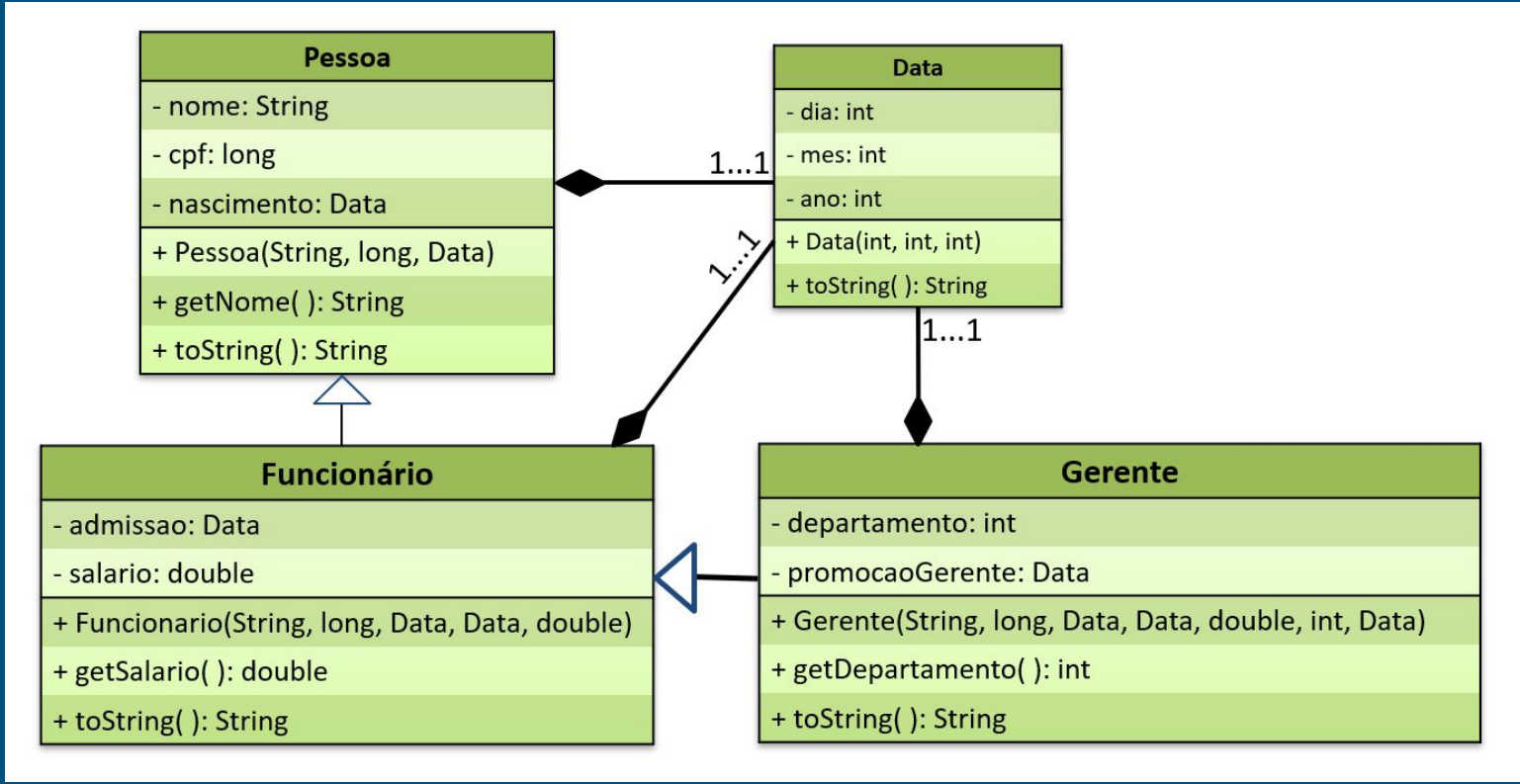
CC3642
Orientação a Objetos

Prof. Danilo H. Perico

Herança

Exercício 1 - Implemente no Java



*Atenção para o que é **herança** e o que é **composição**!*

Exercício 1 - Implemente no Java

No método *main* instancie alguns funcionários e alguns gerentes em dois *ArrayLists* distintos. Chame, por meio dos objetos que estão dentro dos *ArrayLists*, os métodos de ambos para testar.

Exercício 2

(Deitel 9.3) Muitos programas escritos com herança podem ser escritos com composição e vice-versa. Reescreva a classe *BasePlusCommissionEmployee* da hierarquia *CommissionEmployee*–*BasePlusCommissionEmployee* para utilizar composição em vez de herança (projeto com Herança disponível no Moodle).

Depois de fazer o código, avalie os méritos relativos das duas abordagens. Que abordagem é mais natural? Por quê?

Exercício 3

(Deitel 12.10) (Hierarquia de herança de Account) Crie uma hierarquia de herança que um banco possa utilizar para representar as contas bancárias dos clientes. Todos os clientes nesse banco podem depositar (isto é, creditar) dinheiro em suas contas e retirar (isto é, debitar) o dinheiro delas. Há também tipos mais específicos de contas. As contas de poupança, por exemplo, recebem juros pelo dinheiro depositado nelas. As contas correntes, por outro lado, cobram uma taxa por transação (isto é, crédito ou débito).

Crie uma hierarquia de herança contendo a superclasse *Account* e as subclasses *SavingsAccount* (Poupança) e *CheckingAccount* (Conta Corrente) que herdam da classe *Account*. A superclasse *Account* deve incluir um membro de dados do tipo double para representar o saldo da conta.

