Bootcamp Speed Wiz

Desafio

Módulo 2	Fundamentos de C#

Objetivos

Exercitar os seguintes conceitos trabalhados no Módulo:

- ✔ Fixar a aprendizagem referente a tipos, valores, variáveis e constantes.
- ✔ Criar estruturas básicas de decisão e repetição.
- ✔ Praticar os conceitos de orientação a objetos

Enunciado

O desafio final consiste em construir uma aplicação C# que simula um sistema bancário com gerenciamento de múltiplas contas com a habilidade de realizar operações de depósito, saque, transferência entre contas. Estas também estarão aptas a imprimir o saldo e outros detalhes no terminal. O objetivo é utilizar os conceitos da orientação por objetos para reforçar elementos como listas, classes, herança e polimorfismo.

Atividades

Os alunos deverão desempenhar as seguintes atividades:

1. Crie um novo projeto no Visual Studio para construir uma aplicação que simule um sistema bancário.

2. Os artefatos produzidos para essa solução serão (Conta.cs, Transacao.cs, SaqueTransacao.cs, DepositoTransacao.cs, TranferenciaTransacao.cs, Banco.cs e Program.cs).

3. Para a classe **Conta** que representa uma versão simples de uma conta bancária possuímos os seguintes membros:

   a. **Campos:** *saldo* (**decimal**) e *nome* (**string**) para representar o nome do cliente.

   b. **Propriedades:** *Saldo* (**decimal**) e *Nome* (**string**) somente leitura encapsulando os atributos.

   c. **Construtor** recebe os parâmetros *saldo* (**decimal**) e *nome* (**string**), o momento da construção do objeto será a única forma de inicializar os atributos supracitados.

   d. **Métodos:**

      i. **Deposito** recebe um **parâmetro** *valor* (**decimal**), realiza validações para que ocorram depósitos válidos, ou seja, não existem depósitos com valores negativos, e retorne um valor booleano indicando que a transação foi realizada com sucesso.

      ii. **Saque** recebe um parâmetro *valor* (**decimal**), realiza validações para as operações de saques (não sendo possível se este for maior que o saldo disponível e o valor inferior a 0, e retorne um valor do tipo **bool** indicando que a transação foi realizada com sucesso.

      iii. **Imprimir** para exibir o nome do proprietário da conta e o saldo disponível.

4. Para **Transacao** teremos uma classe abstrata que representará o topo da herança para as demais transações, lembrando que uma classe abstrata não atua como interfaces pois contém implementações, então teremos:

a. **Campos**: *valor* (**decimal**) representando o valor para a transação, este será um atributo protegido (**protected**) visível somente para as classes-filha; *executado* (**bool**) para representar que uma transação foi executada, ela também será uma forma de controlar para que a mesma transação seja executada uma única vez; *revertido* (**bool**) para indicar que em caso de falha a operação possa ser revertida e seja também executada uma única vez.

b. **Propriedades:** *Executado* (**bool**), *Revertido* (**bool**) encapsulando os atributos supracitados, e uma propriedade abstrata *Sucesso* (**abstract**), responsável por indicar o status da transação.

c. **Construtor:** recebe o parâmetro *valor* (**decimal**) indicando um valor para a transação, o momento da construção do objeto atual será a única forma de inicializar o atributo valor.

d. **Métodos**:

   i. **Imprimir** é um método abstrato que é implementado pelas classes que herdam.

   ii. **Executar** é um método virtual que será sobrescrito pelas classes filhas, este deve verificar se a transação já foi realizada e levantar uma exceção alertando o cliente

   de que a transação não pode ser executada novamente, caso contrário definimos como uma transação executada.

   iii. **Reverter** é um método virtual que será sobrescrito pelas classes filhas, este deve verificar se a transação ainda não foi realizada, pois não há como reverter uma transação que ainda não foi executada. Este método também verifica se transação já foi desfeita para evitar que ela seja executada mais de uma vez, caso contrário definimos como uma transação revertida.

5. A classe **SaqueTransacao é** uma herança da classe abstrata Transacao que representa as operações de saque. Essa é constituída dos seguintes membros:

a. **Campos:** *Conta* (**Conta**) representando a conta a qual será aplicada a operação de saque, *valor* (**decimal**) representando o valor a ser sacado.

b. **Propriedades:** A propriedade *Success* (**bool**) deve ser implementada, pois é um dos membros obrigatórios devido a classe base e essa representa o status da transação propriamente.

c. **Construtor:** recebe o parâmetro *conta* (**Conta**) indicando a conta para a transação e o *valor* (**decimal**) representando a quantia atual, o momento da construção do objeto atual será a única forma de inicializar os atributos.

d. **Métodos:**

   i. **Executar** é o método responsável pela execução da operação, mas é importante lembrar que as implementações da classe base precisam ser estendidas e dessa forma a sobrescrita será o mecanismo mais útil, mas não devemos descartar o que a classe ancestral fornece, pois faz parte da validação da operação. (Considere utilizar a operação de saque da instância da Conta neste método também).

   ii. **Reverter** é o método responsável pela reversão da operação, mas é importante lembrar que as implementações da classe base precisam ser estendidas e dessa forma a sobrescrita será o mecanismo mais útil, mas não devemos descartar o que a classe ancestral prove pois faz parte da validação da operação. (Considere utilizar a operação de depósito da instância da Conta neste método também, isso reverterá a operação).

   iii. **Imprimir** deve ser implementado para exibir as informações pertinentes à operação.

