A Ciência da Computação é a única que é exclusiva para ajudar outras áreas;

IDE: Integrated Development Environment;

ASP: Active Server Pages;

-- RAZOR / BLAZOR --

ORM: Object Relation Mapping;

Aplicação WEB:

- Opera em uma arquitetura Client / Server, sendo o cliente a máquina que está entrando na aplicação e o Server vai ser uma máquina que vai estar rodando a aplicação;

DNS: Domain Name System;

HTTP / HTTPS: Hyper Text Transfer Protocol (Secure);

HTML: HyperText Markup Language;

Hyper text pode ser: imagem, texto e etc que está vinculado a um tipo de recurso;

-----------------------VISUAL STUDIO-----------------------

Ctrl + Alt + L : Abrir gerenciador de soluções

Shift + Alt + Enter: Tela cheia

-----------------------------------------------------------

------------------------POWER SHELL------------------------

git remote -v: Ver os repositórios remotos

git remote add origin link: Puxar o repositório remoto para local

-----------------------------------------------------------

MVC: Model View Controller - Arquitetura (Foi o tipo de projeto que criamos com o .NET);

-> Model: Especifica os dados que compõem o modelo do APP;

-> View: A UI (User Interface);

-> Controller: Responde aos comandos da aplicação;

var: variável do tipo implícito, onde não determina o tipo da variável. Ele pode se tornar tipo inteiro, string, uma função, um objeto ou algo mais complexo.

Lembrar de colocar o @model no index cshtml para mostrar o tipo dos dados que vão ser recebido na web

Atividade 1: Com base no exemplo feito em laboratório, com os tipos inteiros nas variáveis totalCount e userCount que foram exibidos em tela no Home/Index, contidos na classe Variáveis faça:

Na página 47 do livro é possível acessar os valores mínimos e máximos para os demais tipos númericos. Edite a classe Variáveis, crie novos atributos a ela e armazena os valores Min e Max de cada tipo numérico para em seguida exibir em tela no Home/Index

------------------------Aplicações WEB------------------------

Protocolo: Forma de comunicação de dados;

Todas as aplicações WEB utilizam o protocolo HTTP e algumas o HTTPS (HTTPS é uma versão segura do HTTP, e ele utiliza o HTTP como base). HTTP é stateless e segue o padrão "requisição-resposta". Foram desenvolvidos pelo IETF e o W3C. A versão atual é HTTP/2 e foi padronizada em 2015, sendo suportado pela maioria dos navegadores. Sendo binário, totalmente multiplexado, cabeçalho comprimido para reduzir pacotes.

Web app: É um sistema dinâmico, possui interatividade com o usuário, utiliza banco de dados.

Website: Página estática, exibe imagem, texto, vídeo. Não tem interação com o banco de dados, não possui dinamicidade.

Encriptação Simétrica: Método convencional para garantir integridade dos dados transferidos. Faz uso de uma única chave secreta "Symmetric Key" para criptografar e descriptografar a mensagem, onde apenas o remetente e o destinatário possuem esta chave.

Encriptação Assimétrica: Utiliza duas chaves: Pública e privada. A pública codifica e a privada decodifica.

Request-Response: Baseado em Cliente-Server. Servidor é um computador que recebe requisições de um cliente e os serve. Tipicamente um computador parrudo com memória e processdor suficiente para atender as demandas dos clientes (usuários).

Stateless: Não armazena aos estados servidos e recebidos anteriormente, não possui um histórico.

Cliente pode ser um navegador, servidor pode ser seu computador.

Um bom exemplo é o front end (Cliente) e back end (Servidor).

Verbos HTTP:

- GET: Buscar recursos, acessar cache;

- POST: Criar um novo recurso;

- PUT/PATCH: Atualizar um recurso existente;

- DELETE: Remover um recurso.

Atividade 2: Pegar o algoritmo de Cesar e criptografar a mensagem.

Aspas simples: Para caracteres

Aspas duplas: String

Alt + Shift + . = Seleciona as mesmas palavras uma por uma pra baixo

Não pode usar interpolação nas declarações de variáveis, então precisa utilizar no método construtor

Atividade 3: Crie um novo projeto ASP.NET Core Web MVC para fazer a leitura de uma entrada de dados numérica e ser capaz de devolver à tela do usuário o número informado escrito por extenso. Lembre-se de verificar se trata-se de uma unidade, dezena, centa e milhar.

