SENAI

Serviço Nacional de Aprendizagem Industrial

PELO FUTURO DO TRABALHO

Exercícios

- Corrida de cachorros
- Exercícios da aula
- https://github.com/drhamann/poo-senai
- https://github.com/drhamann/senai-avancado

Programação Orientada a Objetos

Corrida de cachorros

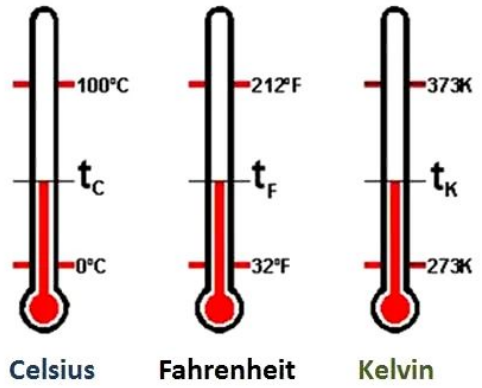
https://github.com/drhamann/poo-senai/tree/main/Programacao.Orientada.Objetos/CorridaDeCachorros

Lista de revisão

https://github.com/drhamann/poo-senai/blob/main/Programacao.Orientada.Objetos/revisao-logica/MinApiWithSwagger/ExercisesController.cs

Programação Orientada a Objetos

- Vamos ver como está o conhecimento
- Crie um projeto para resolver a sequência de fibonacci 1,1,2,3,5,8,13,21 …. deve pedir o valor a ser calculado
- ex: 5
- resposta: 1,1,2,3,5
- Converter temperatura, deve solicitar a temperatura em celsius e dar a conversão em kelvin e fahrenheit
- ex: 10
- respota : 10 graus celsios são 82 °F e 283.15 Kelvin
- Converter para Fahrenheit (°F): Fórmula: °F = (°C × 9/5) + 32
- Converter para Kelvin (K): Fórmula: K = °C + 273.15



Programação Orientada a Objetos

- Calculo de frete para uma compra
- Vamos criar um programa que receba a distância entre o distribuidor e o local de entrega e calcule o valor do frete, sendo que para km é cobrado R$ 0,10
- A cada 200 km, deve adicionar uma taxa de R$ 1,00
- A cada 1000 km, deve adicionar uma taxa de R$ 10,00
- Agora precisamos calcular o frete baseado no preço da encomenda
  - O valor do frete não altera para produtos com até 5kg
  - Multiplicar o valor do frete para produtos entre 5kg até 25kg em 5x
  - Multiplicar o valor do frete para produtos entre 25kg até 75kg em 3x

Programação Orientada a Objetos

- Exercício 2, vamos criar um cenário relacionado a dispositivos eletrônicos.
- Exercício: Dispositivos Eletrônicos
- Crie uma hierarquia de classes/interfaces para representar diferentes tipos de dispositivos eletrônicos (Celular, cafeteira, geladeira …). Cada dispositivo deve ter métodos para ligar, desligar e exibir informações sobre seu status.
- IDispositivoEletronico (Interface):
    - Métodos:
    - Ligar(): Liga o dispositivo.
    - Desligar(): Desliga o dispositivo.
    - ObterStatus(): Retorna uma mensagem indicando o status do dispositivo (ligado/desligado).
-

Programação Orientada a Objetos

- Exercícios
- Vamos criar um software em que inicialmente não seguimos o princípio Open/Closed (OCP). Temos alguns carros e devemos calcular o preço de aluguel de acordo com o tipo de carro.

Programação Orientada a Objetos

```
public class Carro
{
    public string Modelo { get; set; }
    public int Ano { get; set; }
}
public class CalculadoraAluguel
{
    public double CalcularPrecoAluguel(Carro carro)
    {
        double preco = 0;
        if (carro.Ano >= 2020)
        {
            preco = 100;
        }
        else
        {
            preco = 50;
        }
        return preco;
    }
}
```

```
public class CarroElétrico : Carro{    public int AutonomiaBateria { get; set; }}
public class CalculadoraAluguel{
    public double CalcularPrecoAluguel(Carro carro)    {
        double preco = 0;
        if (carro.Ano >= 2020)
        {           preco = 100;       }
        else if (carro is CarroElétrico)       {
            // Violando o OCP - Adicionando lógica específica para CarroElétrico
            var carroEletrico = (CarroElétrico)carro;
            if (carroEletrico.AutonomiaBateria > 300)           {
                preco += 50;           }       }
        else
        {
            preco = 50;
        }

        return preco;
    }
}
```

Programação Orientada a Objetos

- Exercícios
- Vamos criar um software em que inicialmente não seguimos o princípio LISKOV (LSP). Temos um guarda chuva comum e outro automático, vamos implementar como eles devem abrir.

