******* Exercício - POO *******

Classes e Objetos

1. Crie uma classe chamada Jogo com seguintes atributos:
   - Código - (int)
   - Nome - (string)
   - Categoria – (string)
   - Data de lançamento - (DateTime)

Instancie um objeto da classe Jogo, preencha os seus atributos e exiba em vídeo os dados.

Encapsulamento

2. Utilizando a classe jogo criada no exercício anterior, faça um programa que instancie até 10 jogos. Após efetuar os cadastros, exiba os dados em vídeo. Encapsule os atributos. Validações:

   Código - (int) (> 0)
   Nome - (string) (obrigatório)
   Categoria – (string) (válido apenas "ação" "luta" "tiro" e "Esportes")
   Data de lançamento - (DateTime) (deve ser inferior à data atual)

3. Crie uma classe para representar um funcionário. A classe deve ter os seguintes atributos:

Código (int) : deve ser > 0
Nome (string) : (obrigatório)
RG (string) : (obrigatório)
Salario (double) : deve ser >=0

Encapsule os atributos com métodos Get/Set.

Crie também os seguintes métodos:

GetInss(): deve calcular o valor do INSS (11% do salário) e devolver este valor.

CalcularSalario() : Este método deverá retornar o salário do funcionário subtraído do INSS.

Crie um formulário para preencher os campos do cadastro.
Crie um botão para exibir os valores gravados na classe em um Textbox multiline.

Exercício sobre propriedades

4. Altere o exercício anterior para que ele use propriedades como forma de encapsulamento.

5. Crie uma classe chamada Aluno, com os seguintes atributos:
   nome (string)

nota1 (double)
nota2 (double)

Crie propriedades para acessar os atributos da classe.

Crie uma propriedade chamada Media que será somente leitura (apenas get) e devolverá a média aritmética (nota1 + nota2) /2

Validações:

Nota válida deve estar entre 0 e 10

Nome válido deve possuir nome e sobrenome.

Crie um formulário para instanciar o aluno um botão para exibir a média.

6. Altere o exercício anterior para que ele permita salvar até 10 alunos.

Exercício de sobrecarga de métodos

7. Crie três métodos sobrecarredos para efetuar uma soma entre dois valores. Um dos métodos irá receber dois inteiros e devolver um inteiro. O outro método irá receber dois valores double e devolver um double. O outro método irá receber dois valores do tipo string, converter para double, somar e devolver um double.

8. Aplicação tipo console.
   Crie um método para exibir um texto no centro da tela (na horizontal). Crie um método sobrecarregado onde seja possível informar o texto e a cor (tipo de dado da cor é ConsoleColor).

Construtores

9.
Altere o EX 3 adicionando um construtor parametrizado. Altere o programa principal para que ele utilize este construtor.

10.

Crie uma classe chamada ContaCorrente
A classe deverá ter os seguintes atributos:

agencia int
numero da conta string
nometitular string
Saldo (apenas leitura) double (inicia com zero)

Deverá haver um construtor parametrizado para que seja possível passar todos os parâmetros já na criação do objeto. Deverá haver os métodos:

* void Saque (double valor) - subtrai do saldo. Deve gerar uma exceção caso o saldo não seja suficiente.

* void Deposito (double valor) - acrescenta o valor ao saldo.

LISTAS

11.

Altere o exercício 10 para que os dados sejam salvos em uma lista.

- Crie um botão para listar todos os registros, exibindo todas as propriedades.
- Crie um botão para pesquisar uma conta através do nome do titular. Ajuste para que as operações de saque e depósito aconteçam na conta pesquisada.
- Crie um botão para remover um elemento da lista.

12.

