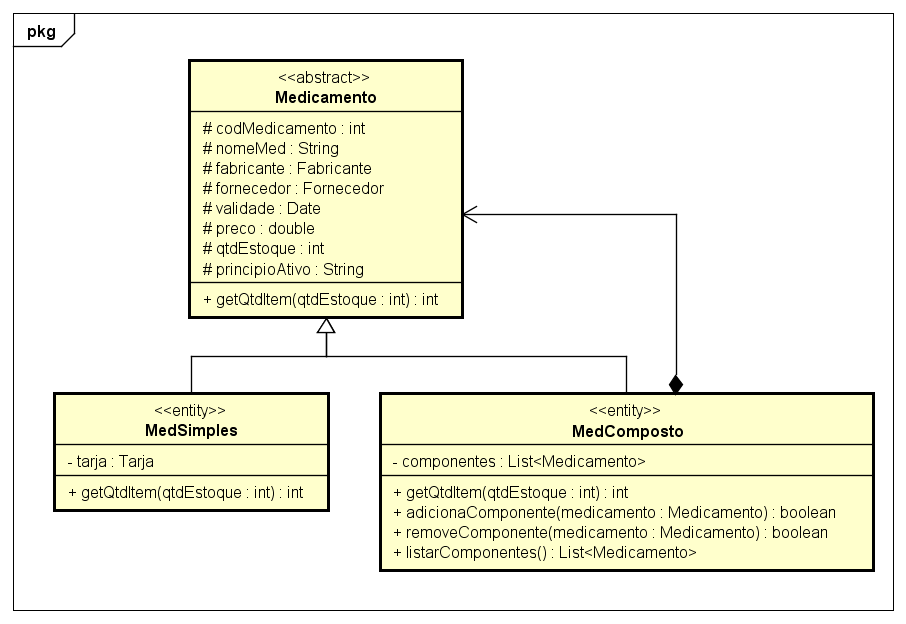
c. Dependências estruturais entre as classes de visão e controle;

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 objetivo do Composite neste diagrama é permitir que um medicamento composto seja criado com base em medicamentos simples, usando uma estrutura hierárquica. Caso o medicamento seja simples, ou seja, não composto de outras partes, ele é instanciado sem a adição de outros medicamentos.

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

public abstract class Medicamento {

protected int codMedicamento;

protected String nomeMed;

protected Fabricante fabricante;

protected Fornecedor fornecedor;

protected Date validade;

protected double preco;

protected int qtdEstoque;

protected String principioAtivo;

public abstract int getQtdItem(int qtdEstoque);

}

public class MedSimples extends Medicamento {

private Tarja tarja;

public int getQtdItem(int qtdEstoque) {

// Code

}

}

public class MedComposto extends Medicamento {

private List<Medicamento> componentes;

public int getQtdItem(int qtdEstoque) {

// Code

}

public boolean adicionaComponente(Medicamento medicamento) {

// Code

}

public boolean removeComponente(Medicamento medicamento) {

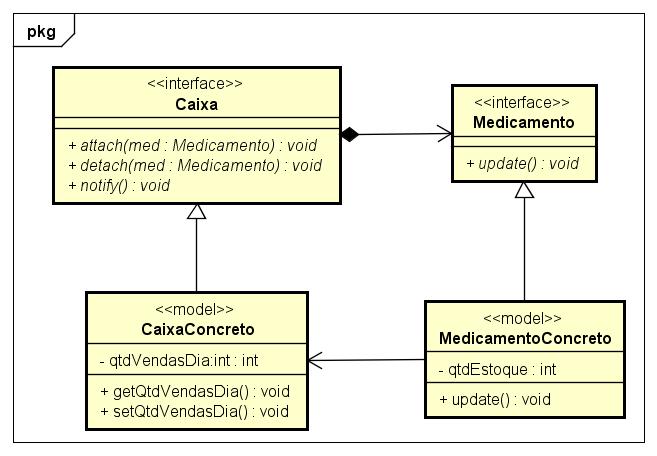
// Code

}

public List<Medicamento> listarComponentes() {

// Code

}

**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?** 

O objetivo do Observer nesse diagrama é para notificar as controllers de estoque e venda que o medicamento mudou o nome ou preço.

