Fundamentos de C#

Exercícios Propostos

Polimorfismo

1 Exercício

Crie a classe *Figura* que representa figuras geométricas, representadas pelas classes *Quadrado* e *Retangulo*. Uma figura pode ter sua área calculada a partir do método *CalcularArea()*, que retorna a área calculada da figura em forma de um *double*.

Crie também a classe *FiguraComplexa*. Uma figura complexa é também uma figura, mas a diferença é que ela é composta por várias figuras (quadrados, retângulos ou até outras figuras complexas). Para calcular a área de uma figura complexa, basta somar a área de todas as figuras que a compõem.

Para executar a aplicação, crie a classe *Program*, que é responsável por criar uma figura complexa e calcular a sua área. Esta figura deve ser composta por:

- 1 quadrado com 3 de lado
- 1 quadrado com 10 de lado
- 1 retângulo com lados 2 e 7
- 1 retângulo com lados 5 e 3

Dica 1: Perceba a diferença entre uma classe ser uma figura e ter uma ou mais figuras. A primeira relação é de herança, enquanto a segunda implica em uma composição.

Dica 2: Para resolver este exercício é necessário conhecer a respeito de arrays. Se você não está familiarizado com eles neste momento, pode voltar a este exercício mais adiante, depois que este tema for abordado no curso.

2 Exercício

Implemente a classe *Colecao* e duas subclasses: *Pilha* e *Fila*. Uma coleção tem um *array* de dados que fazem parte da coleção.

Tanto a pilha como a fila são coleções. A diferença entre elas está na disciplina de acesso. Na pilha, o último elemento inserido é o primeiro a ser removido (como numa pilha de pratos). Na fila, o primeiro elemento inserido é o primeiro a ser removido (como numa fila de banco).

Os métodos da classe *Colecao* responsáveis por estas operações são:

- *void InserirItem(object item)*
- *object RemoverItem()*

Crie um método que recebe uma coleção, adiciona alguns elementos e remove estes mesmo elementos. Imprima os elementos removidos e veja a diferença no resultado.

Dica: Para resolver este exercício é necessário conhecer a respeito de arrays. Se você não está familiarizado com eles neste momento, pode voltar a este exercício mais adiante, depois que este tema for abordado no curso.

3 Exercício

Crie uma classe *Veiculo* com um field *ligado* (privado), que indica se o carro está ligado ou não. Esta classe deve ter também os métodos *Ligar()* e *Desligar()*, que definem o valor para este field, e uma read-only property para retornar o valor do field.

Depois crie três subclasses de *Veiculo*: *Automovel*, *Motocicleta* e *Onibus*. Cada classe destas deve sobrescrever os métodos *Ligar()* e *Desligar()* e deve imprimir mensagens como "*Automóvel ligado*", "*Motocicleta desligada*", etc. Para manter a consistência do modelo, descubra como fazer para que o field ligado de *Veiculo* tenha o valor correto quando os métodos são chamados.

Crie uma aplicação que instancia três veículos, um de cada tipo, e chama os métodos *Ligar()*, *Desligar()* e a property *Ligado*. O resultado obtido deve ser consistente com o que o modelo representa. Por exemplo, ao chamar o método *Ligar()* de um *Automovel*, é esperado que a property *Ligado* retorne *true*.

4 Exercício

Crie uma interface *IAreaCalculavel* com um método *CalcularArea()* e crie classes de figuras geométricas que implementam este método (como quadrado, circunferência e retângulo). Depois crie uma classe com um método *Main()* para exercitar as chamadas aos métodos que calculam a área.

5 Exercício

Crie uma classe *ContaBancaria*, que representa uma conta bancária genérica que não pode ser instanciada. Esta classe deve ter uma property *Saldo* (visível apenas para ela e para as suas subclasses) e os métodos *Depositar(double)*, *Sacar(double)* e *Transferir(double, ContaBancaria)*. Estes métodos devem depositar um valor na conta, sacar um valor da conta e transferir um valor da conta de origem para uma conta de destino, respectivamente.

Além destes, *ContaBancaria* deve ter um método *CalcularSaldo()*. Este método possui a regra do cálculo do saldo final (que pode ser diferente do saldo armazenado em *Saldo*) e deve ser obrigatoriamente implementado pelas subclasses de *ContaBancaria*, pois cada classe possui suas próprias regras de cálculo.

Crie duas subclasses de *ContaBancaria*: *ContaCorrente* e *ContaInvestimento*. Cada uma deverá implementar suas regras para calcular o saldo (método *CalcularSaldo()*). No caso de *ContaCorrente*, o saldo final é o saldo atual subtraído de 10%, referente a impostos que devem ser pagos. Já para a *ContaInvestimento*, o saldo final é o saldo atual acrescido de 5%, referente aos rendimentos do dinheiro investido.

Crie uma aplicação que instancia uma conta corrente e uma conta investimento e executa as operações de depósito, saque, transferência e cálculo de saldo. Verifique se os resultados obtidos são consistentes com a proposta do modelo e com as regras de cálculo estabelecidas.