

Disciplina: POO - Programação Orientada a Objetos

Introdução

Prof^a. Dr^a. Giovana Angélica Ros Miola
giovana.miola@fatec.sp.gov.br

Fatec
Presidente Prudente

CPS
Centro
Paula Souza

Conteúdo Programático

- Conceitos e evolução da tecnologia de orientação a objetos.
- Limitações e diferenças entre o paradigma da programação estruturada em relação à orientação a objetos.
- Conceito de objeto, classe, métodos, atributos, herança, polimorfismo, agregação, associação, dependência, encapsulamento, mensagem e suas respectivas notações na linguagem padrão de representação da orientação a objetos.
- Implementação de algoritmos orientado a objetos utilizando linguagens de programação C#
- Aplicação e uso das estruturas fundamentais da orientação a objetos.

Provas 2º sem/2024

- P1:
 - 30/09/2024
- P2:
 - 25/11/2024
- P3:
 - 02/06/2024

Calendário Acadêmico

	D	S	T	Q	Q	S	S
AGOSTO					01	02	03
	04	05	06	07	08	09	10
	11	12	13	14	15	16	17
	18	19	20	21	22	23	24
	25	26	27	28	29	30	31
SETEMBRO	01	02	03	04	05	06	07
	08	09	10	11	12	13	14
	15	16	17	18	19	20	21
	22	23	24	25	26	27	28
	29	30	P1				
OUTUBRO			01	02	03	04	05
	06	07	08	09	10	11	12
	13	14	15	16	17	18	19
	20	21	22	23	24	25	26
	27	28	29	30	31		

	D	S	T	Q	Q	S	S
NOVEMBRO						01	02
	03	04	05	06	07	08	09
	10	11	12	13	14	15	16
	17	18	19	20	21	22	23
	24	25	26	27	28	29	30
DEZEMBRO	01	02	03	04	05	06	07
	08	09	10	11	12	13	14
	15	16	17	18	19	20	21
	22	23	24	25	26	27	28
	29	30	31				

TOTAL DE SEMANAS LETIVAS

D	S	T	Q	Q	S	S
-	19	20	19	20	19	15

REPOSIÇÕES:

Segunda Feira	24/08
Quarta Feira	21/09
Sexta Feira	19/10
Sábado	26/10 - 09/11 - 23/11- 30/11 e 07/12

- Alunos do Noturno- reposições no período vespertino

Programação Orientação a Objetos

- A POO é um paradigma de programação baseado em objetos, que se destaca pela organização e reutilização do código. Ela se baseia na representação de elementos do mundo real como objetos, que possuem atributos (dados ou características) e métodos (ações ou comportamentos).
- A Orientação a Objetos é um paradigma de análise, projeto e programação de sistemas, baseado na composição e interação entre diversas unidades de software chamadas de objetos
- A OO não se limita apenas em ser uma nova forma de programação. Ela também se preocupa com a modelagem (análise e projeto) dos processos/tarefas que devem ser realizados
- Mais do que um tipo de "linguagem de programação", a OO é uma nova forma de se pensar e representar de maneira mais realista, as necessidades dos softwares
- Ela não é um paradigma que inventou algo realmente revolucionário, mas, na verdade, facilitou o processo de programação a partir do que já existia

Surgimento da POO

- POO, objetivo aproximar o mundo real do digital, nova filosofia de desenvolvimento
- Na década de 40 ou 50 tinha a linguagem de **baixo nível**, se tivesse computador era binário, usava linguagem binária ou decimal
- Linguagem de **alto nível**,
 - Programação **linear**
 - Algoritmo, programação **estruturada**, origem a sistemas, mas com o crescimento dos sistemas a programação estruturada começou a **falhar** em alguns conceitos
 - Evoluiu para programação **modular**, permitia que criasse pequenos módulos estruturados valorizando dados e funcionalidade e colocava eles em pequenas **capsulas** protegidas que poderiam compor sistemas maiores
 - Teve um pequeno tempo de vida, devido ao surgimento do novo paradigma de **programação orientada a objetos**

POO

- 1970 - Alan Kay
 - Matemática – Biologia - Educação
 - **Postulado:** “O computador ideal deve funcionar como um organismo vivo, isto é, cada célula se relaciona com outras a fim de alcançar um objetivo, mas cada uma funciona de forma autônoma, as células poderiam também reagrupar-se para resolver um outro problema ou desempenhar outras funções”
- Trabalhou na Xerox, Apple, Disney e HP
- Autor da linguagem de programação Smalltalk 71 a 80
- Alguns conceitos de POO já eram utilizados na linguagem Simula 67, criada por Ole Johan Dahl e Kristen Nygaard, faleceram em 2001



Dynabook – Alan Kay

- Um computador pessoal para crianças de todas as idades
- Dispositivo educacional portátil conceitual, como o tablet
- Parte da motivação e do financiamento para o projeto Dynabook, veio da necessidade de manutenção militar (Departamento de Defesa dos Estados Unidos) de forma portátil como:
 - reparo e documentação de operações;
 - eliminar, mover grandes quantidades de papéis de difícil acesso.



Alan Kay segurando a maquete do Dynabook. (5 de novembro de 2008 em Mountain View, [CA](#)) **Tinha 68 anos**



Vantagens de Utilização da POO

- **Confiável:** a geração de código baseado no conceito de objetos e classes fornece uma grande independência ao programa. Assim, qualquer intervenção que seja necessária não afetará outros pontos do sistema. Isso confere robustez e confiabilidade.
- **Oportuno:** a criação paralela de código torna todo o processo bastante ágil. Ganha-se em tempo e eficiência, tornando um software com paradigma POO oportuno. Várias equipes podem trabalhar no mesmo projeto de forma independente.

Vantagens de Utilização da POO

- **Ajustável:** essa característica diz respeito ao processo de manutenção do código-fonte. Ao atualizar uma parte pequena, o conceito de herança garante que, automaticamente, todas as partes que utilizarem tal método sejam beneficiadas. Essa característica torna especial o paradigma de POO, pois é muito comum que equipes de desenvolvimento de softwares sejam escaladas para trabalhar com softwares já existentes. Dessa forma, torna-se mais fácil executar o trabalho de manutenção.
- **Extensível:** o uso do princípio da reutilização do software adiciona funcionalidades ao sistema já existente. Não é preciso “reinventar a roda”, reescrevendo o código. Isso confere maior capacidade de estender um sistema já existente.

Vantagens de Utilização da POO

- **Reutilizável:** um mesmo objeto pode ser utilizado em aplicações diferentes, desde que sejam compatíveis. Se tivéssemos um objeto “aluno”, por exemplo, ele poderia ser utilizado em sistemas de empresas diferentes, desde que elas contassem com alunas e alunos na sua estrutura. Assim, tanto uma escola de música como uma academia poderiam fazer uso do objeto “aluno” em um possível software para uso próprio, pois ambas as empresas têm essas pessoas como seu público.
- **Natural:** o conceito de POO que traz para a programação o mundo concreto, tal qual vemos no dia-a-dia, faz com que se ganhe naturalidade. O ponto de vista humano se torna mais próximo do virtual. Assim, pensa-se no problema geral, em vez de concentrar o foco em pormenores.

Comparação entre Paradigmas

- **Paradigma Estruturado/Procedural**

- Tipos de Dados
- Variável
- Função
- Chamada de Função

- **Paradigma de Objetos**

- Classes
- Objeto instanciado
- Operação / Método Serviço
- Envio de Mensagem

Pilares da Orientação a Objetos

- Para uma linguagem de programação ser considerada orientada a objetos, deve haver quatro comportamentos característicos.