Programação Orientada a Objetos

```
public class GuardaChuva
{
    public virtual void Abrir()
    {
        Console.WriteLine("Guarda-chuva aberto.");
    }

    public virtual void Fechar()
    {
        Console.WriteLine("Guarda-chuva fechado.");
    }
}
```

```
public class GuardaChuvaAutomatico : GuardaChuva{
    private bool _estaAberto;
    public override void Abrir()    {
        Console.WriteLine("Guarda-chuva automático abrindo.");
        _estaAberto = true;
    }
    public override void Fechar()    {
        Console.WriteLine("Guarda-chuva automático fechando.");
        _estaAberto = false;
    }
    public void Agitar()    {
        if (_estaAberto)
        {
            Console.WriteLine("Guarda-chuva automático agitado.");
        }
        else       {
            Console.WriteLine("Não é possível agitar um guarda-chuva fechado.");
        }
    }
}
```

Programação Orientada a Objetos

- Agora queremos mostrar os produtos que tem na loja, para isso implemente

```
public void MostrarProdutos()
 {
    using (var connection = new SQLiteConnection(ConnectionString))
    {
       connection.Open();
       using (var command = connection.CreateCommand())
       {
          command.CommandText = "SELECT * FROM Products";
          using (var reader = command.ExecuteReader())
          {
             while (reader.Read())
             {
                Console.WriteLine($"Product: {reader["Name"]}, Price: {reader["Price"]}, Discount: {reader["DiscountPercentage"]}");
             }
          }
       }
    }
 }
```

Programação Orientada a Objetos

- Baseado no código anterior, altere crie um novo método, para Trazer todos os Produtos
- public Produtos[] TragaTodosOsProdutos(), e use esta implementação no MostrarProdutos()

Programação Orientada a Objetos

- Exercício, dado o código, aplique a correção para adequar ao DIP

```
public class MemoriaRepositorio
{
    public void SalvarEmMemoria(string dados)
    {
        // Lógica para salvar em memória
    }
}

public class BancoDadosRepositorio
{
    public void SalvarNoBancoDeDados(string dados)
    {
        // Lógica para salvar no banco de dados SQL
    }
}
```

```
public class PersistenciaService{
    private readonly MemoriaRepositorio memoriaRepositorio;
    private readonly BancoDadosRepositorio bancoDadosRepositorio;
    public PersistenciaService(MemoriaRepositorio memoriaRepositorio,
BancoDadosRepositorio bancoDadosRepositorio)    {
        this.memoriaRepositorio = memoriaRepositorio;
        this.bancoDadosRepositorio = bancoDadosRepositorio;
    }
    public void SalvarDados(string dados, bool usarBancoDeDados)    {
        if (usarBancoDeDados)       {
            bancoDadosRepositorio.SalvarNoBancoDeDados(dados);
        } else
        {
            memoriaRepositorio.SalvarEmMemoria(dados);
        }
    }
}
```

Programação Orientada a Objetos

- Exercício, agora implemente a ação de salvar um objeto do tipo Pessoa em memória e banco de dados

```
public class Pessoa{
    public string Nome;
    public string Idade;
}
public class SqliteRepositorio {
   private const string ConnectionString = "Data
Source=MeuBancoDeDados.db;Version=3;";
public SqliteRepositorio()    {
       CriarBancoDeDados();
   }
   public void SalvarDados(string dados)    {
      using (SQLiteConnection connection = new
SQLiteConnection(ConnectionString))
      {
         connection.Open();
 using (SQLiteCommand command = new
SQLiteCommand(connection))          {
            command.CommandText = "INSERT INTO Dados (Texto)
VALUES (@dados)";
            command.Parameters.AddWithValue("@dados", dados);
            command.ExecuteNonQuery();
        }      }   }
   private void CriarBancoDeDados()    {
      using (SQLiteConnection connection = new
SQLiteConnection(ConnectionString))       {
         connection.Open();
         using (SQLiteCommand command = new
SQLiteCommand(connection))          {
            command.CommandText = "CREATE TABLE IF NOT EXISTS
Dados (Id INTEGER PRIMARY KEY AUTOINCREMENT, Texto TEXT)";
            command.ExecuteNonQuery();
         }
      }
   }
```

C# Avançado - Exercícios

Escreva um programa, para saber como as pessoas estão, com as seguintes regras:

1 Pergunte o nome de quem está digitando

2 Pergunte como ela está

3 Grave as respostas em arquivo dividindo em datas

4 O nome do arquivo deve ser no formato “como_esta_vc_DD_MM_YYYY_hh_mm_ss.txt”

Escreva outro programa que leia o arquivo feito no primeiro programa e tenha a seguintes regras:

1 Mostrar os nomes

2 Busque as respostas por data

3 Busque as respostas por nome

4 Gera uma versão do arquivo em json

C# Avançado - Exercícios

Exercício: Sistema de Gerenciamento de Música

Você está desenvolvendo um sistema de gerenciamento de música e precisa implementar a funcionalidade de salvar e carregar dados de músicas usando JSON. Para isso, você deve utilizar a biblioteca `System.Text.Json` do C#.

