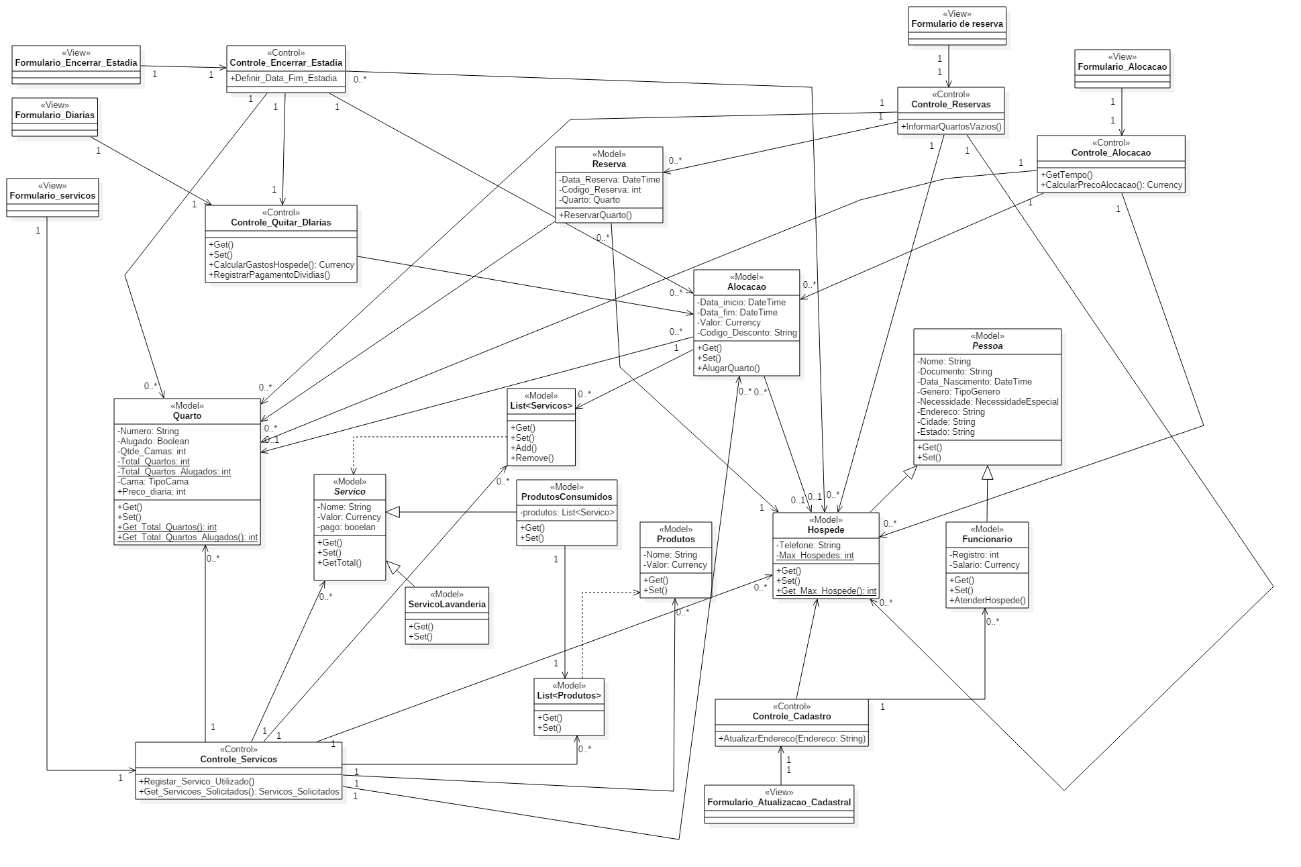
1. **Modele o diagrama de classes de projeto com padrão de projeto MVC, complementando as classes com atributos e métodos, se necessário. A notação de classe com seu devido <> deve continuar sendo mantida.**



1. **Qual é a classe de modelo mais coesa e a menos coesa do diagrama de classes de projeto? Justifique a tua resposta**

A classe Serviço é a mais coesa porque seus métodos e atributos são muito mais relacionados com sua responsabilidade do que das outras classes.

