Paradigmas de Programação

Paradigmas de programação são modos diferentes de montar a estrutura e execução de urn código. Um paradigma de programação fornece e determina a visão que o programador possui sobre a estruturação e execução do programa. Cada paradigma surgiu de necessidades diferentes. Dado isso, cada um apresenta maiores vantagens sobre os outros dentro do desenvolvimento de determinado sistema. Sendo assim, um paradigma pode oferecer técnicas apropriadas para uma aplicação especifica. Entre eles, estão a programação orientada a objetos (POO) e a programação estruturada (PE).

Programação Estruturada

A programação estruturada tem como principal característica sua interpretação linha por linha, em pequenos trechos de código, podendo eles não estar em uma ordem específica. Há três tipos de estruturas básicas para navegar pelo código: sequência, seleção e repetição.

- **sequência**s: são os comandos a serem executados de cima para baixo, linha a linha do programa, de forma sequencial;

- **seleções**: sequências que só devem ser executadas se uma condição for satisfeita (exemplos: if-else, switch e comandos parecidos);

- **repetições**: sequências que devem ser executadas repetidamente ate uma condição for satisfeita (for. while. do-while etc).

Programação Orientada a Objetos

A POO é baseada no conceito de "objetos", que podem conter dados na forma de campos (atributos) e códigos na forma de procedimentos (métodos). Uma característica é que um procedimento de objeto pode acessar, e geralmente modificar, os campos de dados do objeto com o qual eles estão associados e cada um é capaz de receber, processar e enviar dados, podendo ser visto como uma "máquina independente". Para que uma linguagem possa ser considerada orientada a objeto é preciso atender a quatro tópicos bastante importantes, e esses tópicos são chamados de "Os 4 pilares da Programação Orientada a Objeto". Sao eles:

- Abstração;

- Encapsulamento;

- Herança;

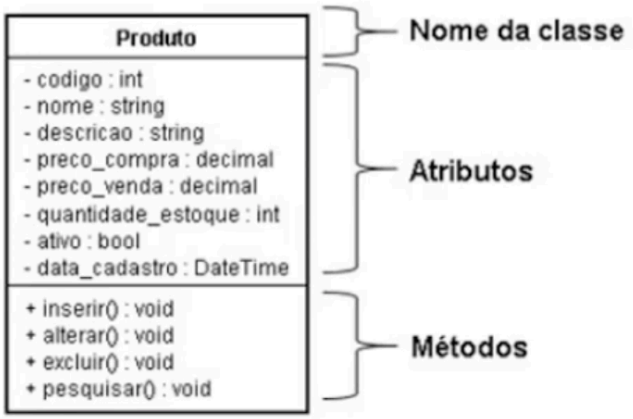
- Polimorfismo.

Orientação a objetos em .NET

A importância da POO é simples e direta. Tudo em .NET é objeto. Mesmo os tipos de dados mais simples são considerados objetos, já estes também contêm métodos e propriedades. Implicitamente, todo e qualquer tipo ou objeto em .NET possui urn ancestral comum.

Classes

Podem ser consideradas como se fosse um molde para o objeto, contendo dentro de si as principals informações para a sua criação. Define os atributos e métodos comuns que serão compartilhados por um objeto.




```
1  public class Produto //Nome da classe
2  {
3    private int codigo; //Atributo da classe
4      private string nome; //Atributo da classe
5      private decimal preco; //Atributo da classe
6
7      public int Codigo { get => codigo; set => codigo = value; } //Propriedade da classe
8      public string Nome { get => nome; set => nome = value; } //Propriedade da classe
9
10     public decimal Preco { get => preco; set => preco = value; } //Propriedade da classe
11   }
```

Objetos

Considera-se um objeto tudo aquilo que em geral possui atributos, comportamentos e um estado. A classe em si é um conceito abstrato, como urn molde, que se torna concreto e palpável através da criação de um objeto. Na programação o objeto é uma instanciação (criação a partir) de uma classe. Para criarmos ou "instanciarmos" objetos de uma determinada classe, utilizamos o operador new.

```
1   Produto obj = new Produto();
```

Visibilidade

A visibilidade é importante para a proteção do código e suas funcionalidades, define quern pode alterar cada dado dos trechos de código em 3 principals níveis:

- Pública (representada pelo símbolo "+");
- Privada (representada pelo símbolo "-");

- Protegida (representada pelo simbolo “#").

