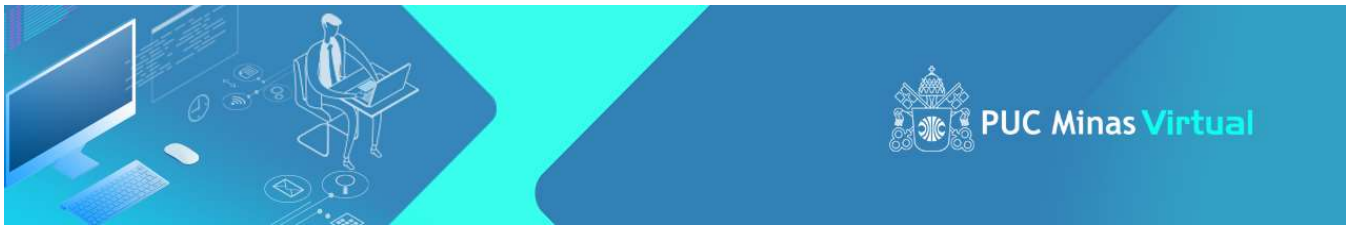
UNIDADE 2 - Tema 1 - Atributos, Propriedades e Métodos de Classe



Microfundamento: Algoritmos e abstração de dados

Eixo (https://pucminas.instructure.com/courses/49042) > **Microfundamento (https://pucminas.instructure.com/courses/49011/pages/microfundamento-algoritmos-e-abstracao-de-dados)** > **UNIDADE 2 - Tema 1 - Atributos, Propriedades e Métodos de Classe**

ATRIBUTOS, PROPRIEDADES E MÉTODOS DE CLASSE

Nesta sessão vamos continuar a evoluir na construção de classes. Temos que saber como identificar as entidades e informações sobre um problema e depois modelá-las em tipos abstratos de dados.

Os campos de dados de uma estrutura, formados por tipos primitivos de dados, como inteiros e *strings*, por exemplo, irão ganhar um novo nome: Atributo. E vamos fazer mais. Vamos aprender a criar propriedades de classes e encapsular métodos dentro das classes. Vamos entender o que significa Ocultação de Informação.

NESTA SESSÃO

Tipo Abstrato de Dados como Modelo

É muito importante que você leia com atenção a essas duas afirmações:

- “Uma classe (...) especifica uma estrutura de dados e os métodos operacionais permissíveis que se aplicam a cada um de seus objetos.” (MARTIN e ODELL, 1995)
- “Um objeto é qualquer coisa, real ou abstrata, a respeito da qual armazenamos os dados e

Um objeto é qualquer coisa, real ou abstrata, a respeito da qual armazenamos os dados e os métodos que os manipulam.” (MARTIN e ODELL, 1995)

Suponha que você precise desenvolver um *software* para uma clínica odontológica. Eles precisam controlar melhor a agenda de pacientes. Como modelar isso através de Tipos Abstratos de Dados?

Primeiramente, temos que identificar as fontes de informação, os dados pertinentes ao problema que precisam ser armazenados. Informações sobre Pacientes é importante, é algo que precisamos fazer, correto? A Agenda também possui elementos que precisam ser guardados para, no futuro, serem recuperados. E por aí vai... Paciente pode ser uma classe, você com certeza concorda com isso.

Que informações sobre um Paciente teria um *software* como esse? Em outras palavras, um objeto do tipo Paciente teria quais dados armazenados? Nome, Endereço, Telefone de Contato...

Mas tem mais uma coisa: uma vez localizadas as informações importantes que precisamos trabalhar temos que identificar o que fazemos com elas, como manipulamos essas informações. O que fazemos com um objeto Paciente? Que funções, ou métodos, esse Tipo Abstrato de Dados deveria ter? Cadastrar Paciente, Pesquisar Paciente, Atualizar Dados de Paciente...

Você compreende o modelo? Definimos um tipo novo de dados que irá encapsular informações importantes sobre algo e os métodos que irão trabalhar sobre esses dados.