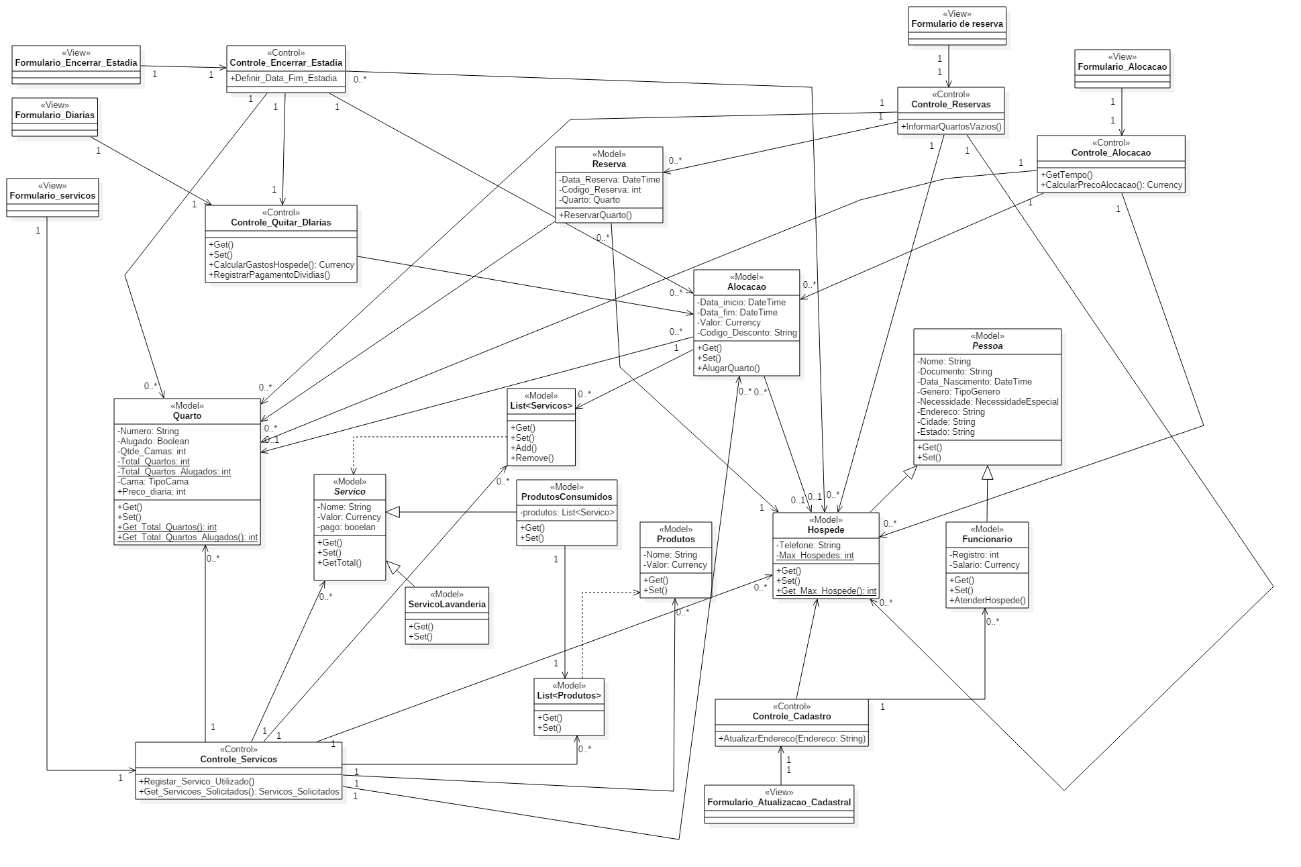
A classe Funcionário é a menos coesa porque nem todo funcionário possui a responsabilidade de atender um hospede

1. **Qual é a classe de modelo mais acoplada e a menos acoplada do diagrama de classes de projeto? Justifique a tua resposta.**

A classe Hóspede é a mais acoplada porque o programa depende em grande parte do diagrama

A classe Funcionário é a classe menos acoplada pois ela apenas tem função de confirmação de dados e ações.

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



A vantagem da dependência estrutural é que ela é mais fácil para implementar, porém por outro lado ela aumenta muito mais o acoplamento do que a dependência não estrutural

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

Exemplo em C# de duas classes relacionadas por uma dependência estrutural do diagrama, as classes Alocacao e Quarto

class Alocacao

{

private DateTime data\_inicio;

private DateTime data\_fim;

private double valor;

private string codigo\_desconto;

private Quarto quarto;

}

class Quarto

{

private string numero;

private bool alugado;

private int qtde\_camas;