6. A classe **DepositoTransacao** é uma herança da classe abstrata Transacao que representa as operações de depósito, essa é constituída dos seguintes membros:

a. **Campos:** *Conta* (**Conta**) representando a conta a qual será aplicada a operação de saque, *valor* (**decimal**) representando o valor a ser sacado.

b. **Propriedades:** A propriedade *Success* (**bool**) deve ser implementada, pois é um dos membros obrigatórios devido a classe base e essa representa o status da transação propriamente.

c. **Construtor:** recebe o parâmetro *conta* (**Conta**) indicando a conta para a transação e o *valor* (**decimal**) representando a quantia atual, o momento da construção do objeto atual será a única forma de inicializar os atributos.

d. **Métodos:**

   i. **Executar** é o método responsável pela execução da operação, mas é importante lembrar que as implementações da classe base precisam ser estendidas e dessa forma a sobrescrita será o mecanismo mais útil, mas não devemos descartar o que a classe ancestral fornece pois faz parte da validação da operação. (Considere utilizar a operação de depósito da instância da Conta neste método também).

   ii. **Reverter** é o método responsável pela reversão da operação, mas é importante lembrar que as implementações da classe base precisam ser estendidas e dessa forma a sobrescrita será o mecanismo mais útil, mas não devemos descartar o que a classe ancestral prove pois faz parte da validação da operação. (Considere utilizar a operação de saque da instância da Conta neste método também).

   iii. **Imprimir** deve ser implementado para exibir as informações pertinentes à operação.

7. A classe **TransferenciaTransacao é** uma herança da classe abstrata Transacao que representa as operações de transferência, essa é constituída dos seguintes membros:

a. **Campos:** *ContaDe* (**Conta**) que representa a conta origem da operação de transferência, *ContaPara* (**Conta**) que representa a conta destino na operação de transferência, *valor* (**decimal**) representando o valor a ser transferido, *deposito*

(**DepositoTransferencia**) para a operação de depósito na conta destino e *saque* (**SaqueTransferencia**) para a operação de saque na conta origem.

b. **Propriedades:** A propriedade *Success* (**bool**) deve ser implementada, pois é um dos membros obrigatórios devido a classe base e essa representa o status da transação propriamente.

c. **Construtor:** recebe o parâmetro *contaDe* (**Conta**) indicando a conta origem, *contaPara* (**Conta**) indicando a conta destino e o *valor* (**decimal**) representando a quantia atual; inicialize os atributos *deposito* (**DepositoTransferencia**) e *saque* (**SaqueTransferencia**) com as respectivas contas e quantia, este é o momento da construção do objeto atual e será a única forma de inicializar os atributos.

d. **Métodos:**

   i. **Executar** é o método responsável pela execução da operação, mas é importante lembrar que as implementações da classe base precisam ser estendidas e dessa forma a sobrescrita será o mecanismo mais útil, mas não devemos descartar o que a classe ancestral fornece pois faz parte da validação da operação. Considere utilizar a operação de saque da conta de origem e se a operação for realizada com sucesso prossiga para a operação de depósito na conta de destino , se essa falhar reverta a operação.

   ii. **Reverter** é o método responsável pela reversão da operação, mas é importante lembrar que as implementações da classe base precisam ser estendidas e dessa forma a sobrescrita será o mecanismo mais útil, mas não devemos descartar o que a classe ancestral prove, pois faz parte da validação da operação. Considere utilizar a operação reversão para ambas as contas.

   iii. **Imprimir** deve ser implementado para exibir as informações pertinentes à operação.

8. A classe **Banco** é a classe que envolve os principais elementos da solução, como por exemplo manter as transações e contas :

a. **Campos:** contém uma lista de *contas* (**List<Conta>**) e uma lista de *transacoes* (**List<Transacao>**).

b. **Construtor:** inicializa as listas supracitadas.

c. **Métodos:**

   i. **AdicionarConta** é o método responsável por adicionar uma nova conta à lista de contas e recebe como parâmetro uma nova *conta* (**Conta**).

   ii. **RecuperarConta** é o método responsável por recuperar uma conta, neste caso utilizando o nome do cliente, desta forma cria uma possibilidade de trabalhar com uma conta em específico, o retorno é consequentemente a conta encontrada.

   iii. **ExecutarTransacao** é o método responsável por adicionar uma nova transação e consequente executá-la.

9. A classe **Program** é responsável por trabalhar com as demais classes através da instância de banco (**Banco**) é:

a. **Método Main:**

   i. Crie uma instância de *banco* **(Banco).**

   ii. Crie um menu com as seguintes opções**:**

      1. Nova Conta
      2. Saque
      3. Deposito
      4. Transferência
      5. Imprimir
      6. Sair

Utilize estruturas de repetição para deixar o menu fluídico, com a sensação de que o sistema está em execução contínua, para cada uma das opções acima utilize métodos.

Respostas Finais

Os alunos deverão desenvolver a prática e, depois, responder às seguintes questões objetivas: