INSTITUTO FEDERAL DO CEARÁ

CURSO DE GRADUAÇÃO EM ENGENHARIA DA COMPUTAÇÃO

PEDRO CAVALCANTE DE SOUSA JUNIOR

FICHAMENTO DOS CAPITULOS 6 E 7 DO LIVRO DA CADEIRA DE PROGRAMAÇÃO ORIENTADA A OBJETO

FORTALEZA

2021

Capitulo 6: Os conceitos relacionais

O capitulo começa apresentando o núcleo de ideia geral que vai ser abordado durante os tópicos. Neste é dito que os conceitos relacionais são o que possibilita a criação de classes.

6.1 Herança.

Herança é um dos conceitos que possibilita a representação do mundo real. Como exemplo dado por autor, pensemos na herança como a biologia faz a classificação dos seres vivos: Reino, Filo, Classe, Ordem, Família, Gênero e espécie. Cada uma dessas classificações "herda" características da passada, e tem em comum com seus semelhantes certas heranças.

Na orientação a objeto esse processo de herança é feito através de classes, uma classe herda de outra. Herança é uma relação onde uma classe mãe (ou super classe), transmite algumas características (processo de herança) para a classe filha (ou subclasse) como por exemplo métodos e atributos, este é o que é caracterizado como processo de herança. Além disso, tal só ocorre entre classes, objetos não herdam de outros objetos pois os mesmos só existem em tempo de execução do código.

A partir disso nos deparamos com o conceito de níveis de herança, a herança pode acontecer em quantos níveis se achar necessário, no entanto, é recomendado que apenas usemos 4 níveis pois a cada nível nos distanciamos mais do conceito base e mais difícil fica de se entender o código e definir o que realmente se quer modelar. Esse processo se dá através do fato de uma classe poder herdar de uma classe que já está herdando de outra classe, tal relação é conhecida como Hierarquia de classes.

Ademais, percebemos a intrínseca ligação entre herança e o fundamento de reuso já explicado anteriormente, afinal, quando definimos uma classe da forma mais abstrata possível nosso objetivo é justamente reutilizar seus conceitos e atributos para definir seus similares. Sendo assim, uma classe ao herdar de outra pode apenas acrescentar atributos e nunca excluir, gerando assim os subtipos.

Depois de apresentados esses conceitos o autor remete novamente ao exemplo do Hospital, já tratado em capítulos anteriores, onde define a classe "Pessoa", para um sistema hospitalar e demonstra a herança para outras classes, como por exemplo, a classe Funcionário que herda da classe Pessoa, fazendo assim uma relação de herança

entre subclasse e superclasse, depois disto a classe Funcionário se torna superclasse das Classes Gerente e médico, mostrando assim que uma mesma classe pode ser subclasse de uma e super classe de outra.

A partir dessas caracterizações de cada herança de uma classe, podemos definir atributos e características únicas para a herdeira, como por exemplo, a classe Funcionário tem seus atributos definidos e serve para o molde das classes Gerente e Medico, no entanto, a classe Medico tem uma característica única, os médicos possuem CRM (conselho regional de medicina) e isso pode ser especificado na sua classe. Tais processos são conhecidos como Generalização e especialização, quanto mais se sobe na hierarquia de classes mais genérico fica o conceito.

Mas e quando usamos a herança? O livro nos da a resposta por forma de uma pergunta. “Uma coisa é outra?” e se a resposta for não, não deveremos usar a herança para trabalhar com isso. Por exemplo, médico é funcionário? Sim, então devemos usar a herança para a representação do mesmo.

Tipos de classe

Uma classe pode ser de dois tipos: abstrata e concreta.

Classe abstrata:

- É a implementação do processo de abstração.
- Conceitos Genéricos
- Estão incompletas e são completadas pelas classes que herdam da mesma
- Não podem ser instanciadas (criar objetos a partir delas)
- Estão no topo da hierarquia de classe

Por exemplo, no exemplo do hospital pode não ser interessante trabalhar diretamente com a classe Pessoa, pois é importante que seja feita a especificação entre paciente, médico, gerente e etc. Sendo assim, manipularemos apenas suas subclasses, ademais também é possível transformar a classe Pessoa em abstrata mesmo que na sua primeira definição ela não tenha sido colocada assim, obrigando-a a trabalhar apenas com suas subclasses.

Por demais, também é visto que as vezes é necessário que a classe Médico se torne também em abstrata, pois por exemplo, um médico anestesista (especialidade da

medicina) é mais interessante de ser trabalhado do que o conceito de médico em si, por isso transformar a classe Medico em abstrata seria uma boa prática em OO, sendo assim, classes abstratas podem herdar de classes abstratas, isso implica que classes abstratas não estão apenas no topo da hierarquia de classes, como também no meio.