1. **Crie uma classe `Music`**:

   - A classe deve ter propriedades como `Title`, `Artist`, `Album`, `Genre`, `Year`, etc.

   - Inclua anotações de atributo para personalizar a serialização/desserialização.

2. **Crie uma classe `MusicLibrary`**:

   - A classe deve conter uma lista de músicas (`List<Music>`).

   - Implemente métodos para adicionar e listar músicas na biblioteca.

3. **Serialização**:

   - Crie um método que recebe uma lista de músicas e serializa para JSON.

   - Salve o JSON resultante em um arquivo chamado "musicLibrary.json".

4. **Desserialização**:

   - Crie um método que lê um arquivo JSON ("musicLibrary.json") e desserializa as músicas de volta para uma lista.

5. **Teste**:

   - Crie algumas instâncias de músicas e adicione-as à sua biblioteca.

   - Serialize a biblioteca para JSON e salve no arquivo.

   - Limpe a biblioteca e depois desserialize as músicas do arquivo.

   - Liste as músicas para verificar se a desserialização foi bem-sucedida.

Dicas:

- Use a biblioteca `System.Text.Json.JsonSerializer`.

- Utilize o `File.WriteAllText` e `File.ReadAllText` para salvar e ler arquivos.

Desafio Adicional:

- Implemente um menu simples no console para permitir que o usuário adicione, liste e salve músicas através do programa.

C# Avançado - Exercícios

Hora do Café

Um gerente de uma cafeteria está realizando uma promoção e deseja oferecer um desconto para as bebidas de café. O programa que você recebe recebe o valor do desconto como entrada e define um dicionário, onde os nomes das bebidas de café são definidos como chaves, e seus preços são definidos como valores.

Escreva um programa para aplicar o desconto em todos os preços e exibir uma nova lista de preços no formato mostrado abaixo.

Entrada de Exemplo:`

10

Saída de Exemplo:

Americano: 45

Latte: 63

Flat White: 54

Espresso: 54

Cappuccino: 72

Mocha: 81

```
class Program    {
        static void Main(string[] args)
        {
            int discount = Convert.ToInt32(Console.ReadLine());

            Dictionary<string, int> coffee = new Dictionary<string, int>();
            coffee.Add("Americano", 50);
            coffee.Add("Latte", 70);
            coffee.Add("Flat White", 60);
            coffee.Add("Espresso", 60);
            coffee.Add("Cappuccino", 80);
            coffee.Add("Mocha", 90);

            //Resposta aqui em baixo

        }
    }
```

C# Avançado - Exercícios

Escreva um programa, para gerenciar as atividades do dia e semana:

1 Pergunte o título da atividade

1.1 Não deve permitir repetir a mesma atividade

2 Pergunte a descrição dela

3 Grave as respostas em um único arquivo por dia

4 O nome do arquivo deve ser no formato “atividades_DD_MM_YYYY_.txt”

Escreva outro programa que leia o arquivo feito no primeiro programa e tenha a seguintes regras:

1 Mostrar os atividades

2 Busque as atividades por dia

3 Busque as atividades por mês

4 Gera uma versão do arquivo em json

C# Avançado - Exercícios

Vamos continuar com nossa pizzaria. Agora vamos aplicar novas funcionalidades com uso de data:

1. Adicionar cliente.

1.1 O cliente deve ter histórico de ações

2. Adicionar pizza.

2.1 Pizza deve ter dado de criação

3. Adicionar pedido.

3.1 O pedido deve ter horário de solicitação

3.2 O pedido deve ter horário de finalização da preparação

3.3 O pedido deve ter horário de saída para entrega

3.3 O pedido deve ter horário de finalização da entrega

4. Obter todos os clientes.

5. Obter todas as pizzas.

6. Obter todos os pedidos.

7. Gerar relatório em arquivo texto dos pedidos finalizados mostrando os dados de quem pediu, a pizza, horário que o pedido começou e horário de finalização da preparação, horário que saiu e finalizou a entrega.

7.1 Deve permitir gerar o relatório escolhendo a data inicial e final

7.2 Deve ter uma opção de escolher entre arquivo texto e json