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

public interface Caixa {

private Medicamento medicamento;

public abstract void attach(Medicamento med);

public abstract void detach(Medicamento med);

public abstract void notify();

}

public class CaixaConcreto implements Caixa {

private int qtdVendasDia:int;

public void getQtdVendasDia() {

// Code

}

public void setQtdVendasDia() {

// Code

}

}

public interface Medicamento {

public abstract void update();

}

public class MedicamentoConcreto implements Medicamento {

private int qtdEstoque;

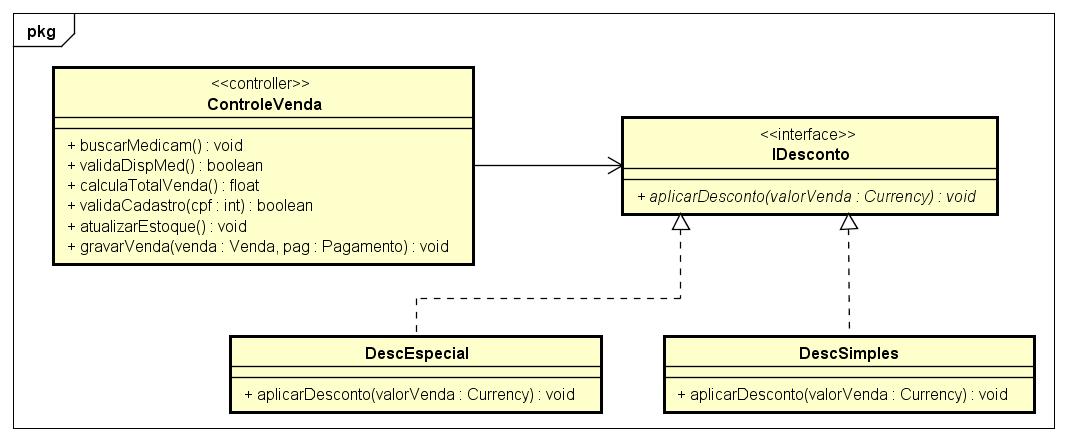
private CaixaConcreto caixaConcreto;

public void update() {

// Code

}

}

**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?** 

O objetivo do Strategy é permitir que as duas possibilidades de desconto sejam devidamente encapsuladas, implementando uma interface em comum e delegando a escolha do tipo de desconto as classes de desconto.

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

public class ControleVenda {

private IDesconto iDesconto;

public void buscarMedicam() {

// Code

}

public boolean validaDispMed() {

// Code

}

public float calculaTotalVenda() {

// Code

}

public boolean validaCadastro(int cpf) {

// Code

}

public void atualizarEstoque() {

// Code

}

public void gravarVenda(Venda venda, Pagamento pag) {

// Code

}

}

public interface IDesconto {

public abstract void aplicarDesconto(Currency valorVenda);