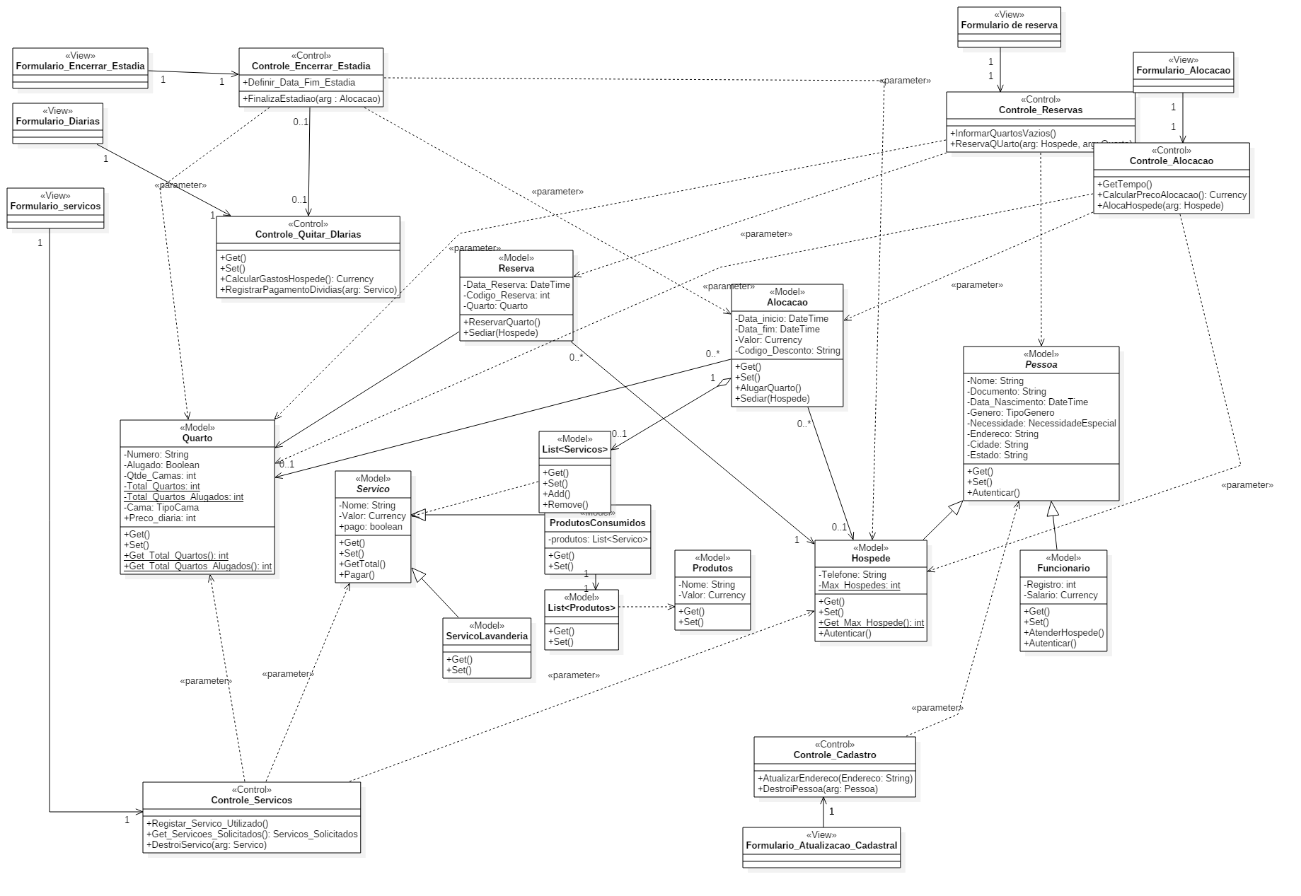
private static int total\_quartos;

private static int total\_quartos\_alugados;

private double preco\_diaria;

}

1. **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.**



A vantagem da dependência não estrutural é que faz com que o encapsulamento de cada classe aumente e diminua o acoplamento entre as classes. A desvantagem é que se houver alguma alteração no modelo independente pode ocorrer uma mudança no modelo dependente.

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

Exemplo em C# de duas classes relacionadas por uma dependência não estrutural por parâmetro do diagrama, as classes Alocacao e Reserva

class Alocacao

{

private DateTime data\_inicio;

private DateTime data\_fim;

private double valor;

private string codigo\_desconto;

private Quarto quarto;

}

class Reserva

{

private DateTime data\_reserva;

private int codigo\_reserva;

public void ReservarQuarto(Quarto quarto)