São eles:

- Encapsulamento
- Herança
- Polimorfismo
- Abstração

Pilares da Orientação a Objetos

Encapsulamento

- Ocultação da informação, blinda-se o aspecto interno do objeto em relação ao mundo exterior, que tem como finalidade evitar resultados inesperados, acessos indevidos, entre outros problemas.
- O encapsulamento é a capacidade que determinado método ou atributo de um objeto tem de se manter invisível. Ou seja, ele continua funcional, mas sem mostrar como. Sabe-se o que faz, mas não sabe como se faz.
- Para o programa, pouco interessa como determinado método é construído. O que de fato importa é que ele faça aquilo que promete, pois assim poderá ser utilizado com eficiência.
- Isso garante uma camada extra de proteção para a aplicação, pois os detalhes de implementação não são revelados. Se essa é a intenção da pessoa que faz a programação, ela é garantida por meio da declaração de que aquele método é privado e não público.

Pilares da Orientação a Objetos

Herança

- A pior prática em programação é a repetição de código. Isto leva a um código frágil, propício a resultados inesperados e difícil de manutenção
- Como o próprio nome diz, trata-se de uma relação de receber algo pré-existente. No caso da POO, a herança é um evento que ocorre entre classes. A classe que concede características é chamada de classe base/superclasse. Já a classe que herda é chamada de classe derivada.
- Quando ocorre uma herança, a classe derivada herda as características da classe base. Isso é **bastante útil para um reaproveitamento de código**, pois não seria necessário refazer algo que já existe. Parte-se de um ponto e se desenvolvem novos métodos.
- Ocorre também reuso de código na associação entre classes, mas é diferente, uma classe pede ajuda a outra para poder fazer o que ela não consegue fazer por si só, ou seja, uma classe fornece uma porção de código a outra, evitando a repetição de código. Os códigos ficam mais robustos, manuteníveis e fáceis de entender.

Pilares da Orientação a Objetos

Polimorfismo

- Já o polimorfismo é uma característica referente aos métodos dos objetos.
- Significa dizer que um mesmo método pode ser utilizado em diferentes objetos, de diferentes classes, assim ocorre o tratamento de objetos de diferentes classes de forma uniforme
- Pode-se imaginar esse tipo de ação ocorrendo num sistema bancário: o extrato (método) mostra a movimentação da conta de clientes de determinada categoria (objeto).
- Pode-se usar essa funcionalidade para clientes no geral. Além disso, ele pode ser utilizado no sistema de outros bancos (classes).

Pilares da Orientação a Objetos

Abstração

- A abstração simplifica a complexidade, definindo objetos e classes como representações de entidades do mundo real.
- Conhecida como interface ou *template*. A ideia é representar um objeto de forma abstrata, que seja obrigatoriamente herdado por outras classes.
- Processo pelo qual se isolam características de um objeto, considerando os que tenham em comum certos grupos de objetos.
- Concentra-se apenas nos aspectos essenciais, pois permite, a partir de um contexto inicial, modelar necessidades específicas.
- Isso possibilita flexibilidade no processo de programação, pois é possível não trabalhar com o conceito alvo diretamente, mas sim com suas abstrações.

Preconceitos sobre a Orientação a Objetos

- A orientação a objetos não é uma metodologia para o desenvolvimento de interfaces gráficas amigáveis, ou seja, o paradigma de objetos não está necessariamente relacionada a programação visual;
- Padrões de projeto não são necessariamente abordagens relacionadas ao paradigma de objetos;
- **A orientação a objetos não garante a reutilização de código, ela oferece mecanismos para que isso ocorra, mas sempre será função do desenvolvedor garantir isso.**

Linguagens Orientadas a Objetos

- Distinção entre as linguagens de programação:
 - **Híbridas:** *São linguagens que originalmente **não** foram projetadas orientadas a objetos, mas que passaram a incorporar os conceitos deste paradigma.*
 - Ex: Visual Basic, C++, PHP, Object Pascal (Delphi ambiente de desenvolvimento)
 - **Puras:** *São linguagens que foram projetadas originalmente orientadas a objetos.*
 - Ex: Smalltalk, Java, C#, Python.