}

public class DescSimples implements IDesconto {

public void aplicarDesconto(Currency valorVenda) {

// Code

}

}

public class DescEspecial implements IDesconto {

public void aplicarDesconto(Currency valorVenda) {

// Code

}

}

**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?**

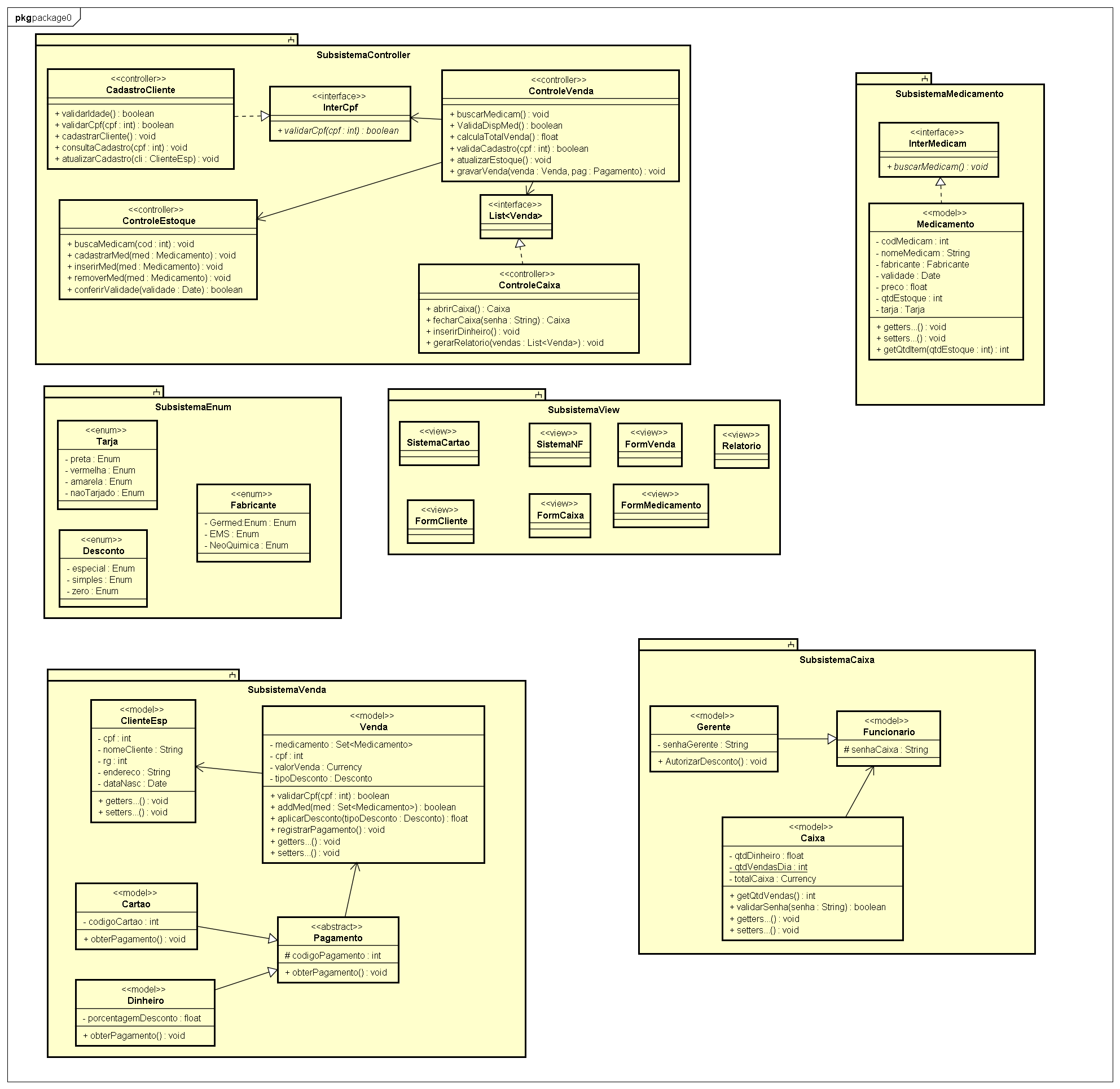
**9- Apresente a estrutura básica de código em JAVA, C# ou C++ para implementar o padrão de projeto Factory Method.**

