

C# Essencial



P00 – Exercícios Resolvidos

Exercícios

1- Uma classe derivada pode interromper a herança *virtual* declarando um *override* como :
(☐) inherits (☐) extends (☐) private (☐) not inheritable (☒) sealed

2- Qual das palavras-chave a seguir é usada para alterar os dados e o comportamento de uma classe base substituindo um membro de uma classe base por um novo membro derivado ?

(☒) new (☐) base (☐) overloads (☐) override (☐) overridable

Qual das seguintes afirmações está correta?

A(☐) Os métodos estáticos podem ser um método virtual.

B(☐) Os métodos abstratos não podem ser um método virtual.

C(☐) É obrigatório substituir um método virtual.

D(☒) Ao substituir um método virtual, os nomes e as assinaturas do método de substituição devem ser os mesmos do método virtual que está sendo substituído.

E(☐) Podemos substituir métodos virtuais e não virtuais.

Exercícios

```
class A
{
    public int i;
    public void Exibir()
    {
        Console.WriteLine(i);
    }
}

class B: A
{
    public int j;
    public void Exibir()
    {
        Console.WriteLine(j);
    }
}
```

```
B b = new B();
b.i = 1;
b.j = 2;
b.Exibir();
Console.ReadKey();
```

Qual o resultado da execução do código acima considerando as classes A e B.
Onde B herda da classe A.

R: Quando o método **Exibir()** é chamado usando objetos da classe 'B', o método **Exibir()** da classe 'B' é chamado em vez do método da classe 'A'.
O resultado exibido será o valor : 2

Exercícios

```
class A
{
    public virtual void Exibir()
    {
        Console.WriteLine("A");
    }
}

class B: A
{
    public override void Exibir()
    {
        Console.WriteLine(" B ");
    }
}
```

```
A a = new A();
B b = new B();
A x;
x = a;
x.Exibir();
x = b;
x.Exibir();
```

Qual o resultado da execução do código acima considerando as classes A e B.
Onde B herda da classe A.

R: **O resultado exibido será : A e B**

Isso ocorre porque x inicialmente recebe uma instância de A e executa seu método **Exibir()** e a seguir x recebe uma instância de b e executa o seu método **Exibir()**.

Exercícios

Quais dos seguintes requisitos são necessários para o polimorfismo em tempo de execução?

- 1- O método base substituído deve ser *virtual* ou *abstract*
- 2- Tanto o método *override* quanto o método *virtual* devem ter o *mesmo modificador de nível de acesso*.
- 3- Uma declaração *override* pode alterar a acessibilidade do método virtual.
- 4- Uma propriedade abstrata herdada não pode ser substituída em uma classe derivada.
- 5- Um método *abstracto* é implicitamente um método *virtual*.

() 1 e 3 (x) 1, 2, e 5 () 2, 3 e 4 () somente 4

Exercício 1

Crie um programa que implemente uma interface **IVeiculo** com dois métodos, um para **Dirigir** do tipo *void* e outro para **Abastecer** do tipo *bool* que possui um parâmetro do tipo *int* com a quantidade de gasolina a abastecer.(defina isso como uma propriedade)

Em seguida, crie uma classe **Carro** com um construtor que receba um parâmetro com a quantidade inicial de gasolina do carro e implemente os métodos **Dirigir** e **Abastecer** do carro.

O método **Dirigir** deve exibir na tela "*Dirigindo o carro..*", se a gasolina for maior que 0, caso contrário deve exibir "*Sem gasolina...*", e o método **Abastecer** deve aumentar a gasolina do carro e retornar *true*.

Para testar, crie um objeto do tipo **Carro** com 0 de gasolina inicial e solicite ao usuário a informação de uma quantidade de gasolina para abastecer via teclado e por fim execute o método **Dirigir** do carro.

Exercício 1 - Resposta

```
public interface IVeiculo
{
    void Dirigir();
    bool Abastecer(int quantidade);
}

class Carro : IVeiculo
{
    public int Gasolina { get; set; }

    public Carro(int gasolina)
    {
        Gasolina = gasolina;
    }

    public void Dirigir()
    {
        if (Gasolina > 0)
        {
            Console.WriteLine("Dirigindo o carro...");
        }
        else
        {
            Console.WriteLine("Sem gasolina...");
        }
    }

    public bool Abastecer(int quantidade)
    {
        Gasolina += quantidade;
        return true;
    }
}
```

```
Carro carro = new Carro(0);

Console.WriteLine("Informe quantos litros de gasolina: ");

int gasolina = Convert.ToInt32(Console.ReadLine());

if (carro.Abastecer(gasolina))
{
    carro.Dirigir();
}
```

- Criamos a interface **IVeiculo** definindo o contrato para os métodos **Dirigir** e **Abastecer**
- A seguir criamos a classe **Carro** que implementa a interface e definimos a propriedade **Gasolina** do tipo *int* e os métodos **Dirigir()** e **Abastecer()**
- No construtora da classe **Carro** atribuímos a quantidade de gasolina na criação do objeto do tipo **Carro**
- Para testar criamos uma instância de *Carro* atribuindo o valor zero(0) e solicitamos a entrada de uma quantidade de gasolina e a seguir verificamos se o retorno do método **Abastecer** é igual a **true**, e, neste caso executamos o método **Dirigir()**

Exercicio 2

Crie um programa para gerenciar um álbum de fotos usando os conceitos da programação orientada a objetos.

Para começar, crie uma classe chamada **LivroFotos** com um atributo privado **numPaginas** do tipo int. Defina também um método público **GetNumeroPaginas** que retornará o número de páginas do álbum de fotos

O construtor padrão deverá criar um álbum com 16 páginas a classe deve possuir um construtor adicional, com o qual podemos especificar o número de páginas que queremos no álbum. Crie também uma classe **SuperLivroFotos** cujo construtor criará um álbum com 64 páginas.

Por fim, execute as seguintes ações:

- Criar um álbum de fotos padrão e exibir o número de páginas
- Criar um álbum de fotos com 24 páginas e exibir o número de páginas
- Criar um álbum de fotos grande e exibir o número de páginas

Exercicio 2 - Resposta

```
public class LivroFotos
{
    protected int numeroPaginas;

    public LivroFotos()
    {
        numeroPaginas = 16;
    }

    public LivroFotos(int numeroPaginas)
    {
        this.numeroPaginas = numeroPaginas;
    }

    public int GetNumeroPaginas()
    {
        return numeroPaginas;
    }
}

public class SuperLivroFotos : LivroFotos
{
    public SuperLivroFotos()
    {
        numeroPaginas = 64;
    }
}
```

```
LivroFotos meuAlbum1 = new LivroFotos();
Console.Write("Criando um Livro de fotos com ");
Console.Write(meuAlbum1.GetNumeroPaginas());
Console.Write(" páginas\n");
```

```
LivroFotos meuAlbum2 = new LivroFotos(24);
Console.Write("Criando um Livro de fotos com ");
Console.Write(meuAlbum2.GetNumeroPaginas());
Console.Write(" páginas\n");
```

```
SuperLivroFotos meuSuperAlbum1 = new SuperLivroFotos();
Console.Write("Criando um Livro de fotos com ");
Console.Write(meuSuperAlbum1.GetNumeroPaginas());
Console.Write(" páginas\n");
```

```
Console.ReadLine();
```

Exercicio 3

Crie um programa que solicite ao usuário três nomes de pessoas e os armazene em uma matriz de objetos do tipo **Pessoa**. Haverá duas pessoas do tipo **Aluno** e uma pessoa do tipo **Professor**.

Para isso, crie uma classe **Pessoa** que possua uma propriedade **Nome** do tipo string, um construtor que receba o nome como parâmetro e sobrescreva o método **ToString()**.

Então crie mais duas classes que herdam da classe **Pessoa**, elas serão chamadas de **Aluno e Professor**. A classe **Aluno** possui um método **Estudar** que escreve pelo console que o aluno está estudando. A classe **Professor** terá um método **Explicar** que grava no console que o professor está explicando.