View > Controller > Model (Sendo model o centro da arquitetura MVC)

GUI: Guide User Interface

UI: User Interface

UX: User Experience

Debuggar:

Escolher ponto de parada;

Clicar F10 para ele ir executando próxima linha;

Clicar F5 para ir até o fim do código.

FLAG, CONTADOR E ACUMULADOR (ESTRUTURA DE REPETIÇÃO)

FLAG - EXPRESSÃO BOOLEANA

CONTADOR - CADA VEZ QUE O LAÇO ACONTECER, VAI CONTAR

ACUMULADOR - FAZ UM PROCESSO DE ACUMULAÇÃO

LAÇOS DE REPETIÇÃO = WHILE, DO WHILE, FOR, FOREACH

RECURSIVIDADE = Chamar a si mesmo dentro do próprio escopo

CASO BASE

CASO RECURSIVO

Para colocar um botão de navegação a mais na tela do home o caminho é este: Views > Shared > Layout e nesta folha de código eu alterei a "ul" colocando um novo li copiando de cima e mudando a action.

Comentário HTML: <!-- -->

Comentário Razor: @\* \*@

Atividade:

1) Escreva um programa recursivo em C# para imprimir os números de n até 1 (decrescente);

2) Escreva um programa em C# capaz de sumarizar os números de 1 a n. Por exemplo: n = 10 [1+2+3+4+5+6+7+8+9+10];

3) Escreva um programa recursivo em C# capaz de contar quantos caracteres tem em uma string;

4) Escreva um programa recursivo que seja capaz de identificar se uma palavra é ou não um PALÍNDROMO: OVO, ARARA, MUSSUM, RADAR, ETC.

Conteúdos da prova: if, for, switch, vetor, matriz, visual studio, dotnet, c#, http, post, get, put, delete, conceitos.

Aula Teórica

OO -> Orientada a Objetos

POO -> Programação Orientada a Objetos = OOP -> Object Orientation Program

POG -> Programação Orientada a Gambiarra

API -> Serviço distribuído, serve para conectar um sistema a outro, tipo um Gateway

Acess Program Interface.

Paradigma é tipo um procípio, quando falamos "Quebrar paradigmas", é mudar a forma de fazer algo.

Classe -> Você define a classe, você escreve a classe, a palavra-chave é "class"

Formada por propriedades, funções/métodos que compõem os membros da classe, quando você escreve uma função, um procedimento, por exemplo, você está definindo os membros

da classe.

public class Customer

{

public int CustomerId { get; set; }

public string EmailAddress { get; set; }

public string FirstName { get; set; }

public string LastName { get; set; }

public bool validate() { ... }

}

Quando você vai escolher um personagem em um jogo, você pode escolher a classe, os atributos deles mudam. Por exemplo, um arqueiro tem um atributo (ele tem mais mira),

um bárbaro tem outros (ele é mais forte), um clérigo tem outras (tem mais religião).

Objeto -> é uma instância dentro da classe, ocupa um espaço na memória do computador, é uma variável que você criou, está pronta para ser usada a qualquer momento. Variável que possibilita o acesso aos métodos e propriedades especificadas pela classe. O personagem que você cria no jogo, que você vê, que você mexe, é a representação gráfica, ele é o objeto (Que pode receber os atributos da classe que você definir para ele). Se você cria uma variável, o "=" é um "igual" no sentido de atribuição.

**Se tem parênteses, é um método.**

var customer = new Customer(); //Customer é consumidor em inglês;

customer.firstname = "Frodo";

customer.Validate();

Métodos -> Podemos definir como se fossem verbos (correr, andar, pular, nadar...);

Os objetos possuem um formato especificado pela classe, porém, cada um deles podem armazenar valores diferentes em suas propriedades ocasionando comportamentos diferentes em seus métodos.

**Classe:** É a forma de fazer bolacha

**Objeto:** É a bolacha que foi feita com o molde, o resultado, tendo objetos diferentes, uma é salgada, outra é doce, outra é rosa e etc...

**Business Object:** São as classes projetadas para solucionar problemas de um determinado domínio de aplicação de software.

Certamente teríamos uma definição de classe Consumidor em um sistema comercial, contendo nome e email como propriedades por exemplo.