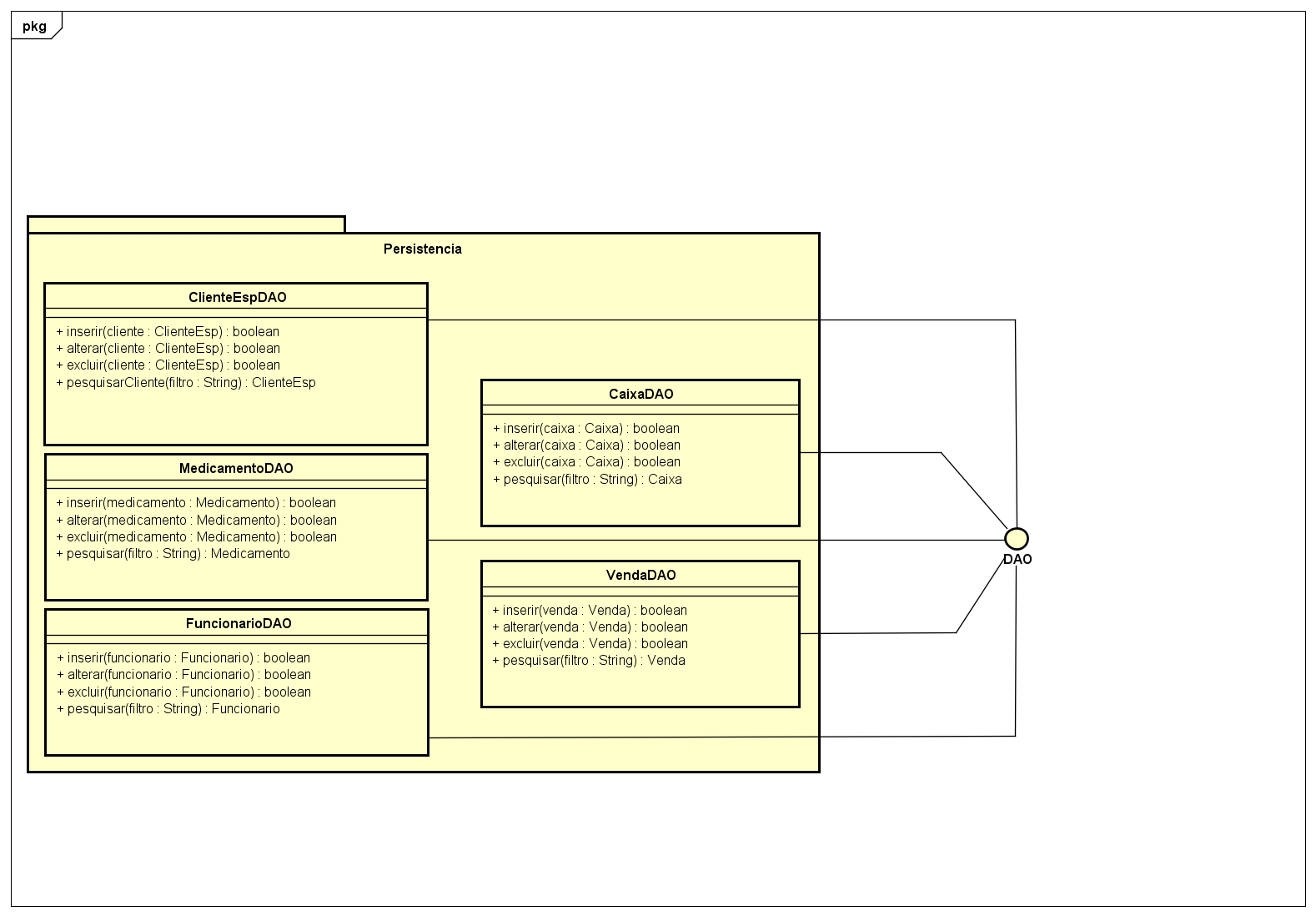
**10- Com base no diagrama de classes de projeto refinado nesta lista, modele o padrão de projeto Front Controller. Qual o propósito desse padrão no diagrama?**

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

**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(object T);

public boolean alterar(object T);

public boolean excluir(object T);

public T pesquisar(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(String filtro) {

