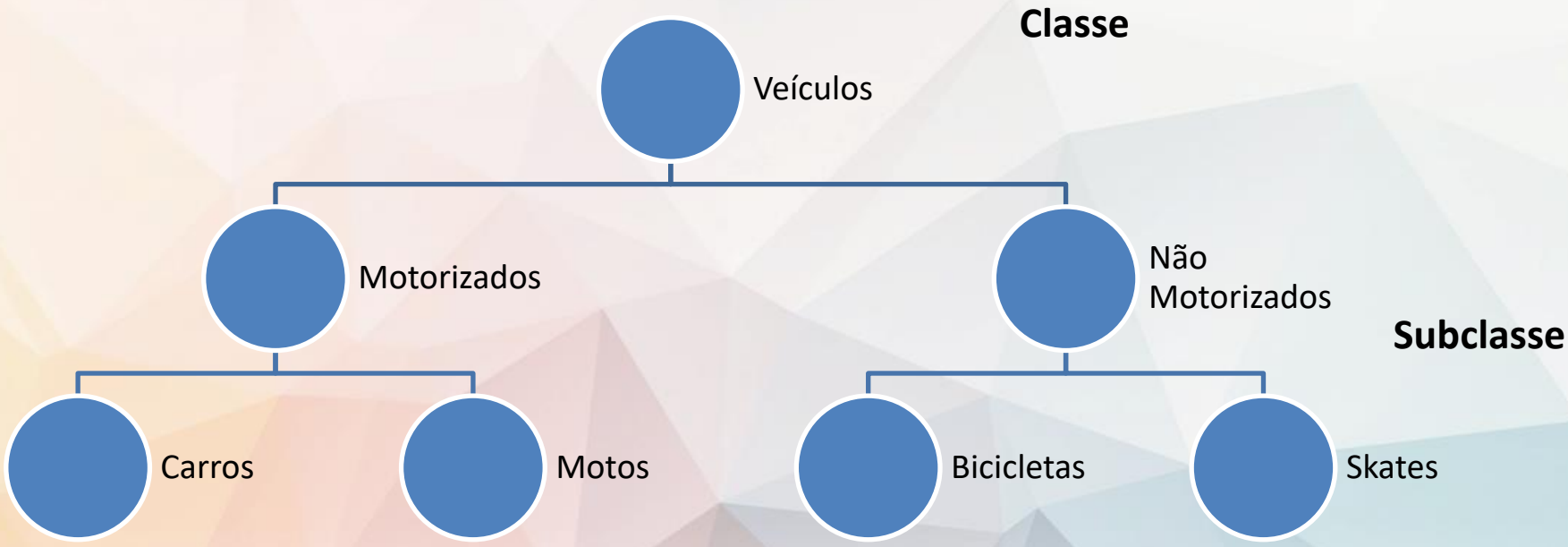
FACET
PÓS-GRADUAÇÃO

Projeto de Banco de Dados e OO .NET

Programação Orientada a Objetos em C#

Prof: Foad Mobini Kesheh
foadmk@gmail.com

Programação Orientada a Objetos



Propriedades
Cor, Peso, Velocidade Máxima, Capacidade

Métodos
Mover(), Acelerar()

Objeto
Vectra GT, Corsa,
Yamaha XTZ, Skate Maha

Projetando Objetos – S.O.L.I.D.

- · *Single Responsibility Principle*: princípio da responsabilidade única;
- · *Open Closed Principle*: princípio do aberto/fechado;
- · *Liskov Substitution Principle*: princípio da substituição de Liskov;
- · *Interface Segregation Principle:* princípio da segregação de *Interfaces*;
- · *Dependency Inversion Principle:* princípio da inversão de dependência.

S: Single Responsibility Principle (SRP)

- Uma classe deve ter apenas um trabalho.

```
public class Retangulo extends Figura{
    public void area()
    {
    }
    public void desenhar()
    {
    }
 }
```

Desenho deveria ser responsabilidade uma outra classe

O: Open closed Principle (OSP)

- Objetos devem ser abertos para extensão sem necessidade de modificação da classe base (fechados para modificação)

```
class Cliente
{
      public double getDesconto(double ValorTotal)
      {
            if (_CustTipo == 1)
            {
               return ValorTotal - 100;
            }
            else
            {
               return ValorTotal - 50;
            }
      }
}
```

Transformar em várias classes derivadas para cada tipo de consumidor com seu próprio método de cálculo.

