Programação Orientada a Objetos
Aula 10

- Revisão
- SOLID
- Avaliação

Programação Orientada a Objetos

- SOLID “ISP” Princípio da segregação de interface:
- Esse princípio lida com as desvantagens das "interfaces gordas" (interfaces que contêm muitos métodos), indicando que as interfaces de uma classe podem ser divididas em grupos de métodos. Cada grupo atende a um conjunto diferente de clientes, onde alguns clientes usam um grupo de métodos e outros clientes usam outros grupos.
- O ISP reconhece que existem objetos que requerem interfaces não coesas, mas sugere que os clientes não devem conhecê-las como uma única classe. Em vez disso, os clientes devem conhecer classes abstratas que possuem interfaces coesas.
- Resumidamente, o ISP promove a criação de interfaces menores e focadas, utilizando classes abstratas para proporcionar uma visão coesa para os clientes. Isso visa melhorar a manutenibilidade e a flexibilidade do código, evitando as armadilhas associadas às "interfaces gordas".

Programação Orientada a Objetos

```
public interface IMaquina
{
    void Imprimir();
    void Digitalizar();
    void EnviarFax();
}
```

```
public interface IImprimivel
{
    void Imprimir();
}

public interface IDigitalizavel
{
    void Digitalizar();
}

public interface IFaxavel
{
    void EnviarFax();
}
```

Programação Orientada a Objetos

- Exercicio, dado o código, aplique a correção para adequar ao ISP

```
public interface INotificador{
    void EnviarMensagem(string mensagem);
    void EnviarAnexo(string mensagem, byte[] anexo);
}
public class EmailNotifier : INotificador{
    public void EnviarMensagem(string mensagem)    {
        // Lógica para enviar e-mail
    }
    public void EnviarAnexo(string mensagem, byte[] anexo)    {
        // Lógica para enviar e-mail com anexo
    }
}
```

Programação Orientada a Objetos

- SOLID “DIP” Princípio da inversão de controle:
- O Princípio da Inversão de Dependência (Dependency Inversion Principle - DIP) enfatiza a importância de depender de abstrações, não de implementações concretas. O DIP ajuda a reduzir o acoplamento no código, tornando-o mais flexível e fácil de manter.
- Módulos de Alto Nível não devem depender de Módulos de Baixo Nível:
    - Ambos devem depender de abstrações.
- Abstrações não devem depender de Detalhes:
    - Detalhes devem depender de abstrações.

Programação Orientada a Objetos

```
public class EmailService{
    public void EnviarEmail(string destinatario, string mensagem)    {
        // Lógica para enviar e-mail
    }
}


public class Usuario{
    private readonly EmailService emailService;
    public Usuario(EmailService emailService)    {
        this.emailService = emailService;
    }

    public void Notificar(string mensagem)    {
        // Lógica de notificação usando o serviço de e-mail
        emailService.EnviarEmail("usuario@example.com", mensagem);
    }
}
```

```
public interface INotificador{
    void Notificar(string destinatario, string mensagem);
}
public class EmailService : INotificador{
    public void Notificar(string destinatario, string mensagem)    {
        // Lógica para enviar e-mail
    }
}
public class Usuario{
    private readonly INotificador notificador;
    public Usuario(INotificador notificador)    {
        this.notificador = notificador;
    }
    public void Notificar(string mensagem)    {
        notificador.Notificar("usuario@example.com", mensagem);
    }
}
```

Programação Orientada a Objetos

- Exercicio, 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

- Exercicio, agora implemente a ação de salvar um objeto do tipo Pessoa em memoria 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();
         }
      }
   }
```

SENAI

Serviço Nacional de Aprendizagem Industrial

PELO FUTURO DO TRABALHO

0800 048 1212 **sc.senai.br**

Rodovia Admar Gonzaga, 2765 - Itacorubi - 88034-001 - Florianópolis, SC