// 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(String filtro) {

// 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(String filtro) {

// 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(String filtro) {

// 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

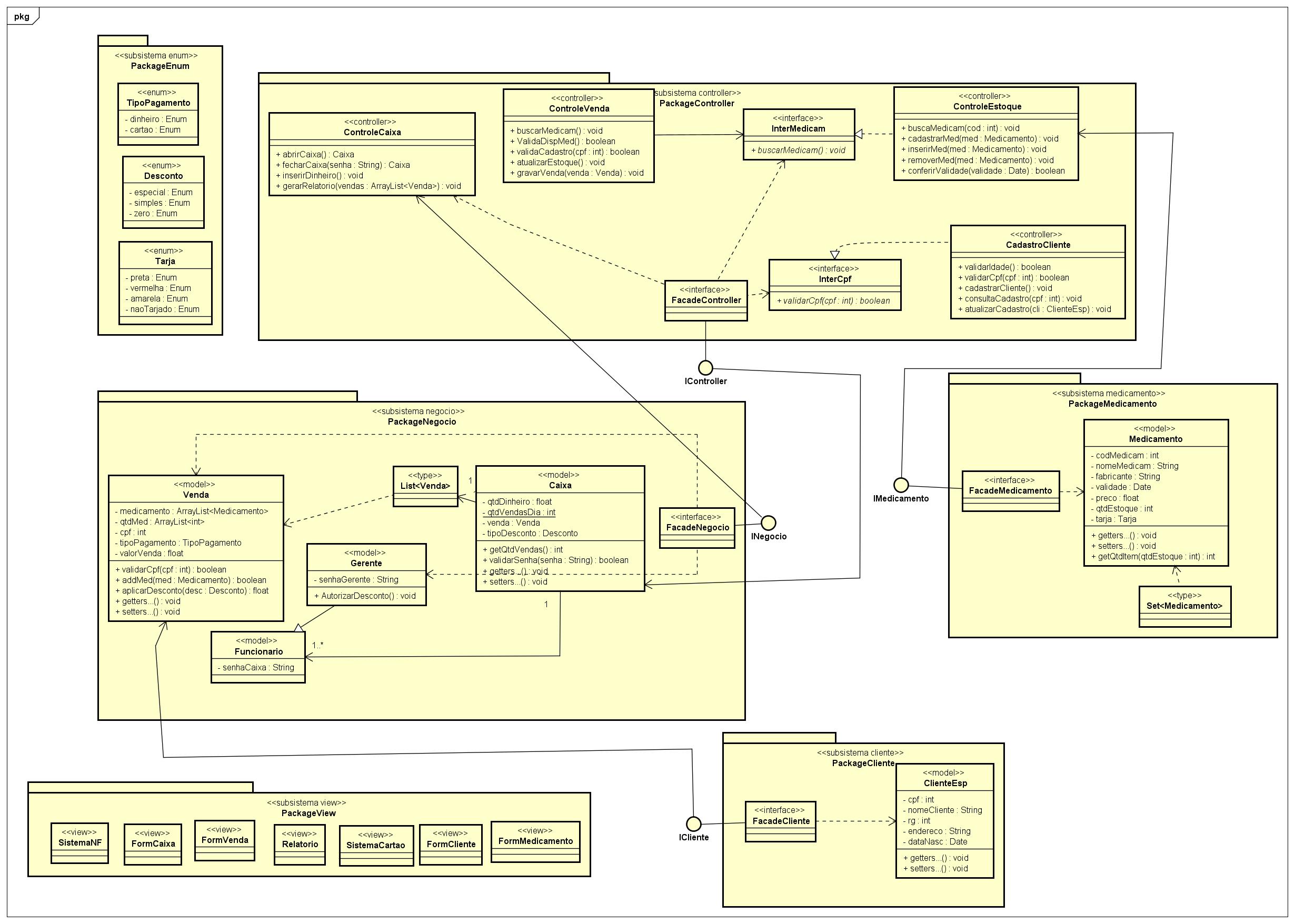
public Caixa pesquisar(String filtro) {

// 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 Facade.**

package PackageController;

public interface FacadeController {

private IController iController;

private IController iController;

}

package PackageCliente;

public interface FacadeCliente {

private ICliente iCliente;

}

package PackageMedicamento;

public interface FacadeMedicamento {

private IMedicamento iMedicamento;