Atributos, Propriedades e Métodos de Classe

Tanto para as estruturas quanto para as classes, criamos campos de informação de diversos tipos primitivos que, afinal, iriam compor o nosso tipo abstrato de dados:

```
class Pessoa
{
    public string Nome;
    public double Salário
}
```

Na classe Pessoa, Nome e Salário são chamados de atributos. São os elementos que definem a estrutura do tipo abstrato de dados Pessoa e também são conhecidos como variáveis de uma classe.

Embora isso funcione normalmente quando usamos essa classe como tipo, não devemos fazer assim. Observe uma “reconstrução” da classe Pessoa:

```
class Pessoa
{
    private string _Nome;

    public string Nome
    {
        get { return _Nome; }
        set { _Nome = value; }
    }
```

```
    }

    private double _Salário;

    public double Salário
    {
        get { return _Salário; }
        set { _Salário = value; }
    }
}
```

Você consegue perceber a diferença? Agora nós temos um atributo, uma variável de classe chamada "***_Nome***", e outra, chamada "***_Salário***". É comum utilizarmos o caractere "_" antes do nome de uma variável. Observe que eles também são do tipo "***private***" e não "***public***"

como estávamos fazendo até agora. Significa que esses campos só podem ser acessados de dentro da classe.

E como acessar as informações desses campos de fora dessa classe, como em *main*, por exemplo?

Observe:

```
public string Nome
{
    get { return _Nome; }
    set { _Nome = value; }
}
```

Agora temos uma declaração de uma string ***Nome***, pública, visível fora da classe. E essa declaração possui duas "***funções***" especiais, digamos assim, ***get*** e ***set***. O ***get*** é acessado quando queremos acessar o conteúdo do campo **_Nome**. O set é acessado quando queremos atribuir para o campo **_Nome** algum valor.

Observe que tanto o ***get*** quanto o ***set*** são rotinas próprias, delimitadas pelos caracteres **{** e **}**. Possuem lógica, podemos colocar código normalmente nessas funções. O ***get*** retorna o valor que está no campo **_Nome** e o ***set*** armazena um valor (que é definido pela palavra-chave value) no campo.

Quando definimos um atributo como privado e funções próprias que manipulam o valor desse atributo dentro da classe, estamos definindo uma Propriedade. Propriedade é um atributo público e que possui "acessadores" próprios, o ***get*** e o ***set***.

Preste atenção no programa a seguir que usa a classe Pessoa. Observe as linhas assinaladas:

```
class Program
{
    static void Main(string[] args)
    {
        Pessoa xPes = new Pessoa();

        xPes.Nome = "Gustavo Campos Nascimento";
        xPes.Salário = 4567.89;

        Console.WriteLine($"{xPes.Nome}");
        Console.WriteLine($"Salário: R${xPes.Salário}");
```

```
        }
    }
}
```

Toda vez que um valor é atribuído a uma propriedade o ***set*** é acionado para colocar o valor ***(value)*** dentro do campo da classe.

Toda vez que queremos usar o valor de uma propriedade, o ***get*** é acionado para retornar o valor contido no campo da classe.

Você compreendeu bem esses conceitos?

Pois bem, um Tipo Abstrato de Dados encapsula os dados e também as funções que trabalham esses dados. Falamos sobre isso antes... Podemos definir também métodos dentro de uma classe. Métodos são funções de uma classe, também chamados de Serviços.

Veja a classe a seguir:

```
class Círculo
{
    private double _Raio;

    public double Raio
    {
        get { return _Raio; }
        set { _Raio = value; }
    }

    public double CalculaÁrea()
    {
        return Math.PI * Math.Pow(_Raio, 2);
    }
}
```

A classe Círculo possui um atributo de nome **_Raio** (***private***), define uma propriedade de nome Raio (***public***) e também um método (***public***) **CalculaÁrea**(),

Poderíamos então fazer:

```
static void Main(string[] args)
{
    double Área;

    Círculo xCírculo = new Círculo();

    xCírculo.Raio = 2.5;

    Área = xCírculo.CalculaÁrea();

    Console.WriteLine($"Á Área do Círculo é {Área:F2}");

    Console.ReadKey();
}
```

O que temos aqui é um objeto x Círculo, da classe Círculo, que “sabe” armazenar o valor de Raio e calcular a área de um círculo. Entenda esse conceito. Um programador utiliza essa estrutura, mas, em síntese, não sabe como ela foi implementada. Está lembrado do Math.Pow(a,b)? Ele sabe o que a definição faz, mas não vê a forma como foi feita.