Além e definir classes como abstratas também podemos definir métodos como abstratos, ao ser declarado não deve possuir nenhuma implementação e apenas sua declaração.

Classe concreta:

As classes concretas em si são da maneira com que já estava sendo trabalhadas durante o livro:

- Representam o uso direto do que vai ser trabalhado
- Devem ser instanciadas
- São especificadas
- Manipulação delas é vital para o uso do programa

No exemplo do hospital, as classes concretas seriam: Anestesista, Gerente, Paciente e etc. Tais classes possuem métodos implementados e são manipuladas para o funcionamento do programa. A sua definição no código segue a mesma como já era definida anteriormente nos capítulos. Também é supracitado que, ao herdar um método abstrato de uma classe abstrata, elas devem por obrigação implementar o método pois essas classes são de uso direto e os métodos devem ser usados em algum ponto do programa.

Tipos de Herança

Existem dois tipos de herança: a simples e a múltipla:

Simples: Acontece quando uma subclasse herda de apenas uma classe mãe, ou seja, herdou e especificou atributos e métodos de apenas uma super classe.

Múltipla: Ocorre quando a subclasse não necessita de apenas uma super classe, mas sim de duas ou mais. Sendo assim, a classe filha pode especificar mais de um conceito da aplicação. A seguir, o livro do exemplo de um hospital que dê ao médico chefe da equipe a responsabilidade de gerenciamento do hospital, sendo assim ele seria tanto médico quanto gerente tendo que herdar conceitos das duas classes.

Upcast e Downcast

Upcast: Operação de conversão onde ocorre a promoção de subclasses a superclasses.

Pela subclasse ser do mesmo tipo da superclasse essa operação é permitida. A seguir o livro trás o exemplo em código de como isso é feito. Ao ser apresentando é notado que é a conversão e o processo de upcast é dada de forma implícita. Ao compararmos o "cast" de uma linguagem estruturada com esse caso especifico em OO, notamos que uma ideia entre as duas se aproxima bastante, a ideia de "caber", um int pode sofrer cast em um float por que "cabe" nele, a ideia é a mesma para OO, no entanto, é no conceito de subtipo, se uma classe é subtipo de outra ela consequentemente cabe nela.

Downcast: Operação contrária a Upcast, onde ocorre a transformação de super classe em subclasses. Esta operação é desencorajada por que podem ocorrer diversas especializações distintas através de uma generalização.

Por exemplo: todo gerente é funcionário, todo médico é funcionário, mas nem todos funcionários são médicos por isso existem diversos tipos de funcionários e cada um deve ter sua especialização em uma classe. A seguir o autor apresenta exemplos em códigos de como fazer este processo e sintaxe.

O Poliformismo

Em certos contextos precisamos que um mesmo método faça trabalhos diferentes, por exemplo, um médico em uma cirurgia de trabalho de parto pode ter diferentes funções, o anestesista, o obstetra e o pediatra. Todos estão operando nesse momento, no entanto, as ações do "operar" são diferentes para cada um, um aplica a anestesia, outro faz a cirurgia e o ultimo faz a checagem com exames que a criança nasceu saudável.

Sendo assim, o mesmo método pode ser especializado para diferentes contextos, isso é o que chamamos de poliformismo, a vantagem do mesmo é que objetos distintos continuam executando a mesma ação, dando assim flexibilidade ao código.

A seguir no capitulo é demonstrando um exemplo de implementação, o exemplo consiste no já supracitado "parto", onde todos os médicos realizam a mesma ação "operar". Tal fato demonstra que durante o procedimento pode-se alterar os médicos de

posição sem alterar a ação que está sendo executada, pois ela se adaptará a quem está executando-a. A melhor forma de se trabalhar com poliformismo é usar classes e métodos abstratos, pois os mesmos serão especificados pelas subclasses que o herdarem.

A relação entre poliformismo e herança é intrínseca, para que exista poliformismo é necessário que se exista herança, apenas assim um método abstrato definido anteriormente em uma classe poderá ser especificado e implementado. No entanto, o contrário não é verdadeiro, para usar herança não é necessário usar poliformismo.

