

LINGUAGEM DE PROGRAMAÇÃO

Orientada a objeto

Roteiro da aula

Revisão:

- **Orientação a Objeto**
- **Interface**

Orientação a Objeto

- OO - sistemas com uma **coleção de objetos que interagem entre si**.
- OO permite criar programas, que separam as partes do sistema por responsabilidades e que se comuniquem entre si.
- Permite melhorar a reusabilidade e extensibilidade do software.

Atributos da Classe

- Representam as informações que um objeto contem.
Pode ser:

Campos – são variáveis que podem ser lidas ou alteradas diretamente ou

Propriedades – são lidas ou alteradas como campos, mas são implementadas através de blocos Get e Set, que fornece maior controle sobre os valores.

- Métodos – ações de um objeto
- Para utilizar a classe é necessário instanciar:
`Produto p = new Produto();`
- Para ter acesso aos seu campos propriedades e métodos

Aplicação em 2 camadas


- Ao utilizar classes estamos dividindo a aplicação em:
interface do usuário e
regra de negócios
- No formulário o código fica mais simples
- Na classe ficam os métodos com as regras de negócios

Propriedades de uma classe

- Crie uma variavel local (private) e uma propriedade publica.
- Use GET para retornar o valor da variavel
- Use SET para definir o valor

```
private int codigo  
public int Codigo  
{  
    get { return codigo; }  
    set { codigo = value; }  
}
```

- Padrão camelCase para variaveis locais e PascalCase para classes, metodos e propriedades

- 
- Propriedade automático. Não disponível nas versões express do VisualStudio:
 - Crie variaveis locais (private)
 - TDM – Refactor – Encapsulate Field
 - Altere nome - Ok

Método

- Bloco de código independente que pode ser executado a partir de uma chamada.
- Criado dentro da Classe.
- Dentro dos métodos das classes utilizamos variáveis privadas
- Tipo Void não retorna valores

`void MostraDia();`

- Métodos que retornam valores tem o comando Return

`string ObterHora();`
`{ ... Return hora; }`

- Métodos podem ou não conter parâmetros

parâmetros: declarados na declaração do metodo
usandos somente dentro dos metodos criados

`string MostraDia(int d);`

Revisão

- A4-Exercicio 2

Não devo instanciar a classe quando:

- Atributos estaticos –

`public static string Cor`

```
Classe c = new Classe();  
Classe.Cor
```

- Metodo Estatico –

```
public static void Ligar()  
{   Status = "Carro" + cor + "ligar"; }
```

```
Classe c = new Classe();  
Classe.;Ligar();
```

- A palavra chave **static** faz com que os métodos/atributos estejam associados a classe e não com uma instância particular da classe.

Sobrecarga de método

- Dentro de uma mesma classe, métodos com nome iguais, mas com parâmetros diferentes (assinatura diferente):

```
public void Buscar(int codigo)
{ Status = "Codigo: " + codigo;}
```

```
public void Buscar(string nome)
{ Status = "Nome: " + nome; }
```

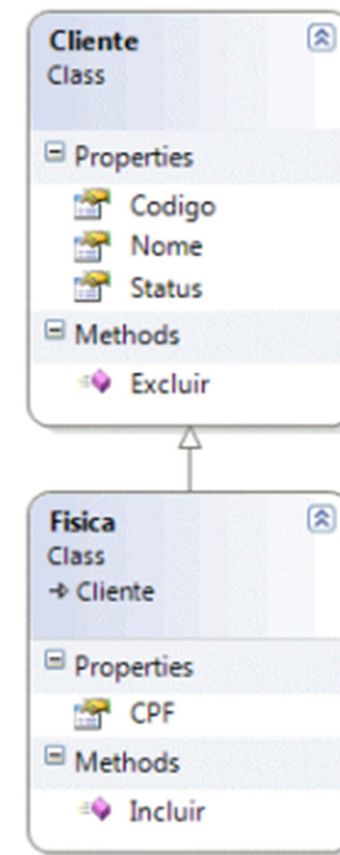
- Quando utiliza o método aparece 2 opções disponíveis na classe

Revisão

- A5-Exercicio 1A e 1B

Herança

- Capacidade de uma classe herdar os atributos e métodos de outra classe.
- Para criar uma classe a partir de outra use dois pontos(:)
- Com herança podemos criar classes mais especializadas
- Cliente é a superclasse e Fisica Derivada ou filha



Sobrescrevendo métodos nas classes derivadas

- Para sobrescrever um método dizemos que o método da classe pai é virtual e na classe filha dizer que estamos sobrescrevendo utilizando as palavras chaves:

virtual – classe pai
override – classe filha

```
public class Cliente
{
    public virtual void Excluir()
    {
        status = codigo + " excluído!";
    }
}

public class Fisica : Cliente
{
    public override void Excluir()
    {
        status = cpf + " excluído!";
    }
}
```

Polimorfismo

- Capacidade de um objeto se comportar como outro em tempo de execução

Classe Abstrata

- É uma classe que não pode ser instanciada
- Ela serve de base para outras classes
- Exemplo a classe Cliente poderia ser abstrata, para utilizar qualquer metodo da classe Cliente poderia fazer isso a partir das classes derivadas

`public abstract class Cliente`

- Finalidade da classe abstrata
 - Servir de base para outras classes
 - Obrigar as classes derivadas a implementar certos métodos

Metodo Abstrato

- Um metodo abstrato não tem corpo, apenas a declaração do método

```
public abstract void Alterar();
```


- Isso significa que as classes derivadas **NECESSARIAMENTE**, que implementar esses metodo

Interface

- Assim como as classes, definem um conjunto de propriedades e métodos.
- Ao contrario das classes, não oferecem implementação das propriedades e métodos (só a assinatura, sem código)
- A classe que implementa uma interface deve criar todas as propriedades e métodos existentes na interface

`class Conexão : IConexao`

- Por padrão utilize a letra “I” na frente do nome da interface.



```
interface IConexao
{
    string Status
    {
        get;
    }
    void Conectar();
    void Desconectar();
}
```

Interface

- Usada para forçar a padronização das classes derivadas
- Havendo Herança e Interfaces, primeiro vem a herança e depois a interface separadas por virgulas:

```
class Func : Conexao, ICadastro, IEndereco
```

Revisão

- A5-Exercicio 2 e 3