C#

- O Visual C# (ou apenas C#) é uma linguagem de programação da Microsoft projetada para criar aplicações diversas, tanto para Windows, como para a Web, que são executadas no .NET Framework.
- É uma linguagem simples, moderna, segura quanto a tipos, orientada a objetos e familiar a programadores C, C++ e Java, pois destas herda várias características. Embora herde características dessas linguagens, C# traz novos recursos e conceitos de programação, tais como indexadores, propriedades e delegates.
- O código de C# é compilado como um código gerenciado, isto quer dizer que ele se beneficia dos serviços do Common Language Runtime (CLR), que incluem interoperabilidade de linguagens, garbage collection, segurança e melhor suporte ao controle de versões.
- O seu ambiente de desenvolvimento é altamente interativo com designers visuais para a criação das aplicações. Da suíte Visual Studio, que contempla também o VB.NET, C# é a sua linguagem principal com um número crescente de usuários.
- C# está se posicionando como o paradigma no desenvolvimento de aplicações no ambiente Windows.

C# - Histórico

- A linguagem C# foi criada junto com a arquitetura .NET. Embora existam várias outras linguagens que suportam essa tecnologia (como VB.NET, C++, J#), C# é considerada a linguagem símbolo do .NET pelas seguintes razões:
 - Foi criada praticamente do zero para funcionar na nova plataforma, sem preocupações de compatibilidade com código de legado.
 - O compilador C# foi o primeiro a ser desenvolvido.
 - A maior parte das classes do .NET Framework foram desenvolvidas em C#.
- A criação da linguagem, embora tenha sido feita por vários desenvolvedores, é atribuída principalmente a Anders Hejlsberg, hoje um Distinguished Engineer na Microsoft.
- Anders Hejlsberg era desenvolvedor de compiladores na Borland, e entre suas criações mais conhecidas estão o Turbo Pascal e o Delphi.

O porquê do nome C#?

- Muitos pensam que o nome C# viria de uma sobreposição de 4 símbolos "+" dando a impressão de "++++".
- Na verdade o "#" de C# refere-se ao sinal musical (sustenido), que aumenta em 1/2 tom uma nota musical.
- O símbolo real seria o \sharp e não o #, porém, devido a limitação de telas, fontes e alguns browsers, no momento da normalização junto a ECMA, fora especificado apenas que o nome da linguagem seria uma letra C maiúscula e o sinal "#", facilitando assim, publicações e artigos com um caracter encontrado facilmente dos layouts de teclado padrões.
- Desta forma, caso o nome fosse usado em português, seria "C-Sustenido" (ou "Dó-Sustenido"), e não "C-cerquilha".

Tipos primitivos de dados

Tipo	Tamanho	Valores Possíveis
bool	1 byte	true e false
byte	1 byte	0 a 255
sbyte	1 byte	-128 a 127
short	2 bytes	-32768 a 32767
ushort	2 bytes	0 a 65535
int	4 bytes	-2147483648 a 2147483647
uint	4 bytes	0 to 4294967295
long	8 bytes	-9223372036854775808L to 9223372036854775807L
ulong	8 bytes	0 a 18446744073709551615
float	4 bytes	Números até 10 elevado a 38. Exemplo: 10.0f, 12.5f
double	8 bytes	Números até 10 elevado a 308. Exemplo: 10.0, 12.33
decimal	16 bytes	números com até 28 casas decimais. Exemplo 10.991m, 33.333m
char	2 bytes	Caracteres delimitados por aspas simples. Exemplo: 'a', 'ç', 'o'

Tipos primitivos e Classes associadas

Tipo	Classe associada
bool	System.Boolean
byte	System.Byte
char	System.Char
decimal	System.Decimal
double	System.Double
float	System.Single
int	System.Int32
long	System.Int64

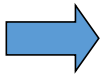
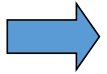
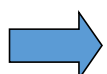

Operadores Aritméticos

- Os operadores aritméticos são utilizados para obter resultados numéricos.
- Os símbolos para os operadores aritméticos são:

Operador	Ação
+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Módulo da divisão (resto)
++	Incremento de 1
--	Decremento de 1

Operadores de Incremento e Decremento

Operador	Exemplo	Significado
++	++a	adicionar 1 à variável a e depois calcular a expressão na qual a reside
	a++	calcular a expressão na qual a reside e depois adicionar 1 à variável a
--	--a	subtrair 1 da variável a e depois calcular a expressão na qual a reside
	a--	calcular a expressão na qual a reside e depois subtrair 1 da variável a

Exemplos	Variável	Valor final
 a = 10;	a	10
 b = a++;	b	10
 c = a;	c	11
 d = ++a;	d	12

Expressões Lógicas - Operadores Relacionais

- Os operadores relacionais são utilizados para realizar **comparações** entre dois valores do tipo básico.
 - Estes valores são representados por constantes, variáveis ou expressões aritméticas, e retornam valores lógicos (verdadeiro ou falso).