A sobrescrita

A sobrescrita nada mais é que a redefinição de um método herdado por uma classe, essa alteração pode acrescentar ou eliminar um certo comportamento do método anteriormente definido. O exemplo dado pelo livro é o de um anestesista, este possui o método "operar", no entanto, se um residente de anestesia for fazer o processo, provavelmente faltará alguns passos que um médico já experiente executa, por isso ele "sobrescreverá" o método antes definido pelo anestesista, excluindo certos passos. Sendo assim houve a sobrescrita do método "operar" da classe Anestesista pela classe Residente.

Porém, um adendo, para a sobrescrita ser efetivada é necessário que o método sobrescrito da subclasse necessita utilizar integralmente todos os comportamentos do método da superclasse, e depois realizar seus passos específicos.

6.2 **A associação**

A associação é um conceito que permite classes e objetos se relacionarem entre si para conseguirem representar de forma mais expressiva e objetiva o conceito que querem.

Por exemplo: No exemplo do hospital queremos que o medico e o anestesista tenha seu endereço informado, para isso podemos fazer com que ambos herdem de uma Classe Endereço, no entanto devemos nos perguntar um Médico/anestesista é um endereço? Obviamente que não, por isso seria um erro conceitual fazer esse tipo de relação. Para suprir essa necessidade, a classe Medico e a Anestesista podem se relacionar com a classe Endereço de uma maneira que não seja a herança para suprir

essa carência, a essa relação damos o nome de associação. Sendo assim, ambos se complementarão para a representação do problema.

Tal ação, aumenta a coesão do código (um dos princípios da OO), pois ao invés de jogarmos valores dentro das classes Anestesista e Medico, criamos uma classe separada para colocar os valores e relacionar-se com ambas. Ademais, não apenas na criação conceitual pode ser usada a associação, como também na criação de comportamentos específicos das classes. Como por exemplo, a associação da classe Parto com Médico, se as atividades de Parto forem definidas na própria classe vai ser criado uma classe não coesa,acaba que o correto é defini-la dentro da classe Médico especifico, associa-la com a classe Parto e definir as atividades específicas de cada médico.

Mas o que possibilita o conceito de associação? O que faz a intercalação entre classes nada mais é do que o conceito de mensagem que já foi estudado em capítulos e explicado em resumos anteriores.

Os tipos de associação: agregação, dependência e composição

A associação pode ser feita de duas formas: Estrutural e comportamental.

Estrutural possui dois tipos, agregação e composição, a associação estrutural ocorre na estrutura de dados da classe, em seus atributos. Sendo assim, um dos atributos da classe é da classe associada como no exemplo anterior de endereço e médico. A associação de composição ocorre quando um relacionamento da “parte do todo” ocorre. Esse relacionamento é caracterizado por uma das partes não poder existir sem a outra, como por exemplo, um endereço não pode existir sem um médico, pois não teria finalidade nenhuma ele existir sem pertencer a um médico ou a empresa O endereço não existe sem o “todo” o médico.

Já na associação estrutural de agregação, ocorre quando a relação “parte com o todo” não existe, ou seja, o seu valor pode ser compartilhado com vários objetos diferentes, a exemplo da sala de operação, essa pode ser compartilhada entre diferentes médicos, ao contrário do endereço que cada médico possuía o seu e apenas o seu. Assim, a sala pode pertencer a dois “todos distintos”, sendo assim, a sala se agrega aos elementos, mas não compõe única e exclusivamente apenas um deles.

Já a associação comportamental (ou dependência) se configura por que muitas vezes necessitamos passar para um método objetos como parâmetros para seu funcionamento, ao utilizarmos desse recurso acabamos por ter acesso aos membros desses atributos/classes para ajudar o método a realizar seus comportamentos, isso é um processo de associação, no entanto, não está ligado a estrutura da classe/objeto. Essa associação faz parte do método.

As características de uma associação: unária, múltipla, cardinalidade e navegabilidade

Essas características visam auxiliar as associações e o seu entendimento. Para explicar os conceitos é dado o exemplo de um beneficiário de plano de saúde que o usa para cobrir suas necessidades médicas. No entanto, é muito comum que ele possua dependentes, como uma mãe que paga o plano de saúde tanto dela quanto do filho, ambos são beneficiários, no entanto, o filho depende da mãe. Sendo assim, na classe beneficiário é possível criar um atributo para diferenciar se a pessoa é o dependente ou o titular, o mesmo tem o tipo sendo o mesmo da classe Beneficiário. Foi usado apenas o tipo da própria classe, isso é chamado de associação unária.

Já no exemplo da classe Parto dado anteriormente, existe uma associação múltipla pois utiliza o atributo da sala e o vetor de médicos, sendo assim utilizadas duas classes diferentes na associação, caracterizando a associação múltipla. Além disso, a quantidade de salas e médicos envolvido na associação é conhecida como "cardinalidade", sendo essas fixas, mas também poderiam ser indefinidas se necessário.

