LP3 – Exercícios

Aula 5 - Programação Orientada a Objetos com C#

Instruções para entrega das listas de exercícios:

Meio de Entrega: As resoluções das listas de exercícios devem ser entregues exclusivamente por meio do ambiente Moodle (http://eadcampus.spo.ifsp.edu.br). Usar o mesmo usuário e senha do Sistema Aurora.

Forma de Entrega: Para exercícios com implementação de programas console, devem ser entregues os arquivos das classes (extensão CS). Para exercícios com implementação de programas Windows Forms, devem ser entregues as pastas dos projetos que contêm as aplicações desenvolvidas. Para exercícios com banco de dados, também devem ser entregues as instruções SQL usadas (extensão SQL ou TXT). Todos os arquivos da lista devem ser compactados em um único arquivo (extensão RAR ou ZIP), cujo nome deverá conter a aula, o nome e um sobrenome do aluno. Por exemplo: Aula2_JoaoSilva.zip.

Prazo de Entrega: O prazo de entrega está definido na própria página de exercícios do Moodle, lembrando que o sistema bloqueia o envio de arquivos após a data e horário indicados.

Obs.: O material da disciplina e as listas de exercícios estão disponíveis no ambiente Moodle.

1. Faça um programa em C# que apresente ao usuário as opções a seguir, enquanto ele não digitar a opção 0 (zero). De acordo com o número da opção informada, o programa deverá efetuar a operação, solicitando as informações ao usuário quando necessário. Os dados dos clientes devem ser armazenados em dois vetores de objetos, um do tipo PessoaFisica[10], que armazenará objetos contendo código, endereço, telefone, nome, CPF dos clientes, e um vetor do tipo PessoaJuridica[10], que armazenará objetos contendo código, endereço, telefone, razaoSocial e CNPJ dos clientes. Seguem as opções:
   1) **Inserir cliente** – Primeiramente, pergunte ao usuário se o cliente é pessoa física (PF) ou jurídica (PJ). Para pessoa física, solicite o nome, o CPF, o endereço e o telefone. Para pessoa jurídica, solicite a razão social, o CNPJ, o endereço e o telefone. O código do cliente não deve ser informado pelo usuário, deve ser gerado automaticamente a partir do número 1. Além disso, a sequência de códigos deve ser única para pessoas físicas e jurídicas, ou seja, não devem existir clientes PF e PJ com um mesmo código. Informe ao usuário se houve sucesso ou não na inserção.
   2) **Remover cliente** – Primeiramente, pergunte ao usuário se o cliente é pessoa física (PF) ou jurídica (PJ). Solicite então o código do cliente. Na remoção, deve ser atribuído o valor "null" à posição do vetor relacionada ao cliente informado. Informe ao usuário se houve sucesso ou não na remoção (por exemplo: código inexistente).
   3) **Consultar clientes** – A consulta deve apresentar ao usuário os dados de todos os clientes cadastrados, separados em pessoas físicas e pessoas jurídicas.

   A aplicação deve possuir as seguintes classes:

   - **Cliente –** Classe que contém os atributos privados int codigo, string endereco e string telefone, além dos métodos protegidos **get** e **set** necessários.
   - **PessoaFisica –** Classe filha da classe **Cliente**. Contém os atributos privados String nome e String cpf, além dos métodos públicos **get** e **set** necessários.
   - **PessoaJuridica –** Classe filha da classe **Cliente**. Contém os atributos privados String razaoSocial e String cnpj, além dos métodos públicos **get** e **set** necessários.
   - **Inicio –** Classe que contém o método **Main**, onde são instanciadas as classes **PessoaFisica** e **PessoaJuridica**. Na opção 1, deve-se chamar um método que usará os métodos **set** para criar e retornar um objeto contendo os dados da pessoa física ou da

pessoa jurídica, dependendo do tipo de cliente informado. Este objeto deve ser inserido então no vetor de pessoas físicas ou pessoas jurídicas. Por fim, o código deve ser incrementado para inserção do próximo cliente. Na opção 2, deve-se percorrer o vetor de objetos (pessoa física ou jurídica) em busca de um objeto que contenha o código de cliente igual ao código informado pelo usuário. Ao encontrá-lo, deve-se atribuir "null" ao objeto. Na opção 3, deve-se chamar um método que usará os métodos **get** para recuperar os dados de cada objeto armazenado nos vetores de pessoa física e pessoa jurídica e apresentá-los ao usuário.

2. Faça um programa em C# que leia os valores das vendas de uma loja nos últimos seis meses e acumule o valor total em um atributo estático por meio de um método dinâmico, o qual será chamado por um objeto v1 do tipo **Venda**. Ao final, apresente o valor total das vendas ao usuário por meio de outro método dinâmico, o qual será chamado por um objeto v2, também do tipo **Venda**. Transforme o atributo estático em dinâmico, execute novamente o programa e veja se o resultado foi alterado.

3. Faça um programa em C# que apresente ao usuário as opções a seguir, enquanto ele não digitar a opção 0 (zero). De acordo com o número da opção informada, o programa deverá efetuar a operação, solicitando as informações necessárias ao usuário.
   1) **Calcular raiz quadrada** – Deverá chamar o método sobrecarregado e estático ExecutarCalculo(double num) da classe **Calculo**.
   2) **Calcular potenciação** – Deverá chamar o método sobrecarregado e estático ExecutarCalculo(double base, double expoente) da classe **Calculo**.
   3) **Calcular fatorial** – Deverá chamar o método sobrecarregado e estático ExecutarCalculo(int num) da classe **Calculo**.

4. Faça um programa em C# que apresente ao usuário as opções a seguir, enquanto ele não digitar a opção 0 (zero). De acordo com o número da opção informada, o programa deverá efetuar a operação, solicitando uma cadeia de caracteres ao usuário.
   1) **Informar texto** – Deverá chamar o método SetTexto(string texto) da classe **Pesquisa**.
   2) **Buscar string** – Deverá chamar o método virtual BuscarString(string cadeiaCaracteres) da classe **Pesquisa**, que retornará se a cadeia de caracteres procurada existe ou não no texto.
   3) **Buscar string no início** – Deverá chamar o método sobrescrito BuscarString(string cadeiaCaracteres) da classe **PesquisaInicio**, filha da classe Pesquisa, que retornará se a cadeia de caracteres procurada existe ou não no início do texto. Configure este método para que não possa ser sobrescrito novamente em uma eventual classe filha da classe PesquisaInicio.
   4) **Buscar string no fim** – Deverá chamar o método sobrescrito BuscarString(string cadeiaCaracteres) da classe **PesquisaFim**, filha da classe Pesquisa, que retornará se a cadeia de caracteres procurada existe ou não no final do texto. Configure este método para que não possa ser sobrescrito novamente em uma eventual classe filha da classe PesquisaFim.