L: Liskov substitution Principle (LSP)

- A classe estendida não pode alterar o comportamento da classe base. Portanto a classe estendida e a classe base sem que isso altere o resultado

Exemplo de Violação do LSP

```
class Program
  {
     static void Main(string[] args)
     {
        Apple apple = new Orange();

Console.WriteLine(apple.GetColor());
     }
  }
```

Ambas classes deveriam derivar de uma classe Fruit

```
public class Apple
  {
     public virtual string GetColor()
     {
        return "Red";
     }
  }

  public class Orange : Apple
  {
     public override string GetColor()
     {
        return "Orange";
     }
  }
```

I: Interface Segregation Principle (ISP)

- Nenhuma classe que não necessita de determinada interface deve ser forçada a implementar determina da interface.

```
class Linha : IElementosVisuais {
  double Area() {
     throw new Exception(“Linhas não contém área");
  }
}
```

A interface IElementosVisuais deveria ser dividida em diversas interfaces.

Interfaces


S/PDIF
OUT
IN
12
VDC
FIREWIRE
MIDI
IN
OUT
OUTPUTS
LINE
INPUTS
1
2
3
4
3
4

D: Dependency Inversion Principle (DIP)

- Módulos de alto nível não devem depender de módulos de baixo nível. Abstrações não devem depender de detalhes, detalhes devem depender de abstrações.

```
class Forma {
  private MySQLConector myCon;
}
```

A classe MySQLConector deveria ser trocada por uma interface IDBConector

Exercício 1 - POO

- Crie um diagrama de classes simplificado para um Site de Anúncios que onde podem ser anunciados:
  - Residências: casas, apartamentos, sobrados, etc..
  - Automóveis: carros, caminhões, motos, etc..
  - Vagas de Trabalho

Classes

```
class Animal {
        int numeroDePatas;

        public double peso { get; set; }

        string especie;
        public string Especie
        {
                get { return especie; }
                set { especie = value; }
        }

        public Animal()  //Construtor
        {

        }
}
```

Modificadores de Classe

- sealed: Não pode ser estendida
- static: Não pode ser instanciada e deve conter apendas funções e variáveis estáticas
- partial: Classes divididas em vários arquivos

Exercício 2 - POO

- Implemente as classes criadas no exercício 1

Métodos

```
void ImprimeNomeCompleto(string nome, string sobrenome) {
    Console.WrileLine($”{nome} {sobrenome}”);
}

void alteraValor(ref int x) {
    x = 5;
}

void alteraValor(out int x) {
    x = 5;
}
```

Sobrecarga de métodos

- imprimeNome(string nome);
- imprimeNome(string nome, string sobrenome);

Exercício 3 - POO

- Implemente possíveis métodos para as classes do exercício anterior

Herança

```
class Animal {
  public string nome;
  public DateTime
nascimento;

  public void alimentar();
}
```

```
class Cachorro : Animal {
  public string raca;

  public void latir();
}
cachorro = new Cachorro;
Cachorro.alimentar();
```

Encapsulamento

- Public: Acesso irrestrito
- Protected: Acesso limitado a classe e a suas derivadas
- Private: Acesso somente de dentro da classe
- Internal: Acesso somente dentro no mesmo assembly.

Polimorfismo

```
class formas {
  public virtual void Mensagem() {
    Console.writeLine(“Uma forma qualquer”);
  }
}
class triangulos : formas {
  public override void Mensagem() {
    Console.writeLine(“Um triangulo”);
  }
}
```

Exercício 4 - POO

- Utilize herança e polimorfismos nas classes implementadas

Classes Abstratas

- Não podem ser instanciadas
- Podem conter implementação. Override é necessário

```
abstract class formas {
  abstract public void Mensagem();
}
class triangulos : formas {
  public override void Mensagem() {
    Console.writeLine(“Um triangulo”);
  }
}
```

Interfaces

- Não podem ser instanciadas
- Nunca possuem implementação.
- Public é padrão nas interfaces
- Não é necessário override
- Utilizado para definir “contratos” de implementação

```
interface IFormas {
  void Mensagem();
}
class triangulos : IFormas {
  public void Mensagem() {
    Console.writeLine(“Um triangulo”);
  }
}
```

Regras de Herança

- Classe pode estender uma única classe base
- Classe pode estender um ou mais interfaces
- Classe pode estender uma classe base e uma ou mais interfaces