**Entidade:** Qualquer coisa importante do domínio da aplicação ou do mundo real que precise ser representado como uma classe;

Vou olhar o domínio, ou seja, o contexto, separar as entidades, definir as classes e propriedades para produzir os objetos.

**POO:** Uma abordagem para projetar e construir aplicações que são:

- Flexíveis;

- Naturais;

- Bem elaboradas;

- Testáveis;

Sendo focada em objetos que interagir uns com os outros;

**Passos da POO:**

- 1: Identificar as classes dos requisitos ou especificações;

- 2: Analise as classes e separe as responsabilidades. Tenha em mente a aplicação poderá ser decomposta com o mínimo de impacto;

- 3: Estabeleça os relacionamentos entre as classes para que os objetos possam operar em conjunto na resolução da proposta do software;

- 4: Reutilize. Construa seu código pensando em reaproveitamento do código para evitar repetições e facilitar manutenções;

**Atividade:**

Você recebeu uma solicitação por email de uma amiga veterinária, informando que está com problemas para gerenciar os atendimentos dos pets. Na mensagem, ela diz que gostaria de ter registros dos animais, que precisam também dos dados dos seus parceiros veterinários que atendem no centro clínico para poder saber quais médicos atenderam quais animais em um determinado período de tempo, ou até mesmo, registrar quais procedimentos foram realizados com os animais nos últimos atendimento.

Classes: ANIMAIS, PARCEIROS VETERINARIOS, CENTRO CLÍNICO, ATENDIMENTO

Atendimento (DATA/HORA, PARCEIRO\_VETERINARIO, ANIMAL, PROCEDIMENTO, SITUACAO, PARECER\_MEDICO, PROCEDIMENTO);

Animais (NOME, RAÇA, ALERGIA, DONO, IDENTIFICACAO\_DIGITAL, COR, PELAGEM, SEXO, TAMANHO, PESO, IDADE);

Centro Clínico (NOME, ENDEREÇO, HORA\_ATENDIMENTO, CONTATO);

Parceiros Veterinários (NOME, ESPECIALIZAÇÃO, IDADE, CPF, TELEFONE, EMAIL, CRM).

**Abstração:** É um processo que dependerá dos requisitos da aplicação. Em nosso caso, temos a ENTIDADE Veterinário, onde se faz necessário registrar poucos dados (Nome, Email, Telefone, CRM) devido ao escopo reduzido do problema, mas se estivéssemos modelando um sistema mais robusto para aplicações diversas, certamente precisaríamos de mais informações sobre o veterinário, como endereço, especialidade e etc...

**Encapsulamento:** Técnica para "esconder" os dados e os detalhes da implementação em uma classe.

4 pilares do POO:

- Abstração;

- Encapsulamento;

- Herança;

- Polimorfismo.

**Classes Entidades**

Persistir dados: Manter os dados no armazenamento.

Exemplo:

|  |
| --- |
| **Customer** |
| Name |
| Email adress |
| Work adress |
| Validate() |
| Retrieve() |
| Save() |

Organizando a estrutura da Aplicação em CAMADAS:

- Camada de Interface do Usuário

- Camada de Lógica de Negócios

- Camada de Acesso a Dados

- Camada de Código Comum

Vantagens em organizar em camadas:

- Dividir a aplicação em camadas, tornando o trabalho mais simples e organizado. Exemplo: Saber onde mexer quando der algum problema ou precisar fazer alguma melhoria.

- Separar a camada de negócios da interface do usuário, permite, mais tarde, adicionar com mais tranquilidade uma interface web a nossa aplicação, por exemplo.

Objetos são tipos referenciados: Objetos guardam um ponteiro de memória para os dados e não os dados em si mesmos, diferente dos tipos de dados por valor, que armazenam os dados diretamente.

Objetos são apenas “variáveis” que tem um tipo Abstrato e as variáveis que tem o tipo primitivo não são objetos.

Passagem por valor: Quando ocorre uma cópia do conteúdo em uma variável. Exemplo:

int i1 = 50

int i2 = i1

T.A.D: Tipo Abstrato de Dados (Uma representação abstrata de algo do mundo real. Exemplo: Vendedores, Produto, Nota...)