Por último, veremos o conceito de navegabilidade, ela pode ser unidirecional ou bidirecional. A unidirecional acontece quando apenas uma das classes recebe atributos do tipo da outra classe, a bidirecional acontece quando ambas possuem atributos dos tipos das outras. Por exemplo, a classe Parto possui o atributo do tipo sala, no entanto, sala não tem nenhum atributo do tipo parto, logo é unidirecional.

6.3. A interface

A interface é uma lista de comportamentos que ao ser implementada pela classe, a mesma dita obrigatoriedade a classe cumprir todos os comportamentos abstratos disponibilizados pela interface. Um exemplo de implementação disto no mundo real é o do ministério da saúde cobrar a prestação de contas do hospital, por conta de o

ministério precisar dessas informações e não saber como obtê-las, sendo assim o ministério da saúde tem que disponibilizar uma lista de métodos (interface), para que o hospital seja obrigado e tenha como fornecer as informações.

Por fim, não importa como foi feito pelo hospital a operacionalização do trabalho e sim seus resultados, ou seja, se outro hospital for cumprir a mesma lista, pode cumprir de forma completamente diferente do primeiro hospital, no entanto, a interface foi suprida. A seguir o capítulo mostra a sintaxe de algumas linguagens de programação quanto a declaração de interfaces e algumas boas condutas. Além disso, é possível atribuir múltiplas interfaces a classes, bastante usar uma vírgula. A diferença entre herança e interface, é que na herança podemos reutilizar métodos com comportamentos definidos, no entanto na interface, só nos limitamos a assinatura dos métodos, pois todos os métodos da interface são abstratos.

6.4 Resumindo

Este ponto resume o que foi abordado durante o capitulo, e lista os conceitos sobre OO.

6.5 Para o refletir

1. O que é e qual a finalidade da herança?

É a relação de transferência de atributos de uma classe (super classe) para outra classe (subclasse), tal relação ajuda na representação do que é o mundo real.

2. Por que a herança só pode ser feita entre classes?

Pois as classes são feitas e definidas durante a compilação, por exemplo, os objetos não podem herdar pois só são utilizados durante a execução do código.

3. Discorra sobre a frase: "A herança tem como finalidade prover o reúso."

O reuso é intrinsicamente ligado a ideia de herança e a abstração, ao definir uma classe de maneira abstrata é por que precisamos reutilizar seus componentes, logo o conceito de herança provê reuso.

4. Explique generalização e especialização.

Generalização é quando subimos na hierarquia de classes, e a classe vai ficando cada vez mais genérica, já a especialização são as especificações que a classe vai

ganhando ao ser acoplada a ela os processos de herança, fazendo com que o objeto seja cada vez mais preciso.

5. Explique o que são classes abstratas e concretas.

Classes abstratas são classes que não se trabalham diretamente e não são instanciadas, elas servem para o processo de abstração por serem a classe que define as características mais básicas das futuras classes.

Classes concretas são classes como já conhecemos, se operacionalizam e são criados objetos a partir delas.

6. Como funcionam as heranças simples e múltipla?

A herança simples ocorre quando uma classe herda característica apenas de uma superclasse, já a herança múltipla acontece quando uma subclasse herda características de duas ou mais classes simultaneamente.

7- Explique o mecanismo de downcast e upcast. Quais os cuidados em relação ao uso do downcast?

Upcast é a operação de promoção de subclasses a superclasses na hierarquia de classes, downcast é o contrário. No entanto, downcast é um processo que deve ser desencorajado pois gera problemas durante o código, tal qual o exemplo do gerente dado no livro.

8. O que é qual a importância do polimorfismo?

Poliformismo é a capacidade de adequar determinado método para diferentes contextos na programação, a vantagem do mesmo é que diferentes objetos acabam executando uma mesma ação, podendo assim ser manipulados com maior facilidade e ajudando na coesão do código.

9. O que é sobrescrita e qual sua relação com poliformismo?

Sobrescrita é quando um método é redefinido durante o código, sua relação com poliformismo pois o poliformismo é feito através da rescrita do código preexistente o adaptando a um contexto x ou y.

10. O que é associação?

Associação é um tipo de relacionamento entre classes para atribuir atributos para uma ou entre si.

11. Qual a diferença entre associação e herança?

A diferença se da em um meio conceitual, pois ao definimos herança chegamos a pergunta chave “Uma coisa é outra coisa?”, e se a resposta for sim usamos tal conceito, no entanto, existem contextos em que essa pergunta recebe uma negação mas mesmo assim queremos passar o atributo de uma classe a outra, usar a herança seria um erro conceitual, por isso usamos a associação no lugar.