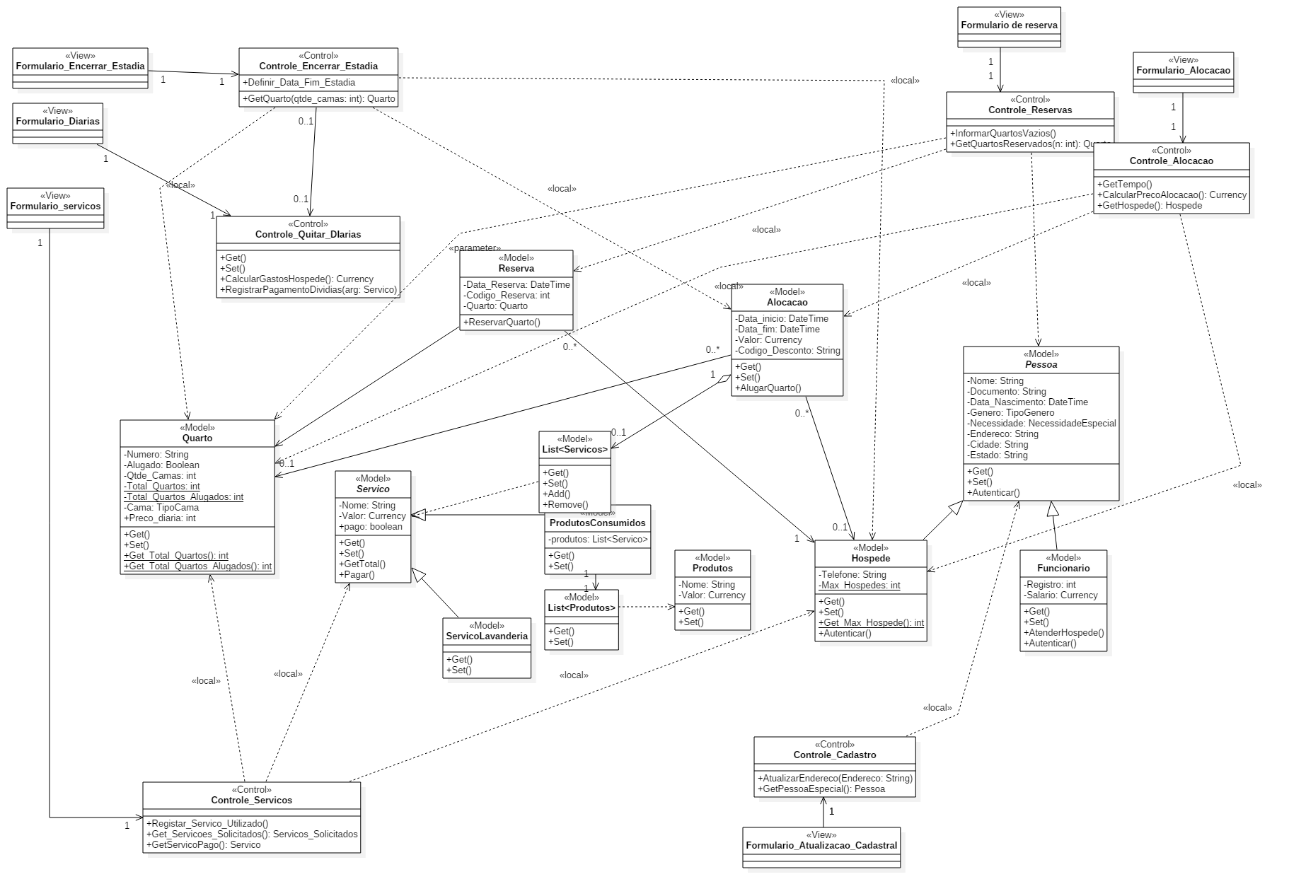
{

//Estrutura do código...

}

}

1. **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.**



Aumenta o encapsulamento de cada classe e diminuir a acoplamento entre as classes. Quanto menos dependências houver no modelo de classes, maior é o encapsulamento e menor o acoplamento.

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

Exemplo em C# de duas classes relacionadas por uma dependência não estrutural por variável local do diagrama, as classes Reserva e Quarto

class Reserva

{

private DateTime data\_reserva;

private int codigo\_reserva;

public Quarto GetQuarto(Controller controller)

{

//Pega o quarto pelo controller e o retorna

}

}

class Quarto

{

private string numero;

private bool alugado;

private int qtde\_camas;

private static int total\_quartos;