Lembre-se de também criar dois construtores nas classes filhas que chamam o construtor pai da classe **Pessoa**.

Termine o programa lendo as pessoas (*o professor e os alunos*) e execute os métodos *Explicar e Estudar*.

Exercicio 3 - Resposta

```
public class Pessoa
{
    protected string Nome { get; set; }
    public Pessoa(string nome)
    {
        Nome = nome;
    }
    public override string ToString()
    {
        return "Olá! Meu nome é " + Nome;
    }
}
public class Professor : Pessoa
{
    public Professor(string nome) : base(nome)
    {
        Nome = nome;
    }

    public void Explicar()
    {
        Console.WriteLine($"{Nome} Explicando...");
    }
}
public class Aluno : Pessoa
{
    public Aluno(string nome) : base(nome)
    {
        Nome = nome;
    }
    public void Estudar()
    {
        Console.WriteLine($"{Nome} Estudando...");
    }
}
```

```
int total = 3;
Pessoa[] pessoas = new Pessoa[total];

for (int i = 0; i < total; i++)
{
    if (i == 0)
    {
        Console.WriteLine("Informe o nome do Professor");
        pessoas[i] = new Professor(Console.ReadLine());
    }
    else
    {
        Console.WriteLine("Informe o nome do Aluno");
        pessoas[i] = new Aluno(Console.ReadLine());
    }
}

for (int i = 0; i < total; i++)
{
    if (i == 0)
    {
        Console.Write("Professor... ");
        ((Professor)pessoas[i]).Explicar();
    }
    else
    {
        Console.Write("Aluno... ");
        ((Aluno)pessoas[i]).Estudar();
    }
}
```

Exercício 4

Crie um programa que solicite ao usuário os nomes de três pessoas e os armazene em uma matriz de objetos do tipo **Pessoa**.

Para fazer isso, primeiro crie uma classe Pessoa que tenha uma propriedade **Nome** do tipo string, um construtor que receba o nome como parâmetro e sobrescreva o método **ToString()**.

Finalize o programa lendo as pessoas e executando o método **ToString()** na tela.

Exercicio 4 - Resposta

```
public class Pessoa
{
    public string? Nome { get; set; }

    public override string ToString()
    {
        return "Ola! Meu nome é " + Nome;
    }
}
```

```
int total = 3;
Pessoa[] pessoas = new Pessoa[total];

Console.WriteLine("Informe o nome de 3 Pessoas");
for (int i = 0; i < total; i++)
{
    pessoas[i] = new Pessoa()
    {
        Nome = Console.ReadLine()
    };
}

Console.WriteLine("\nAcessando os dados...\n");

for (int i = 0; i < total; i++)
{
    Console.WriteLine(pessoas[i].ToString());
}

Console.ReadLine();
```

Exercício 5 - Resposta

```
Database db = new SQLServer();  
db.Configurar();  
db.Conectar();
```

```
Console.ReadKey();
```

Main

```
abstract class Database  
{  
    public virtual void Conectar()  
    {  
        Console.WriteLine("Conectando ao banco de dados...");  
    }  
    public abstract void Configurar();  
}
```

```
class SQLServer : Database  
{  
    public void Conectar()  
    {  
        Console.WriteLine("Conectando ao SQL Server...");  
    }  
    public override void Configurar()  
    {  
        Console.WriteLine("Configurando o banco de dados...");  
    }  
}
```

Para este exercício temos :

- 1- A classe abstrata Database
- 2- A classe concreta SQLServer que herda da classe Database

Quais alterações podemos fazer no código para que ele possa chamar o método Conectar da classe SQL Server sem alterar o código na classe Program (Método Main)

Resposta:

O método connect da classe SQLServer deve ser sobrescrito. Como o método Conectar da classe base abstrata é virtual, ele pode ser substituído na classe derivada.

Ex: public override void Conectar()

Exercício 6 - Resposta

```
B b = new B("Olá Mundo...");  
  
b.Exibir();  
  
Console.ReadKey();  
  
class A  
{  
    String texto;  
  
    public A(string s)  
    {  
        this.texto = s;  
    }  
    public void Exibir()  
    {  
        Console.Write(texto);  
    }  
}  
  
class B : A  
{  
    public B(string s) : base(s)  
    {  
        Console.Write(s);  
    }  
}
```

Dado o código ao lado explique qual vai ser o resultado da execução deste código.