Operador	Função	Exemplo
==	Igual a	3 == 3, x == y
!=	Diferente de	3 != 9, x != y
>	Maior que	5 > 4, x > y
<	Menor que	3 < 6, x < y
>=	Maior ou igual a	5 >= 3, x >= y
<=	Menor ou igual a	3 <= 5, x <= y

Expressões Lógicas – Operadores Lógicos

Operador	Significado	Função	Funcionamento
!	Não (not)	Negação	Nega ou inverte o valor lógico de um elemento.
&&	E (and)	Conjunção	Exige que todos os termos da expressão sejam verdadeiros para a expressão inteira seja verdadeira.
	Ou (or)	Disjunção	Exige que apenas um dos termos da expressão seja verdadeiro para a expressão inteira seja verdadeira.

Exemplos:

$2 < 5 \ \&\& \ 15 / 3 == 5$
V && 5 == 5
V && V
V

$2 < 5 \ || \ 15 / 3 == 5$
V || V
V

Alguns exemplos de funções matemáticas e constantes existentes na classe `System.Math`

FUNÇÃO/CONSTANTES	RESULTADO	EXEMPLO
<code>Abs(x)</code>	Fornece Valor absoluto de x	<code>Abs(4.5) = 4.5;</code> <code>Abs(-4.5) = 4.5;</code>
<code>Ceiling(x)</code>	Arredonda x para cima	<code>Ceiling(4.1) = 5;</code> <code>Ceiling(-4.1) = -4;</code>
<code>Cos(x)</code>	Obtém o cosseno de x	<code>Cos(2.0) = -0.4161.....</code>
<code>Sin(x)</code>	Obtém o seno de x	<code>Sin(2) = 0.909297...</code>
<code>Tan(x)</code>	Obtém o valor da Tangente de x	<code>Tan(1.5) = 14.1014...</code>
<code>PI</code>	Obtém o valor de PI	<code>PI = 3.141516171819...</code>
<code>E</code>	Obtém o valor da constante E	<code>E = 2.7182818284590451</code>
<code>Exp(x)</code>	Obtém o exponencial (e elevado na x)	<code>Exp(5.0) = 54.59...</code>
<code>Floor(x)</code>	Arredonda o valor de x para baixo	<code>Floor(2.9) = 2;</code> <code>Floor(-2.9) = -3;</code>
<code>Log(x)</code>	Calcula o logaritmo de x na base natural e	<code>Log(3.0) = 1.098612...</code>
<code>Log10(x)</code>	Calcula o logaritmo de x na base 10	<code>Log10(3.0) = 0.47712...</code>
<code>Max(x,y)</code>	Obtém o maior valor entre dois números	<code>Max(2.46,2.56) = 2.56;</code> <code>Max(-2.46,-2.56) = -2.46;</code>
<code>Min(x,y)</code>	Obtém o menor valor entre dois números	<code>Min(1.92,1.89) = 1.89;</code> <code>Min(-1.92,-1.89) = -1.92;</code>
<code>Pow(x,y)</code>	Obtém o valor de x elevado na y	<code>Pow(2,4) = 16</code>
<code>Round(x,y)</code>	Arredonda x para y casas decimais	<code>Round(7.6758549,4) = 7.6759</code>
<code>Sqrt(x)</code>	Obtém a raiz quadrada de x	<code>Sqrt(169) = 13</code>