Altere a classe pilha estática, adicionando a ela a opção de instanciar uma pilha passando por parâmetro a capacidade do vetor de dados. Caso seja utilizado o construtor padrão, defina 10 para a capacidade. Você precisará transformara constante em uma variável. Se quiser evitar que ela seja alterada fora do construtor, utilize a palavra chave readonly em sua definição.
Veja mais detalhes sobre variáveis readonly em:
https://docs.microsoft.com/pt-br/dotnet/csharp/language-reference/keywords/readonly
https://www.arungudelli.com/tutorial/c-sharp/the-curious-case-of-readonly-field-in-c/

Uma pilha permite que as operações sejam efetuadas apenas no topo. Vamos criar uma novo método dentro da classe pilha que permitirá empilhar em qualquer posição da pilha.
Além do método Empilha(valor) já existente, iremos criar um novo, com a seguinte assinatura:

Empilha (valor, posicao)

- O método deverá permitir empilhar de 0 até a posição Tamanho().
- Uma exceção deverá ser lançada se a posição informada for inválida.

Exemplo 1:

Para a pilha abaixo, temos:
Topo: 2
Tamanho: 3
Capacidade: 5

4[ ]
3[ ]
2[c]
1[b]
0[a]

Neste caso, deveria ser possível empilhar nas posições 0 até 3.

Exemplo 2:

Empilha("L", 1)

O resultado deveria ser:

4[ ]
3[c]
2[b]
1[L]
0[a]

Seguindo o mesmo conceito do exercício anterior, altere a classe pilha adicionando o método sobrecarregado string Desempilha(posicao), para que o usuário possa informar a posição que deseja desempilhar. Lance uma exceção caso a posição seja inválida.

Exemplo

4[ ]
3[c] << Topo
2[b]
1[L]
0[a]

string x = Desempilha(2)

4[ ]
3[ ]
2[c] << Topo
1[L]
0[a]

Métodos, atributos e classes estáticas e enumeradores

13.

Crie um enumerador para representar o sexo (Masculino e Feminino) e outro para representar os estados civis (casado, solteiro, viúvo) de um funcionário.

Crie a classe funcionário com os seguintes atributos:

string nome
SexoEnum sexo
ECivilEnum estado civil
datetime data de nascimento

Crie uma classe chamada métodos e crie os seguintes métodos estáticos:

bool validanome (string nome) - deverá validar se o nome possui nome e sobrenome.

bool validaData (string data, out idade) - deverá ver se a data informada é valida. Se for válida, retornar a idade no parâmetro de saída.

Crie um construtor padrão e outro parametrizado onde será possível informar todos os atributos.

Crie um formulário e valide os dados utilizando estes métodos

14.

Crie uma Interface ICadastro que possua os métodos Salvar e Pesquisar. As assinaturas dos métodos são:
Void Salvar();
booleano Pesquisar(string valor);

Coloque também 2 Propriedades: int Codigo e string Nome

Crie 2 classes: Funcionario e Produto e implemente esta interface em ambos.

Atributos:
Funcionario: codigo: int, string nome
Produto: codigo: int, string nome, double preco.

O método salvar de ambas as classes deve salvar os dados em um arquivo texto (dados.txt), acumulando os registros, um por linha, de forma que fique assim no arquivo:
F|codigo|nome → *para nomes*
P|codigo|nome|preco → *para produtos*

Faça a interface gráfica para cadastrar produtos e funcionários.

A pesquisa se dará pelo nome e caso encontre os atributos do objeto devem ser preenchidos. Preencha também os campos na tela após efetuar a pesquisa ou limpe-os caso não encontre.

15.

Faça um programa que leia de um arquivo texto as categorias de um produto.
As categorias são: (categorias.txt)
1,brinquedos
2,alimentos
3,bebidas
4,eletrodomésticos
5,eletrônicos

Faça um cadastro de produtos com os seguintes atributos: código, descrição, categoria. A categoria deve estar em um combobox.
Guarde os produtos em uma lista.

Coloque um botão na tela para listar todos os produtos em um listbox ou textbox, listando o código, a descrição do produto e a descrição da categoria.

16.

Crie uma classe para representar jogos.
Deve haver os seguintes campos:

Codigo : int (> 0)
Descricao : string (obrigatório)
Dificuldade: Enumerador(easy, normal, hard) (obrigatório)
Valor : double >= 1,00
Fabricante : Obrigatório. Exiba uma lista (ou caixa combo) de fabricantes e permita que o usuário selecione 1 valor da lista.