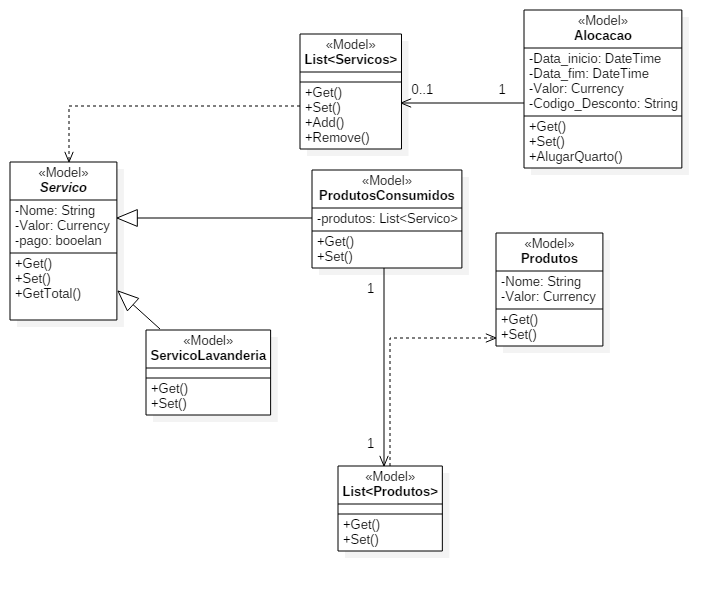
private static int total\_quartos\_alugados;

private double preco\_diaria;

}

**10. Modele as classes parametrizadas com a estrutura <List> para resolver o lado muitos dos relacionamentos.**

**Explique a vantagem e desvantagem dessa estrutura.**



Vantagens: variação de tamanho do vetor conforme necessidade e dados estão sempre ordenados

Desvantagem: usa mais memória para cada dado inserido, leitura somente em um sentido (sentido da escrita dos dados)

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

Exemplo em C# de a classe Alocacao possuindo uma estrutura List<Servico> para usar muitos Servico

class Alocacao

{

private DateTime data\_inicio;

private DateTime data\_fim;

private double valor;

private string codigo\_desconto;

private Quarto quarto;

private List<Servico> servicos = new List<Servico>();

}

class Servico

{

private string nome;

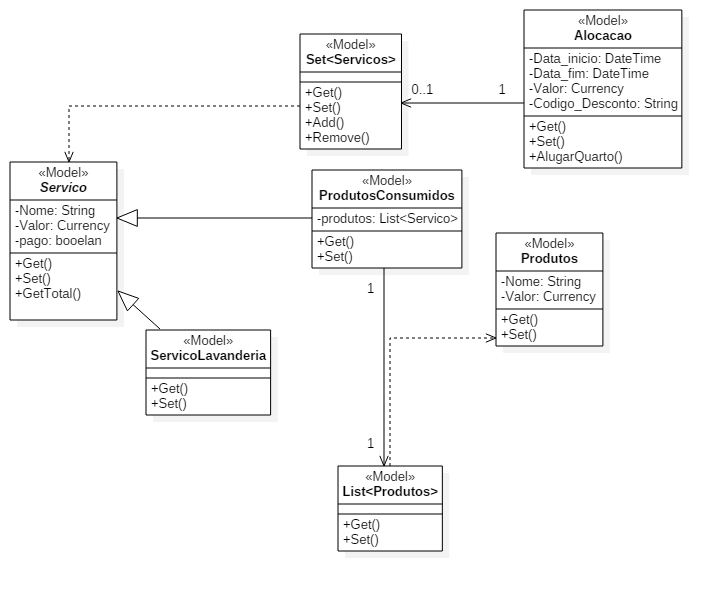
private double valor;

private bool pago;

}

**12. Modele as classes parametrizadas com a estrutura <Set> para resolver o lado muitos dos relacionamentos.**

**Explique a vantagem e desvantagem dessa estrutura.**

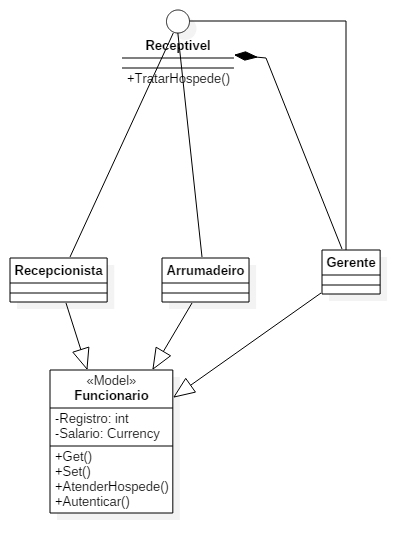


Vantagens: estrutura mais rápida do que o List.

Desvantagens: não tem como fazer pesquisas por índices, e ela é desordenada, logo mais lenta. Além de não permitir dados repetidos, sendo que no modelo ficaria impossível de pedir o mesmo serviço e produto mais de uma vez.