}

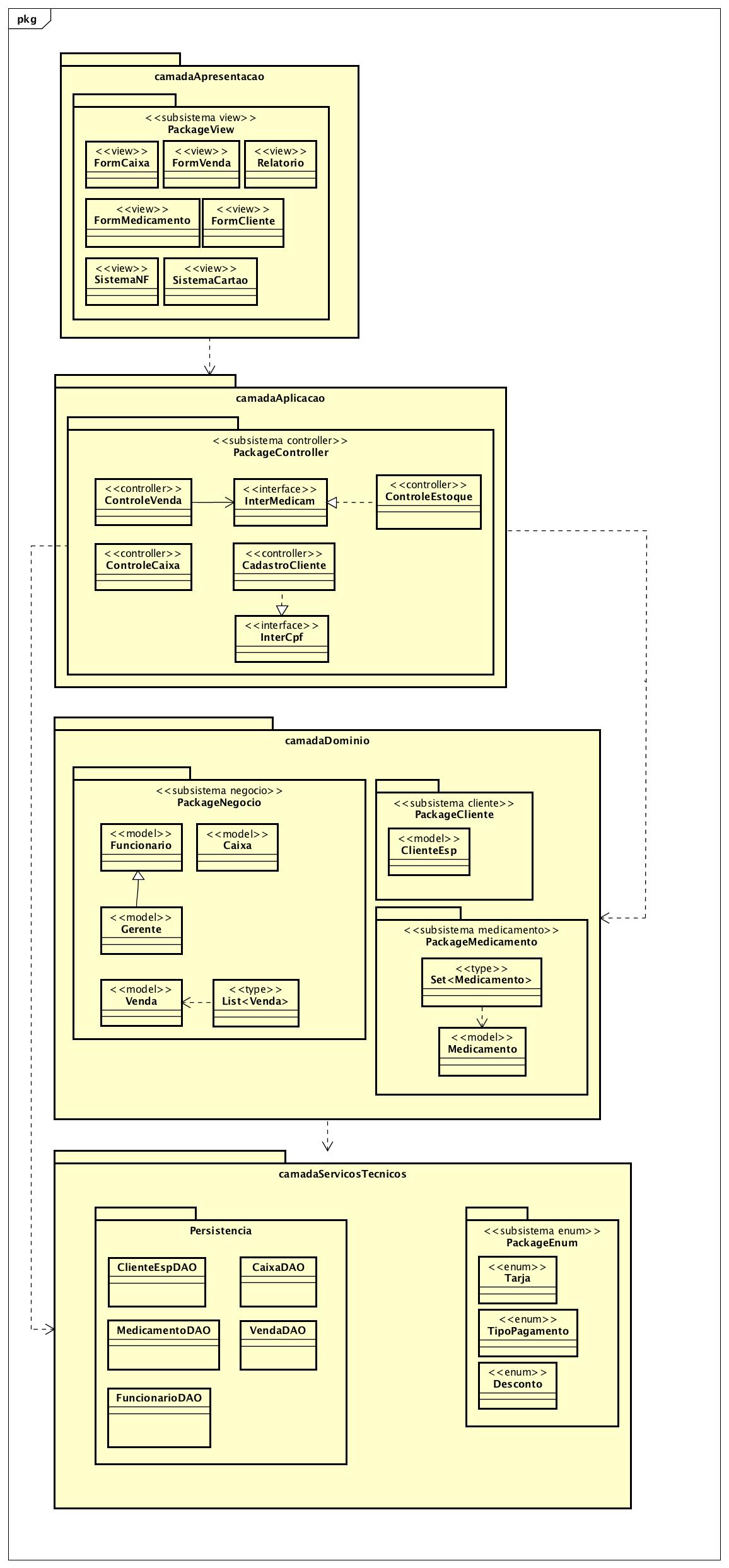
package PackageNegocio;

public interface FacadeNegocio {

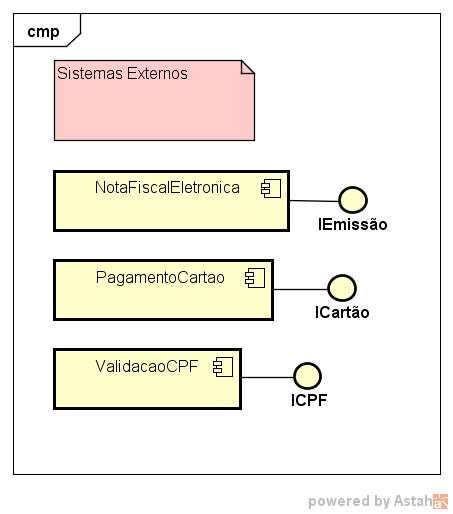
private INegocio iNegocio;