Deve haver 2 métodos sobrecarregados para atualizar o preço dos jogos. Um deles deverá receber por parâmetro um inteiro que representa o percentual que será aplicado sobre o preço do jogo. O outro deverá receber por parâmetro um double que representa o valor em reais que deverá ser adicionado ao preço do jogo.

Crie a classe fabricante:
Fabricante (int codigo, string nome)
Crie uma lista com os seguintes fabricantes (pode ser fixo, no código)
{1 – Konami} { 2 – Capcom } { 3 - Nintendo } { 4 – EA} {5 – Acclaim} { 6 – IronHide } { 7 – SEGA }

No formulário para teste da classe, inclua os botões para salvar a Lista (que irá salvar todos os objetos da lista em um arquivo texto, usando para isso o salvar que há disponível na classe Jogo) e um botão para carregar, que irá preencher uma lista com os objetos carregados do arquivo.

Deve haver um botão para listar todos os jogos cadastrados (verificar na lista).
O mesmo botão deve listar apenas os jogos de uma determinada dificuldade caso seja selecionada a dificuldade em uma caixa combo que ficará ao lado do botão. Para estes 2 utilize sobrecarga de métodos (métodos com o mesmo nome porém com parâmetros diferentes).

Adicione também dois botões para realizar os aumentos de preço nos jogos.

17.

Crie uma classe básica funcionário com os seguintes atributos:

- Código int
- Nome string
- Salário: Double
- Double CalculoSalario() método que calcula o salário. Neste método, apenas retorne o salário.
- Sobrescreva o método ToString() para devolver os atributos básicos, além do salário calculado.

Crie uma classe chamada FuncionarioPeao que herda de funcionário e terá, adicionalmente:

- Double HoraExtraEmReais. Este campo deve entrar no cálculo do salário.

Crie uma classe chamada FuncionarioGerente que herda de funcionário e terá, adicionalmente:

- Double BonusEmReais. Este campo deve entrar no cálculo do salário.
- Int QtdeFuncionariosSubordinados

Crie uma classe chamada FuncionarioVendedor que herda de funcionário e terá, adicionalmente:

- Double MetaDeVendaMesEmReais
- Double VendasdoMesEmReais
- Double PorcentagemSobreVendas (para incorporar no salário). Exemplo: 10%, deve-se calcular 10% sobre o campo VendasdoMesEmReais e incorporar ao salário durante o cálculo. Este valor só será aplicado se a meta for atingida.
-

Crie um formulário para instanciar as 3 classes derivadas, e guarde em uma lista. Crie um botão para exibir as propriedades e o salário final de cada funcionário instanciado.

18.

Exercício: Banco

Abstract Conta Bancaria : Object
Atributos:

- String Nome do cliente
- String Número da conta (formato 9999-9)
- Double Saldo Métodos:
    - void Deposito(double valor);
    - void Saque(double valor);

ContaCorrente: ContaBancaria

Atributos:

- Double Limite de crédito
- Bool UtilizaTalaoCheque

Métodos:

- void Saque(double valor); - O saldo + limite de crédito não pode ser < 0

ContaPoupanca: ContaBancaria
Atributos:

- Int Dia de aniversário:
- string Nº da c.Corrente atrelada

Métodos:

- void Saque(double valor); - O saldo não pode ser < 0

Faça uma tela para cadastrar conta corrente e poupança. Salve em uma lista.
Crie um botão para depositar e 1 botão para sacar. Para tanto, o usuário deverá informar o nº da conta e o sistema irá efetuar a operação na conta selecionada.
Sobrescreva o método toString() para listar todas as informações da conta.
Faça uma opção para listar todas as contas cadastradas (cc. ou poupança) informando inclusive o tipo da conta (Ex: Conta corrente / Poupança)

19.

********** herança **********

Baseado no exemplo do aluno, faça:
* coloque mais 3 botões para listagem:
  - apenas dos alunos do EM.
  - apenas dos alunos da FTT.
  - apenas dos alunos da Pós.