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

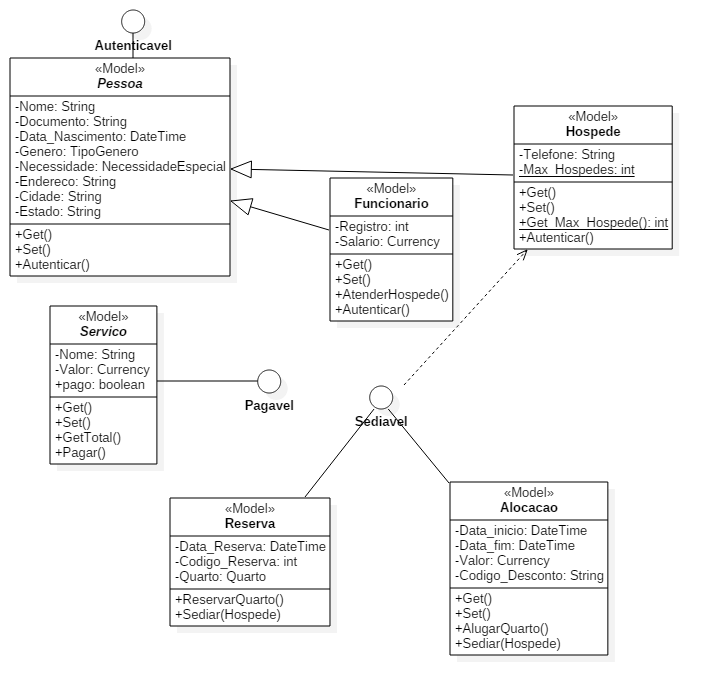
**com a estrutura <Set>.**



**14. Qual a diferença entre classe parametrizada, multiobjetos e collection?**

Classe parametrizada é uma classe que é utilizada para definir outras classes, e uma collection pode ser representada por uma classe parametrizada. Em Java, uma classe parametrizada equivale à uma Collection. Quanto a multiobjetos, eles podem representar o lado muitos numa associação um para muitos

**15. 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.**



Todas as pessoas são autenticáveis, porém dependendo de qual especialização, é pedido que seja informado o Registro de Funcionário ou o Documento caso Hospede.

Todos os serviços devem ter o registro de pagamento.

Tanto Alocação quanto Reserva pertencem a somente a um Hospede.

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

Exemplo em C# de a classe Alocacao possuindo uma relação de interface com Sediavel

class Alocacao : Sediavel

{

private DateTime data\_inicio;

private DateTime data\_fim;

private double valor;

private string codigo\_desconto;

private Quarto quarto;

private List<Servico> servicos = new List<Servico>();

public bool Sediar(Hospede hospede)