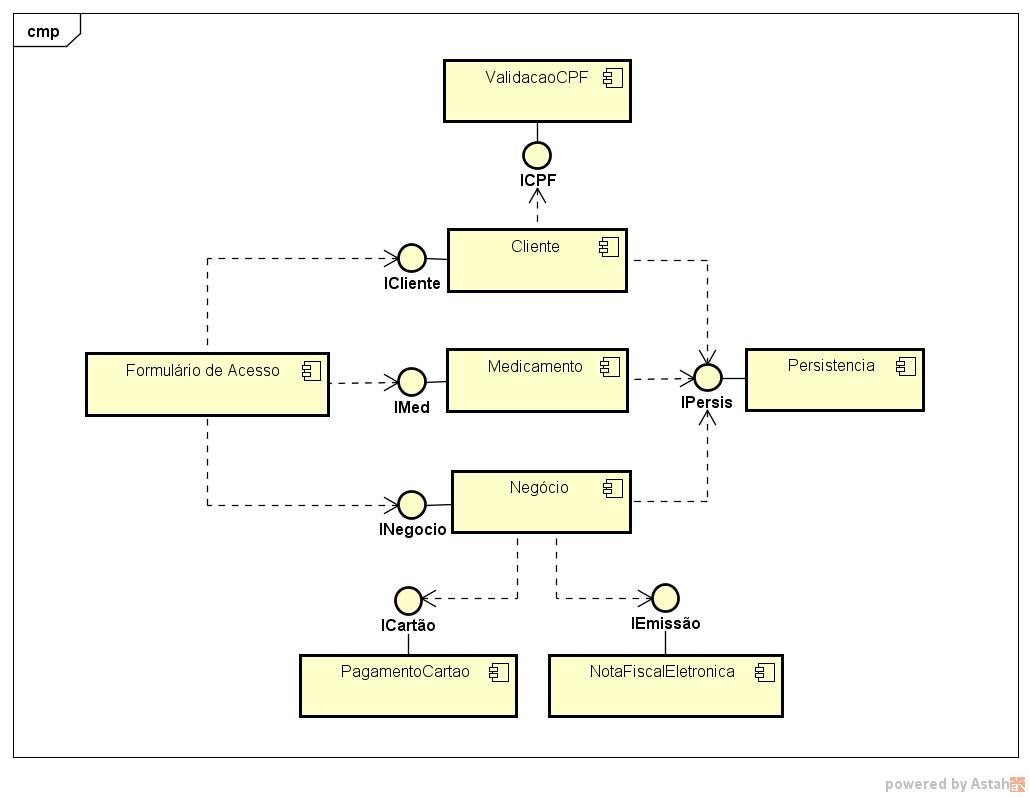
}

PARte c

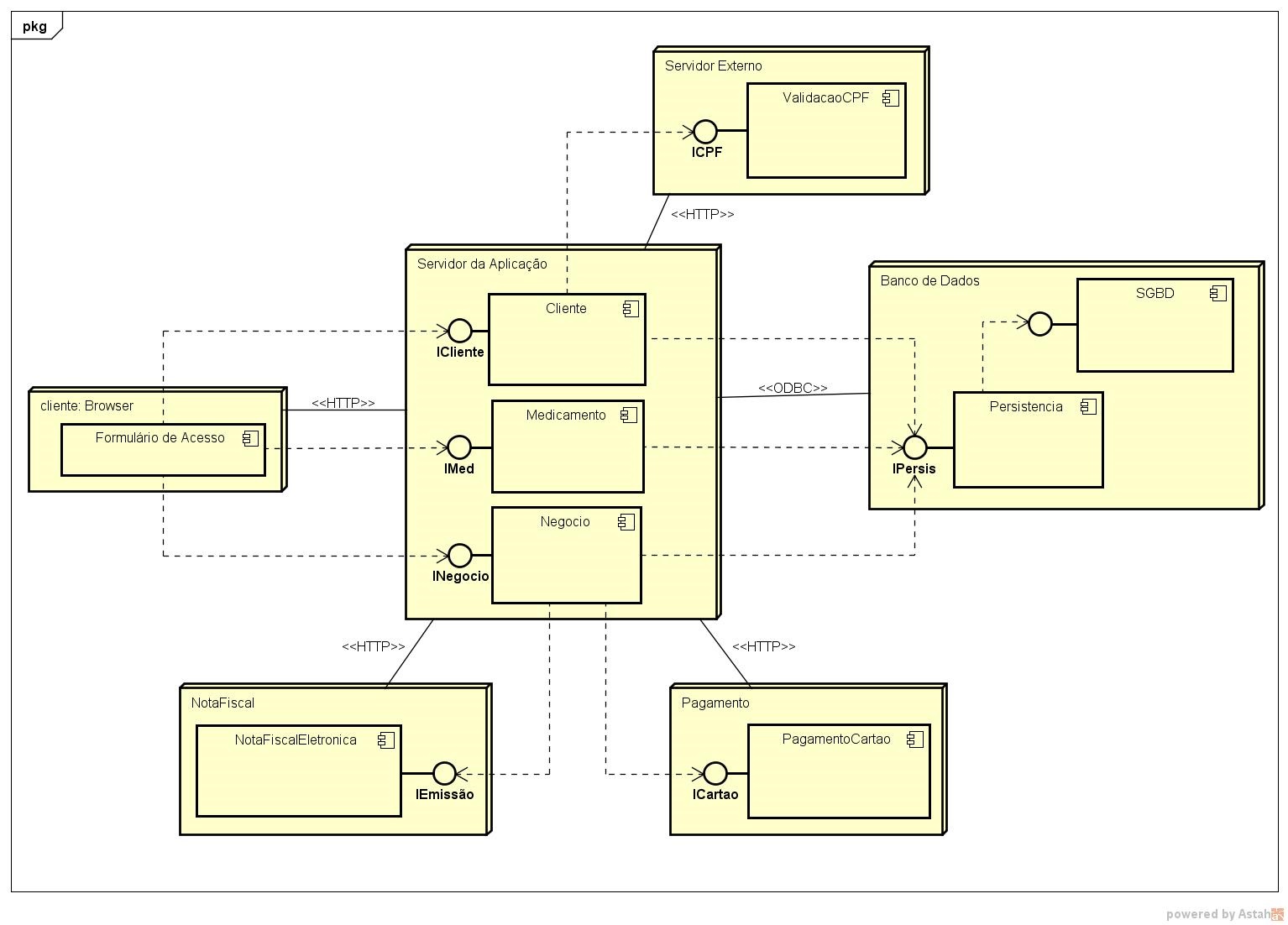
**17- Faça a alocação dos pacotes (subsistemas) nas camadas de software apresentadas em aula. As camadas devem ser representadas no sentido vertical e com arquitetura aberta.**

**18- Modele um componente para gerenciar o pagamento por meio de cartão, a emissão da nota fiscal eletrônica e a validação do CPF do cliente especial, lembrando que esses componentes são serviços terceirizados e que podem ter sido desenvolvidos numa plataforma diferente.**



**19- A partir da visão dos pacotes (subsistemas) e dos componentes de terceiros, construa o diagrama de componentes. Neste exercício, o pacote de classes enumeradas não precisa ser transformado para um componente e as classes de controle do pacote de controle podem ficar com seu respectivo pacote de classes de modelo, no mesmo componente.** 

**20- Com base na alocação dos pacotes (subsistemas) nas camadas de software e no diagrama de componentes, construa o diagrama de implantação distribuindo os componentes em seus respectivos nós. O seu projeto tem quantas camadas? Justifique a tua resposta.**



O sistema tem 4 camadas: Browser, sistemas externos (NotaFiscal, PagamentoCartao, ValidacaoCPF), Servidor da Aplicação e Banco de Dados.

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.**