* Crie um atributo na classe Aluno do tipo int
  para guardar a quantidade de faltas do aluno

* Crie um método na classe Aluno para calcular a frequência do aluno. O método deverá devolver a porcentagem de frequência do aluno com base na seguinte formula: 80 aulas = 100%. Subtraia as faltas para saber qual foi a frequência do aluno.

* Crie um método situação que irá devolver um enumerador que indica se está aprovado ou reprovado. Ele deve levar em consideração a média e também a frequência do aluno(>= 75) para retornar se ele está aprovado ou não. Ou seja, para estar aprovado é necessário tirar a nota mínima do curso e também ter frequência maior ou igual a 75.

* Considerar para o cálculo as seguintes médias de aprovação:
    Ensino médio: 6,0
    Faculdade : 5,0
    Pós : 7,0

20.

Crie as classes em C# (modo console) que melhor representem as necessidades abaixo. Será analisada a capacidade do aluno de abstrair e utilizar da melhor forma os conceitos de orientação a objetos.

O modelo abaixo tem por objetivo representar automóveis, que podem ser carros, caminhões, ônibus e até motos.

Independentemente do tipo de veículo automotor, todos eles devem possuir os seguintes atributos e métodos:

Atributos:

- string descricao
- double capacidadeMaximaEmKg
- double velocidadeAtualEmKM (atributo apenas leitura)
- double capacidadeCarregadaemKg (atributo apenas leitura)

Métodos:

- void Carregar (double peso) -> gerar exceção se exceder a capacidadeMaximaEmKg. Exibir em vídeo a capacidade informada após carregar.
- void Acelerar() ; // aumenta em 1 km a velocidade atual do veículo e exibe no console a nova velocidade
- double PagarPedagio(); // seu cálculo depende de características do tipo de veículo. Todo veículo deve pagar pedágio.
- o método ToString() deverá exibir todos os atributos da classe, concatenados.

Obs: um veículo automotor não deve ser instanciado. Apenas os seus descendentes pode ser instanciados.

Segue abaixo uma descrição dos diversos tipos de automóveis

Um carro é um tipo de veículo automotor que possui, além dos atributos e métodos de um veículo automotor, adicionalmente:

Atributos:

- int quantidade de portas.
- bool UtilizandoReboque ;

Métodos:

- O método ToString deve refletir todos os atributos da classe.

- No método de cálculo do pedágio: caso esteja usando reboque cobrar R$ 10,00. Caso contrário, cobrar R$7,00. Exibir o valor pago em vídeo.

Uma moto é um tipo de veículo automotor que possui, além dos atributos e métodos de um veículo automotor, adicionalmente:

Atributos:
- int cilindradas

Métodos
- void Empinar() -> exibe no console "Empinando...";
- O método ToString deve refletir todos os atributos da classe.
- Cálculo do pedágio: Para moto, considerar o valor R$ 2,00. Exibir o valor pago em vídeo

Um caminhão é um tipo de veículo automotor que possui, além dos atributos e métodos de um veículo automotor, adicionalmente:

Atributos:
- int QuantidadeEixos;

Métodos
- void Carregar(double peso) -> exibe no console "Carregado". Caso o peso ultrapasse o peso máximo, gere uma exceção com o texto "Sobrecarregado!";
- O método Acelerar não deve acelerar o caminhão caso o peso carregado no caminhão ultrapasse o peso máximo. Neste caso, gere uma exceção com o texto "Sobrecarregado!".
- O método ToString deve refletir todos os atributos da classe.
- Crie o método Descarregar() , que irá retirar todo o peso e escrever "Vazio" no console;
- Cálculo do pedágio: Considerar o valor R$ 5,00 por eixo. Exibir o valor pago em vídeo.

Crie as classes em C# (modo Windows forms) que melhor representem as necessidades abaixo. Será analisada a capacidade do aluno de abstrair e utilizar da melhor forma os conceitos de orientação a objetos.

O modelo abaixo tem por objetivo representar automóveis, que podem ser carros, caminhões, ônibus e até motos.