12. Quais são os tipos de associação?

Agregação, composição e dependência

13. O que é e como funciona uma interface?

A interface é uma lista de comportamentos obrigatórios que uma classe tem que cumprir não importando o método. A interface funciona a partir da obrigatoriedade que uma classe recebe sendo a interface a definição de métodos abstratos.

14. Qual a relação entre interface e classe abstrata?

A interface se comporta como uma classe abstrata, pois a mesma tem seus métodos apenas definidos, ou seja, abstratos.

7 - Os conceitos organizacionais

Os conceitos organizacionais são responsáveis por aglutinar classes que sejam similares, tais conceitos limitam também acessos a membros das classes, organizando seu uso dentro do código.

7.1 Pacotes

Ao fazermos um sistema, é comum que exista dezenas ou centenas de classes para representar o nosso problema, no entanto, isso torna o código difícil de se entender e de procurar alguma desejado se deixarmos tudo junto, para isso existem os pacotes.

Os pacotes nada mais são do que a separação de classes com finalidades diferentes, assim é possível deixar a aplicação mais organizada e torna possível separar as classes de finalidades e representações diferentes.

A seguir o livro demonstra como fazer a codificação e sintaxe para criação de pacotes em JAVA e em C, além disso é mostrado que é possível fazer subpacotes (pacotes dentro de pacotes) para mais organização. Sendo assim, após a demonstração vemos que são organizadas pastas no sistema operacional para a organização física das classes, além disso também pode ser feita organização lógica das mesmas.

7.2 Visibilidades

São responsáveis por determinar até onde o acesso as classes, tributos e métodos podem ser utilizados. Sua utilização é imprescindível na OO, diversas boas práticas e conceitos só são atingidos se utilizado elas. Também são chamados de modificadores de acesso. Existem 3 diferentes tipos de visibilidade, a privada, a publica e a protegida, no entanto, nem todas as linguagens se utilizam desses conceitos, por exemplo, em python todos são públicos apenas.

A visibilidade privada define que os atributos definidos só poderão ser manipulados dentro de onde foram declarados, ou seja, se definirmos um membro assim, ele só poderá ser manipulado na classe em que se foi definido. Se tentarmos acessar os dados por outro local que não seja da própria classe será apresentada uma mensagem e erro. A seguir são mostrados exemplos de código de implementação dessa visibilidade.

A visibilidade protegida ou intermediaria, define que os atributos e métodos só podem ser acessados na classe em que foram definidos e pelas classes que herdam a partir desta, ou seja, se for feito assim, só poderão ser utilizados na classe e subclasses dessa classe. A seguir são mostrados exemplos de código de implementação dessa visibilidade.

A visibilidade publica tem todos os membros definidos com ela utilizáveis em qualquer parte do código, independente do relacionamento entre as classes. N entanto, utiliza-la pode resultar em acessos indevidos de atributos e uso indevido de métodos. A seguir, o livro mostra codificações com o uso da visibilidade pública.

7.3 Resumindo

Neste capitulo o autor fala um pouco sobre ter chegado a reta final das explicações conceituais.

7.4 Para refletir.

1 - O que são e para que serve os pacotes?

Os pacotes são as separações de classes com finalidades diferentes, eles servem para organizar a aplicação e tornar possível a separação de classes com representações e finalidades disjuntas.

2 – O que são e para que servem os modificadores e visibilidade?

Os modificadores de visibilidade são responsáveis por determinar até onde o acesso as classes, tributos e métodos podem ser utilizados. Servem para determinar onde certos métodos e atributos devem ser utilizado durante o código.

3- Quais são os tipos de visibilidade e como cada uma opera?

Visibilidade privada: Este tipo de visibilidade faz com que o que seja especificado com ela só possa ser trabalhado e utilizado dentro da classe em que foi definida

Visibilidade intermediaria: Este tipo de visibilidade faz com que o que seja especificado com ela possa ser trabalhado tanto na classe em que foi definido quanto nas suas subclasses.

Visibilidade publica: Este tipo de visibilidade faz com que o que seja especificado por ela possa ser trabalhado em qualquer parte do código.

4- Por que a visibilidade private pode ser considerada a mais importante?

Pois apenas com ela alguns conceitos centrais da OO podem ser implementados e cumpridos.

5- Qual a finalidade de definir um método private?

Fazer com que o mesmo só possa ser usado dentro de onde foi definido.