Exercício 3

A classe deve fornecer um construtor que recebe um saldo inicial e o utiliza para inicializar o membro de dados. O construtor deve validar o saldo inicial para assegurar que ele é maior que ou igual a 0.0. Caso contrário, o saldo deve ser configurado como 0.0 e o construtor deve exibir uma mensagem de erro, indicando que o saldo inicial era inválido. A classe deve fornecer três métodos. O método *credit* deve adicionar uma quantia ao saldo atual. O método *debit* deve retirar dinheiro de *Account* e assegurar que o valor do débito não exceda o saldo de *Account*. Se exceder, o saldo deve permanecer inalterado e a função deve imprimir a mensagem “Saldo insuficiente”. O método *getBalance* deve retornar o saldo atual. A subclasse *SavingsAccount* deve herdar a funcionalidade de uma *Account*, mas também incluir um membro de dados do tipo double para indicar a taxa de juros (porcentagem) atribuída à *Account*.

Exercício 3

O construtor *SavingsAccount* deve receber o saldo inicial, bem como um valor inicial para a taxa de juros de *SavingsAccount*. *SavingsAccount* deve fornecer um método *public calculateInterest* que retorna um double para indicar os juros auferidos por uma conta. O método *calculateInterest* deve determinar esse valor multiplicando a taxa de juros pelo saldo da conta. [Nota: SavingsAccount deve herdar as funções-membro *credit* e *debit* exatamente como são sem redefini-las.] A subclasse *CheckingAccount* deve herdar da classe básica *Account* e incluir um membro adicional de dados do tipo double que representa a taxa cobrada por transação. O construtor *CheckingAccount* deve receber o saldo inicial, bem como um parâmetro que indica o valor de uma taxa. A classe *CheckingAccount* deve redefinir as funções-membro *credit* e *debit* para que subtraiam a taxa do saldo da conta sempre que qualquer uma das transações for realizada com sucesso.

Exercício 3

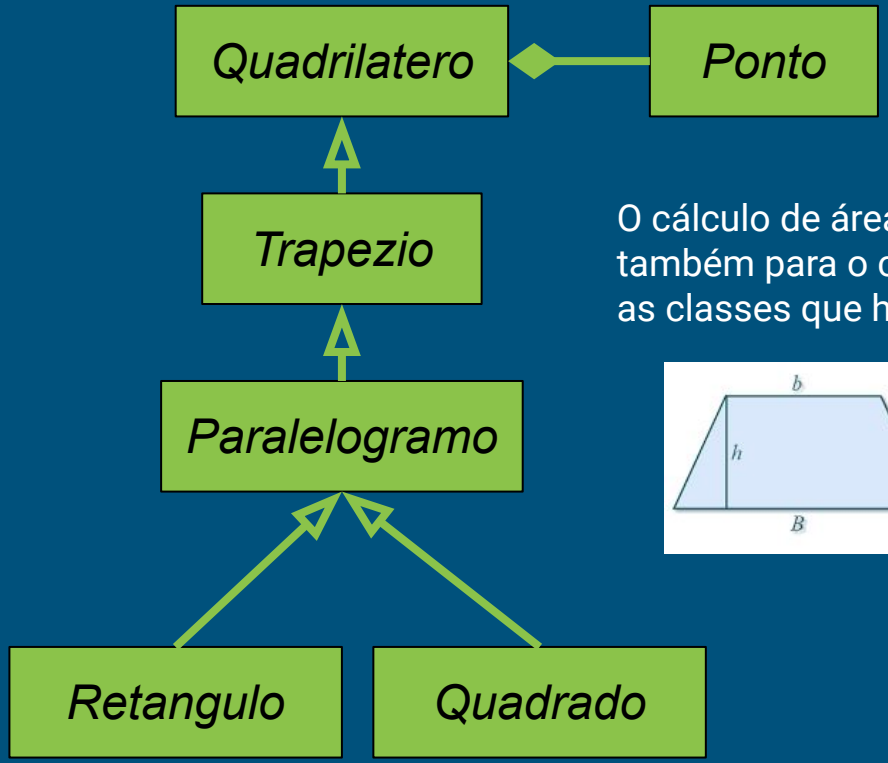
As versões *CheckingAccount* dessas funções devem invocar a versão *Account* da classe básica para realizar as atualizações de saldo de uma conta. A função *debit* de CheckingAccount deve cobrar uma taxa somente se o dinheiro for realmente retirado (isto é, o valor do débito não exceder ao do saldo da conta).

Dica: Defina a função debit de Account para que ela retorne um bool indicando se houve retirada de dinheiro. Em seguida, utilize o valor de retorno para determinar se uma taxa deve ser cobrada.

Exercício 4 - Implemente no Java

(Deitel 9.8) Escreva uma hierarquia de herança para as classes *Quadrilatero*, *Trapezio*, *Paralelogramo*, *Retangulo* e *Quadrado*. Use *Quadrilatero* como superclasse da hierarquia. Crie e use uma classe *Ponto* para representar os pontos (x, y) de cada forma. Faça a hierarquia o mais profunda (isto é, com muitos níveis) possível. Especifique as variáveis de instância e os métodos para cada classe. As variáveis de instância *private* de *Quadrilatero* devem ser *Pontos* para os quatro pontos que delimitam o quadrilátero. Escreva um programa que instancia objetos de suas classes e gera saída da área de cada objeto (exceto *Quadrilatero*). A entrada será feita com a posição de 4 pontos (x, y).

Exercício 4 - Implemente no Java



O cálculo de área do trapézio serve também para o cálculo de área de todas as classes que herdam de Trapézio