{

//Código para sediar

}

}

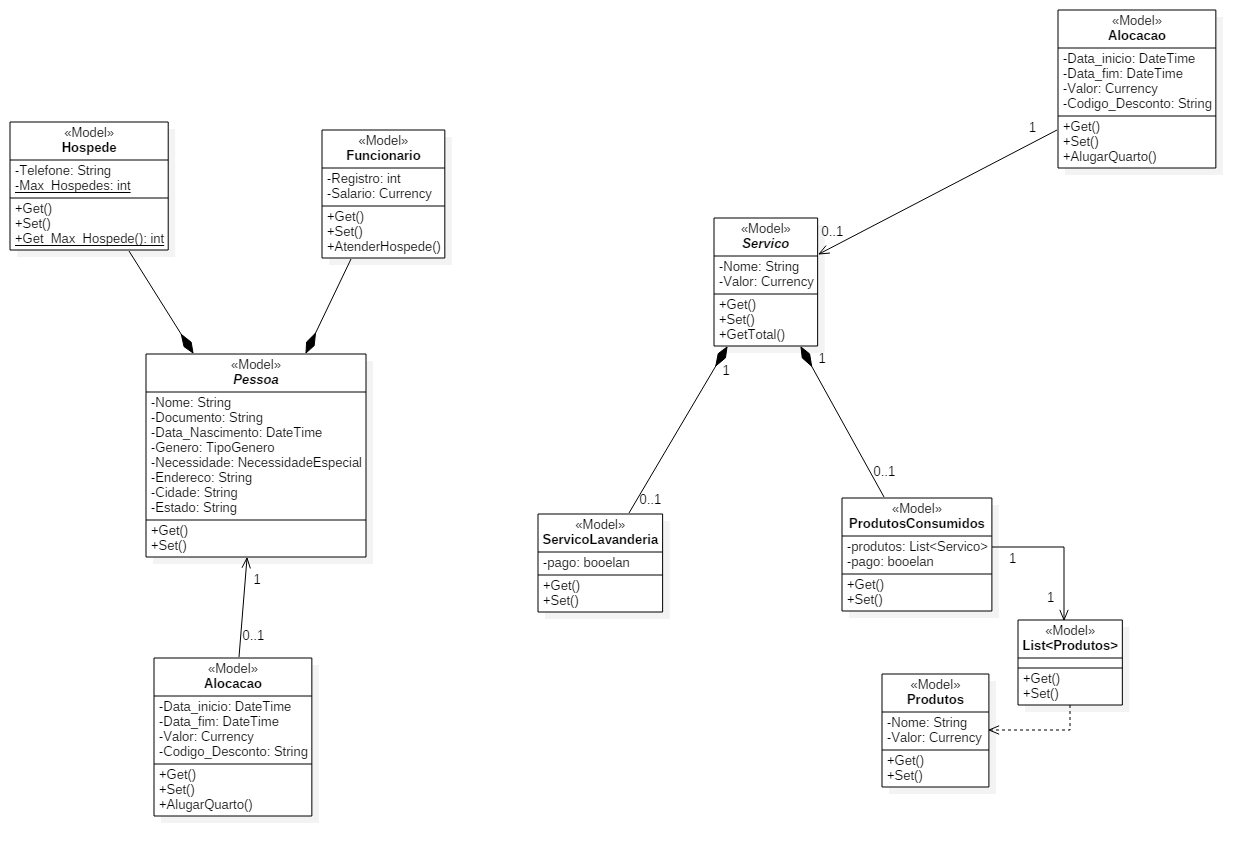
interface Sediavel

{

bool Sediar(Hospede hospede);

}

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



Na primeira é preciso que a classe alocação dependa de Pessoa invés de hospede. Um Funcionário pode tanto deixar ser um Funcionário quanto pode também começar a ser um Hóspede e o inverso também é possível.

Uma Alocação pode ter usufruído ou não de serviços e dessa forma é possível registrar se os serviços foram pagos ou não.

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

Exemplo em C# de a classe Servico possuindo uma delegação com ServicoLavanderia, onde ServicoLavanderia depende de Servico (É composto)

class Servico

{

private string nome;

private double valor;

}

class ServicoLavanderia

{

private bool pago;

private Servico servico;

public ServicoLavanderia(Servico servico)

{

this.servico = servico;

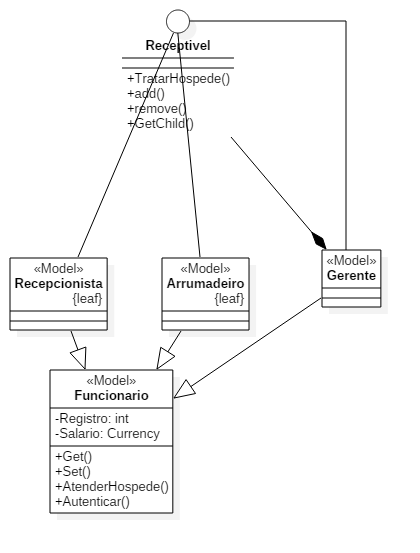
}

}

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

|  |  |  |
| --- | --- | --- |
|  | **Vantagens** | **Desvantagens** |
| **Generalização** | Reusabilidade | Muito estático |
| Extensibilidade | Não pode ser usado com classes parcialmente abstratas |
| **Realização (Interface)** | Revelar as operações de um objeto, sem revelar a sua classe | Pode haver poluição visual e confundir usuários |
| Facilita acoplamento | Pode ter problemas com plataformas diferentes e consumo de memoria |
| **Delegação** | Mudança dinâmica em tempo de execução | Muito genérico |
| Reuso extremo | Desempenho |

**20. Modele o padrão de projeto Composite. Qual o propósito desse padrão no diagrama de classes de projeto?**



O padrão composite é normalmente utilizado para representar listas recorrentes - ou recursivas - de elementos. Esta é forma de representar elementos compostos em uma hierarquia de classes, que permite que os elementos contidos em um objeto composto sejam tratados como se fossem um único objeto, assim poupando recursos. Além disso, os objetos no composite são fracamente acoplados, porque todos eles seguem simplesmente a mesma interface. Finalmente, o composite dá uma boa estrutura para os objetos, ao invés de manter todos eles em variáveis separadas ou em um array.

A desvantagem é um efeito adverso que terá no desempenho se o composite aumentar muito.