Modificador	Significado
public	Sem limitação de acesso.
protected internal	Acesso limitado à própria classe, às classes derivadas e ao próprio assembly
protected	Acesso limitado à própria classe, às classes derivadas.
internal	Acesso limitado ao próprio assembly.
private	Acesso limitado à própria classe.

Esse encapsulamento de atributos e métodos impede o chamado vazamento de escopo, onde um atributo ou método é visível por alguém que não deveria vê-lo, como outro objeto ou classe. Isso evita a confusão do uso de variáveis globais no programa, deixando mais fácil de identificar em qual estado cada variável vai estar a cada momento do programa, já que a restrição de acesso nos permite identificar quem consegue modificá-la.

Tipos por Valor

O c# tem duas grandes categorias de tipos: por valor e por referência. Os tipos por valor são gerenciados diretamente e têm as seguintes características principais:

- Todos os tipos de dados numéricos.
- Não precisam ser inicializados com o operador new.
- A variável armazena o valor diretamente.
- A atribuição de uma variável a outra copia o conteúdo, criando efetivamente outra cópia da variável.
- Não podem conter o valor null.
- Exemplos de variáveis desse tipo são: integers, doubles, floats e char.

Inteiros

Os tipos inteiros (integers) têm sempre o mesmo significado, independentemente da implementação.

Double e Float

Os números de ponto flutuante são bastante convencionais, as operações de ponto flutuante não geram erros.

- Double: ponto flutuante binário com 15 dígitos decimais de precisão.

- Float: ponto flutuante binário com 7 dígitos decimais de precisão.

Caracteres

Em C#, todos os caracteres (char) são armazenados no padrão Unicode e usam 16 bits por caractere. O Unicode permite armazenar os caracteres de todas as línguas vivas (como grego, japonês, chinês e coreano) e algumas mortas (como o sânscrito).

Tipos por referência

Um tipo por referência armazena uma referência a seus dados. Os tipos de referência incluem o seguinte:

- Duas variáveis poderem conter a referencia a um mesmo objeto.
- Operações em uma afetarem a outra.
- Todas as matrizes, mesmo que seus elementos sejam de tipos de valor.
- Exemplos de tipos por referencia: Strings, classes e arrays.

Strings

Semelhante ao char, strings são variáveis do tipo texto. São uma sequência de caracteres, geralmente utilizada para representar palavras, frases ou textos de um programa. As strings são consideradas imutáveis e não podem ser alteradas depois de criadas. Quando voce efetua uma operação qualquer, como por exemplo, concatenar um caractere, você na verdade está criando outra string e descartando a anterior.

Classes

Como visto anteriormente é um tipo definido pelo usuário e correspondem a uma class. As classes são sempre derivadas de object e podem conter campos, métodos e propriedades. Uma classe pode derivar de uma única outra classe, e também de varias interfaces.

Arrays

Um array (matriz) é uma lista de valores onde todos os valores no grupo são referenciados pelo nome da matriz e o índice atribuído ao valor especifico na matriz.

Tipo Ponteiro

Um ponteiro ou apontador é um tipo de dado cujo valor se refere diretamente a um outro valor alocado em outra área da memória, através de seu endereço.

Propriedades

Uma forma mais inteligente de fazer implementações de campos em classes são as propriedades. Propriedades consistem em um par de métodos "get" e "set" que respectivamente servem para recuperar e atribuir o valor a urn campo. Geralmente, para cada método existe uma variável dentro da classe que armazena o valor da propriedade.

Eventos

São mensagens que a classe dispara em determinada situação. Para que o evento funcione, é necessário que urn método seja escrito para ser executado quando ocorrer o evento. A classe apenas fica sabendo que existe esse método em tempo de execução. Para que o mecanismo dos eventos funcione, é necessário usar urn tipo de estrutura chamado Delegate, uma variável que guarda o endereço de uma função. Assim, quando o evento é disparado, essa variável chama a função associada a ela.