Bootcamp Speed Wiz

Trabalho Prático

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 trabalho prático consiste em construir uma aplicação C# para gerenciamento de contas bancárias. O objetivo é realizar o controle das contas dos clientes através de construções de classes, adicionando comportamentos e métodos para fixar os conceitos fundamentais necessários para todo desenvolvedor e desenvolvedora C#.

Atividades

Os alunos deverão desempenhar as seguintes atividades:

1. Crie um novo projeto no Visual Studio para construir a aplicação de gerenciamento bancário.

2. Crie a classe **Conta** que representa uma conta bancária e possui os seguintes campos: **numeroDaAgencia (int)**, **numeroDaConta (int)**, **nomeDoTitular (string)** e **saldo da conta (decimal)**. Ao criar um atributo lembre-se de informar o tipo, nome

e modificador de acesso. Na classe Conta, crie um construtor para inicializar os atributos.

3. Construa os seguintes métodos na classe Conta:

   a. **Sacar**: Método público que recebe como argumento o valor a ser retirado **valorSaque (decimal)**.

      Este método deve permitir o saque apenas se o saldo for maior ou igual ao valor do saque. Neste caso, deve subtrair o valor do saque ao valor do saldo, o método deve retornar um status de verdadeiro ou falso indicando se a operação foi realizada com sucesso. Lembrando que deveremos verificar se o valor do saque é maior que 0.

   b. **Depositar**: Método que recebe como argumento o valor a ser incrementado **valorDeposito (decimal)**. O método deve retornar um status de verdadeiro ou falso indicando se a operação foi realizada com sucesso. Lembrando que deveremos consistir em valores inferiores a zero, pois não há depósitos com valores negativos ou iguais a 0.

   c. **Transferir**: Método que recebe o valor da transferência **valorSaque (decimal)** e a conta de destino **contaDestino (Conta)** para a qual será enviado o valor.

      Este método deve permitir a transferência apenas se o saldo da conta de origem for maior ou igual ao valor a ser transferido. O método deve retornar um status de verdadeiro ou falso indicando se a operação foi realizada com sucesso.

4. Crie a classe **Cliente** contendo os campos: **nome (string)**, **rg (string)**, **cpf(string)** e **endereço(string)**. Modifique a classe **Conta** e faça com que seu atributo **nomeDoTitular** seja do tipo Cliente ao invés de string. Na classe Cliente crie um construtor para inicializar os atributos.

5. Crie um método **SimulaInvestimento** para permitir que o cliente do banco possa saber quanto ele ganhará, ao final de 1 ano, caso ele invista um valor

(valorInvestido (decimal)). O investimento paga 1% do valor investido ao mês. Utilize a instrução **for** para calcular o valor final.

6. Contas de investimento e contas de poupança possuem tributações específicas do governo. Crie as classes **ContaPoupanca** e **ContaInvestimento** que herdam da classe **Conta**. Crie um método void **CalcularTributo** em ambas as classes que calcula o valor desse tributo. Para cada caso, coloque um valor diferente para a taxa de tributação. Não esqueça de encapsular o atributo de saldo em uma propriedade para que essa fique disponível no cálculo do método. Por exemplo, para o caso de 2% ao ano seria:

```
public decimal CalcularTributo() {
return Saldo * 0.02;
}
```

7. Para todas as classes dos itens anteriores, crie instâncias e realize testes com os métodos criados.

Respostas Finais

Para fixar seu conhecimento, abaixo são apresentadas questões baseadas no conteúdo do módulo e no trabalho prático. Os alunos deverão desenvolver a prática e, depois, responder às seguintes questões objetivas.