A esse princípio damos o nome de Ocultação de Informação. Quando encapsulamos os atributos de uma classe (com ***get*** e ***set***) e também os seus métodos definidos, fazemos com que seus objetos escondam sua implementação. É uma função que está dando acesso às informações, temos uma lógica embutida nessa definição.

O encapsulamento protege os dados de um objeto contra um uso eventual, até mesmo não intencional. Ele oculta do usuário os detalhes sobre a lógica interna usada na implementação de um objeto, separa o comportamento que ele possui da sua implementação.

O código a seguir mostra uma lógica colocada no set da propriedade Raio:

```
class Círculo
{
    private double _Raio;

    public double Raio
    {
        get { return _Raio; }
        set
        {
            if (value < 0)
                _Raio = 0;
            else
                _Raio = value;
        }
    }

    public double CalculaÁrea()
    {
        return Math.PI * Math.Pow(_Raio, 2);
    }
}
```

O programador que usar essa classe não tem acesso ao campo ***_Raio***. Valores negativos que ele quiser dar ao campo passam pela definição de sua propriedade **Raio** que, por meio de uma estrutura *if-else,* “cerca” esses valores negativos e atribui 0 (zero) ao campo se for o caso.

Assista ao vídeo abaixo para saber mais.

Mais tarde vamos citar duas propriedades, que na verdade são fatores externos de qualidade de um produto de *software*, chamadas Correção e Robustez. A primeira garante o funcionamento do *software* como previsto, como especificado, e a segunda garante a sua plena execução mesmo em condições imprevistas

Não há dúvida que agora precisaremos exercitar esse conteúdo.

> **REFLEXÃO**
>
> Você percebe a utilidade e a necessidade de termos maior controle sobre os objetos de uma classe? Entendeu bem o sentido de substituirmos o acesso livre e direto a um atributo por uma função?

Faça um programa que defina de forma completa uma classe de nome "Veículo" com os atributos "Nome", "Marca", "Ano de Fabricação" e "Placa". Crie métodos capazes de ler os dados de um veículo e mostrar uma listagem de todos os veículos.

Cadastre em um vetor 30 veículos.

Clique **aqui (https://pucminas.instructure.com/courses/49011/files/2899732?wrap=1)** ↓ **(https://pucminas.instructure.com/courses/49011/files/2899732/download?download_frd=1)** para conferir o gabarito.

↑ *Voltar ao topo*

TIPOS ABSTRATOS DE DADOS - CLASSES - IMPLEMENTAÇÃO

Tema 1: Tipos Abstratos de Dados

Definição de um TAD - Classes e Objetos
(https://pucminas.instructure.com/courses/49011/pages/unidade-2-tema-1-definicao-de-um-tad-classes-e-objetos)

Atributos, Propriedades e Métodos de Classe

Mecanismos de Visibilidade/Acessibilidade
(https://pucminas.instructure.com/courses/49011/pages/unidade-2-tema-1-mecanismos-de-visibilidade-slash-acessibilidade)

Construtores de Classe
(https://pucminas.instructure.com/courses/49011/pages/unidade-2-tema-1-construtores-de-classe)

Síntese e referências
(https://pucminas.instructure.com/courses/49011/pages/unidade-2-sintese-e-

(https://pucminas.instructure.com/courses/49011/pages/unidade-2-sintese-e-referencias)

Atividade objetiva 2
(https://pucminas.instructure.com/courses/49011/quizzes/171709)