Var c1 = new Customer();

C1.FirstName = “Bilbo”;

Var c2 = c1;

C2.FirstName = “Frodo”;

No código acima (Todo objeto é um ponteiro) c2 está sendo um ponteiro para c1, caso você mude o valor de c2 ele muda o valor de c1.

**Modificador STATIC**

O modificador static declara um membro que pertence à classe em si mesma e não à instância do objeto.

Útil para rastrear informações compartilhadas entre todas as instâncias de objeto daquele tipo.

**Exemplo:**

public IActionResult Index()

{

Customer c1 = new Customer();

c1.Name = "Frodo";

c1.ObjectCount++;

Customer.InstanceCount++; // Aqui soma 1

var c2 = new Customer();

c2.Name = "Galadriel";

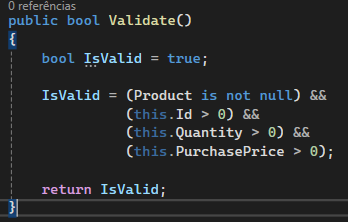
c2.ObjectCount++;

Customer.InstanceCount++; // Aqui soma mais 1, ficando com 2

return View();

}

Digamos que o modificador static é uma propriedade da classe (universal para todos os objetos) e não apenas para um objeto em específico.



This = Serve para quando tem dois nomes iguais, e colocando o this você sinaliza que você quer utilizar o atributo ou o nome que está presenta no escopo da classe, caso não seja colocado this ele vai utilizar o nome do escopo local. Pode ser lido como o nome da classe.z

**Assinatura do Método:**

- Composto pelo nome e conjunto de parâmetros e seus tipos;

- Não inclui o tipo de retorno;

- Utilizado pelo sistema para orientar as chamadas de funções;

- Cada assinatura de um método deve ser única;

- Se trata do nome e dos parâmetros do método, digamos que é o nome dele.



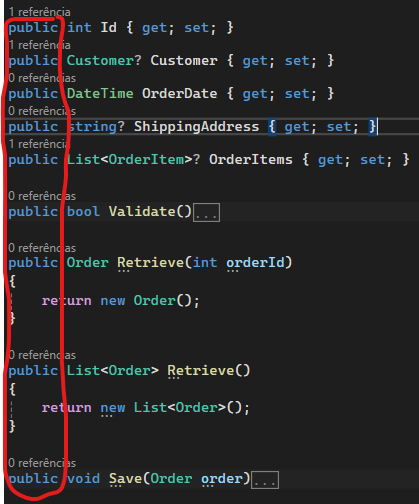
(Apenas nome do método e parâmetro)

**Sobrecarga/Overloading:**

- Usado para descrever método que possuem o mesmo nome com diferentes parâmetros.

**Contract:**

- As especificações dos modificadores de acesso definem o contrato da classe. Quais atributos e métodos estão visíveis, para quem e em que nível de acesso.



**Construtores:**

- É um tipo especial de método com o mesmo nome da classe;

- Geralmente definidos no primeiro parágrafo da classe;

- É o método executado quando um objeto do tipo da classe é criado;

- Um método construtor sem parâmetros é definido como construtor padrão (default constructor);

#region Nome

#endregion

O código acima serve para agrupar essas linhas caso que queira.

**Separando Responsabilidades**

- Mantenha suas classes focadas mantendo as responsabilidades bem definidas;

- Imagine uma empresa em que o Presidente é responsável por tudo. Isso não é uma boa ideia, conforme o negócio cresce ele precisará delegar responsabilidades para outras pessoas;

**- Minimizar o acoplamento:** Quando menos dependências uma classe tiver, mais fácil é de escrever rotinas de teste, manter e atualizar com o tempo. Acoplamento é o grau de dependência de uma classe para com outras com classes.

**- Maximizar a coesão:** É uma medida de quão relacionado tudo na classe é ao propósito da classe. Por exemplo, o endereço da entrega. Ele possui muitos atributos e informações que não são especificamente do pedido, nem do consumidor, mas do endereço em sua essência.

**Bordões:**

YAGNI: You aren´t Gonna Need it (Não adicione código que não é necessário)

KISS: Keep it simple, stupid! (Mantenha as coisas simples, se está complicado provavelmente está errado)

DRY: Don´t Repeat Yourself (Escreva classes e métodos de modo que eles possam ser reaproveitados)

**O Padrão Repositório**

O famoso pattern repository