Comando de Entrada e Saída

```
namespace ExercConsole
```

```
{  
    class Program  
    {  
        static void Main(string[] args)  
        {  
            double areaCirculo = 0;  
            double RaioDoCirculo = 0;  
            Console.WriteLine(" Informe o raio do Círculo : ");  
            RaioDoCirculo = Convert.ToDouble(Console.ReadLine());  
            areaCirculo = Math.PI * Math.Pow(RaioDoCirculo, 2);  
            Console.WriteLine(" A área do círculo de raio " + RaioDoCirculo + " é : " + areaCirculo.ToString());  
        }  
    }  
}
```

Comando de **saída**

Comando de entrada

Concatenação

Comando opcional

Controle de fluxo

```
if (expressão) comando  
    [else comando]
```

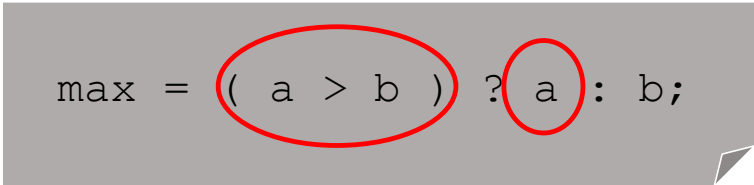
```
switch (expressão) {  
    case valor1: comandos;  
        break;  
    case valor2: comandos;  
        break;  
    ....  
    default: comando;  
}
```

```
if (resposta == OK) {  
    // comandos  
} else {  
    // comandos  
}  
  
int mes;  
switch (mes) {  
    case 1: Console.WriteLine("Jan");  
        break;  
    case 2: Console.WriteLine("Fev");  
        break; ....  
    default:  
        Console.WriteLine("Mês inválido");  
}
```

Operador Ternário

- Sintaxe:

exp1 ? exp2 : exp3



```
max = ( a > b ) ? a : b;
```

- Desenvolva um programa para identificar entre duas variáveis inteiras qual possui o maior valor armazenado.

```
class OperadorTernario {  
    static void Main(string[] args)  
    {  
        Console.WriteLine("...: Exemplo de Operador Ternário :...");  
        int a= 10, b=5;  
        int max = ( a > b ) ? a : b;  
        Console.WriteLine("O maior valor é: "+max);  
    }  
} // Qual foi o resultado?
```

Controle de fluxo - Repetição

while (*expressão*)

comando

for (*ini; cond; fim*)

comando

do {

comandos

} **while** (*expressão*);

break - sai do loop corrente

continue - volta para o teste do loop

Exemplos

```
int i = 3;
```

```
while (i <= 10) {
```

```
    i++;
```

```
    Console.WriteLine(i);
```

```
}
```

```
for (int i=1; i < 10; i++)
```

```
    Console.WriteLine(i);
```


Controle de fluxo - Repetição - foreach

- Declaração de vetor/array

```
int[] vet = new int[3];
```

```
vet[0] = 23;
```

```
vet[1] = 29;
```

```
vet[2] = 18;
```

- Apenas utilize quando o array tiver conteúdo

```
foreach (v in vet)
```

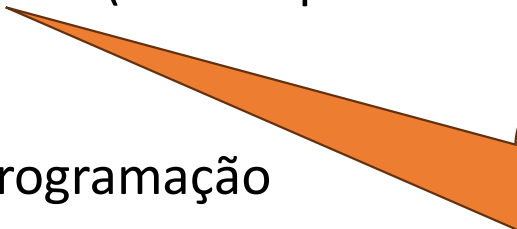
```
    Console.WriteLine(v);
```

Função

- É um bloco de código nomeado que executa uma tarefa específica, que processa um conjunto de valores fornecidos a ela e que pode retornar algum valor da operação requerida.

- Sintaxe:

```
tipo NomeDaFuncao(lista de parâmetros)
{
    //lógica de programação
}
```



Por convenção, as funções no C#, são nomeadas com letras maiúsculas/minúsculas.

Tipos de construção de funções

- Existem quatro tipos de construção de função:
 - Sem parâmetro e sem retorno
 - Com parâmetro e sem retorno
 - Sem parâmetro e com retorno
 - Com parâmetro e com retorno