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

public class Funcionario extends Pessoa

{

private int registro;

private double salario;

/\*\*

\* @return the registro

\*/

public int getRegistro() {

return registro;

}

/\*\*

\* @param registro the registro to set

\*/

public void setRegistro(int registro) {

this.registro = registro;

}

}

public class Gerente extends Funcionario implements Receptivel {

List<Receptivel> lista = new ArrayList<>();

@Override

public void TratarHospede() {

throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.

}

@Override

public void add(Receptivel r) {

lista.add(r);

}

@Override

public void Remove(Receptivel r) {

lista.remove(r);

}

@Override

public Receptivel GetChild(int r) {

return lista.get(r);

}

}

public class Recepcionista extends Funcionario implements Receptivel{

@Override

public void TratarHospede() {

throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.

}

@Override

public void add(Receptivel r) {

throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.

}

@Override

public void Remove(Receptivel r) {

throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.

}

@Override

public Receptivel GetChild(int r) {

throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.

}

public interface Receptivel {

void TratarHospede();

void add(Receptivel r);

void Remove(Receptivel r);

Receptivel GetChild(int r);

}

public class Arrumadeira extends Funcionario implements Receptivel {

@Override

public void TratarHospede() {

throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.

}

@Override

public void add(Receptivel r) {

throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.

}

@Override

public void Remove(Receptivel r) {

throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.

}

@Override

public Receptivel GetChild(int r) {

throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.

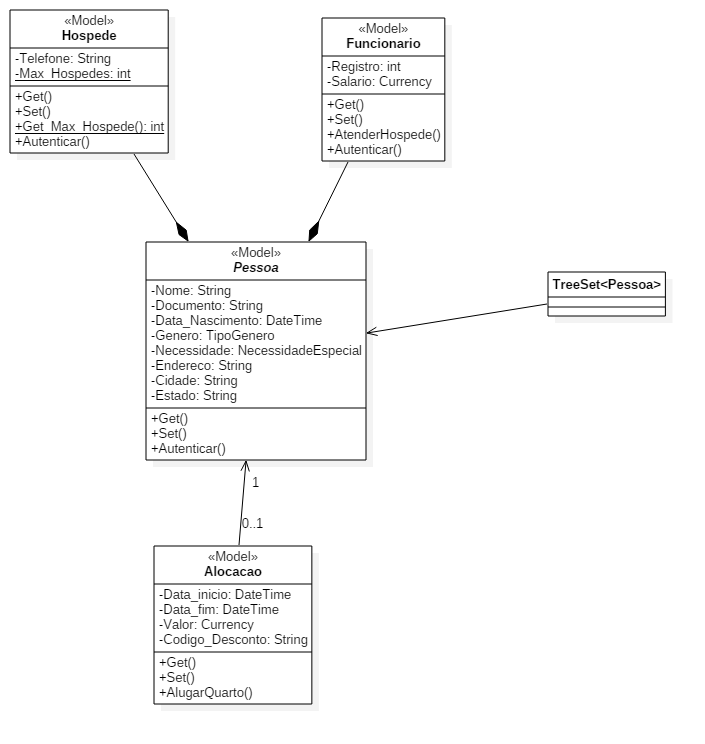
}

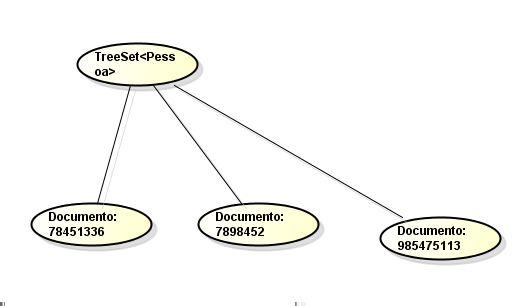
}

**22. - O diagrama de classes de projeto apresenta caso de classificação dinâmica? Justifique a tua resposta**

Sim, pois uma Pessoa pode ser tanto Hospede quanto um Funcionário.

**23. Modele a estrutura <TreeSet>. Justifique a razão dessa estrutura no seu diagrama. Represente graficamente como essa árvore trabalharia em tempo de execução.**





Organização das Pessoas cadastradas por Documento para geração de relatórios impedindo que haja repetição de cadastro e ainda sim ordenando-os

24. Construa um diagrama de sequência para o CSU2 utilizando notação MVC.

25. - Construa um diagrama de sequência para o CSU5 utilizando notação MVC.

Parte B:

Sabiá:

Asa

Bico

Cloaca

Ceco

Papo