Independentemente do tipo de veículo automotor, todos eles devem possuir os seguintes atributos e métodos:

Atributos:
string descricao
double capacidadeCargaMaximaEmKg
double velocidadeAtualEmKM (atributo apenas leitura)
double capacidadeCarregadaEmKg (atributo apenas leitura)

Métodos:
double Carregar (double peso) -> gerar exceção (personalizada) se exceder a capacidadeMaximaEmKg//. Exibir em vídeo a capacidade informada após carregar.
double Acelerar() ; // aumenta em 1 km a velocidade atual do veículo e exibe no console a nova velocidade

double PagarPedagio(); // seu cálculo depende de características do tipo de veículo. Todo veículo deve pagar pedágio.
o método ToString()// deverá exibir todos os atributos da classe, concatenados.

Obs: um veículo automotor não deve ser instanciado. Apenas os seus descendentes podem ser instanciados.

Segue abaixo uma descrição dos diversos tipos de automóveis

Um carro é um tipo de veículo automotor que possui, além dos atributos e métodos de um veículo automotor, adicionalmente:

Atributos:
int quantidade de portas.
bool UtilizandoReboque ;
Métodos:
O método ToString deve refletir todos os atributos da classe.
No método de cálculo do pedágio: caso esteja usando reboque cobrar R$ 10,00. Caso contrário, cobrar R$7,00. Exibir o valor pago em vídeo.

Uma moto é um tipo de veículo automotor que possui, além dos atributos e métodos de um veículo automotor, adicionalmente:

Atributos:
int cilindradas
Métodos
string Empinar() -> exibe no console "Empinando...";
O método ToString deve refletir todos os atributos da classe.
Cálculo do pedágio: Para moto, considerar o valor R$ 2,00. Exibir o valor pago em vídeo

Um caminhão é um tipo de veículo automotor que possui, além dos atributos e métodos de um veículo automotor, adicionalmente:

Atributos:
int QuantidadeEixos;

Métodos
double Carregar(double peso) -> exibe no console "Carregado". Caso o peso ultrapasse o peso máximo, NÃO gere uma exceção, mas exiba em vídeo a mensagem “Sobrecarregado”;
O método Acelerar não deve acelerar o caminhão caso o peso carregado no caminhão ultrapasse o peso máximo. Neste caso, gere uma exceção com o texto “Sobrecarregado!”.
O método ToString deve refletir todos os atributos da classe.

Crie o método Descarregar() , que irá retirar todo o peso e escrever "Vazio" no console;
Cálculo do pedágio: Considerar o valor R$ 5,00 por eixo. Exibir o valor pago em vídeo.

Crie também botões para executar os métodos específicos de cada classe, como o descarregar do caminhão.
Esses métodos deverão atuar sobre 1 objeto na lista. Para isso, você precisará primeiro efetuar uma pesquisa pelo nome do veículo, achá-lo na lista e então executar o método escolhido.

Após terminado o sistema, foi solicitado que fossem incluídas algumas funcionalidades no carro e no caminhão:
int VelocidadeLimpador
void AcionarLimpador();
void AbrirPorta();

Implemente estas funcionalidades nos veículos onde elas se aplicam e crie os botões para testá-las.

21.

Crie uma lista que seja capaz de armazenar qualquer tipo de objeto.
utilize-a depois para armazenar alunos (utilize o exemplo dos alunos, substituindo a lista do C# pela sua lista)

22.

Exercício de recuperação (POO e EDA)

Crie uma lista para armazenar times de futebol.
Cada elemento da lista deverá armazenar:
O nome do time.
O nome do treinador.
Uma lista com os 11 jogadores.
Cadas jogador deverá possuir:
O nome
O nº da camisa

Utilize controle de exceção quando necessário.
Crie classes quando necessário.
Faça em windows forms, com a opção de listar todos os times e seus respectivos jogadores.

Exercício de recuperação (POO e EDA)

Crie uma lista para armazenar funcionários
Cada elemento da lista deverá armazenar:
O nome funcionario
O salário
Uma lista com os dependentes
Cadas dependente deverá possuir:
O nome
A idade

Utilize controle de exceção quando necessário.
Crie classes quando necessário.
Faça em windows forms, com a opção de listar todos os funcionários e seus dependentes.