Resposta:

Neste código, a variável da classe base *texto* está sendo inicializada por meio do construtor de classe derivada usando a palavra-chave *base*.

Assim o construtor da classe base vai atribuir o valor de *s* à variável *texto*

Portanto, na criação do objeto com o parâmetro *'Olá Mundo...'*, essa string está disponível tanto para a classe base quanto para o corpo do construtor da classe derivada.

O resultado será a exibição no console do texto :
'Olá Mundo...'

Exercício 7 - Resposta

```
INota1 p = new Exibir();
p.Classificacao();

Console.ReadKey();

interface INota1
{
    void Classificacao();
}

interface INota2
{
    void Classificacao();
}

class Exibir : INota1, INota2
{
    public void Classificacao()
    {
        Console.WriteLine("Classificacao INota1");
    }

    public void Classificacao()
    {
        Console.WriteLine("Classificacao INota1");
    }
}
```

Dado o código ao lado explique qual vai ser o resultado da sua execução e o que deve ser feito para corrigir o código caso isso seja necessário

Como a classe **Exibir** esta implementando as duas interfaces e como as interfaces possuem a *definição de um método* com o mesmo nome vai haver uma ambiguidade sobre qual dos métodos invocar na classe.

Se duas interfaces tiverem o mesmo contrato, a implementação da classe precisa de um identificador de interface e isso é feito realizando a implementação explícita de interface conforme mostrado a seguir:

```
class Exibir : INota1, INota2
{
    void INota1.Classificacao()
    {
        Console.WriteLine("Classificacao INota1");
    }

    void INota2.Classificacao()
    {
        Console.WriteLine("Classificacao INota2");
    }
}
```


Exercício 8

Na linguagem C# podemos criar um objeto da classe derivada a partir da sua classe base ? Explique e de um exemplo de código.

Resposta : Não.

Quando criamos um objeto de uma classe, sua referência é armazenada na memória *Stack* , o objeto é armazenado na memória *Heap* e o endereço do objeto é atribuído à referência da classe.

Dado o código : **Filha ofilha = new Filha();**

- **Filha** é a classe
- **ofilha** é o objeto da classe , seno uma referência à classe **Filha** criada na memória Stack
- **new Filha();** vai criar o objeto do tipo **Filha** na memória Heap e o endereço dele é atribuído à referência do objeto

Ao tentar criar um objeto da classe derivada a partir da classe base teremos uma chamada cíclica entre o construtor da classe Filha e a classe Pai causando um estouro da pilha (memória Stack). *(veja exemplo de código a seguir)*

Exercício 8 - Resposta

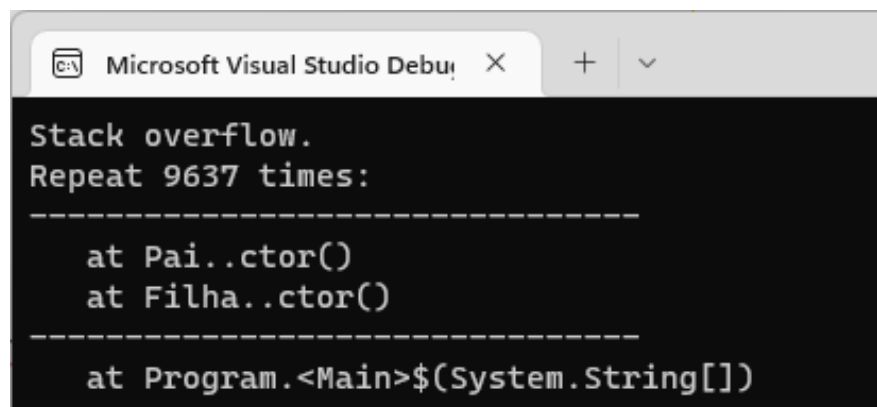
```
Filha ofilha = new Filha();

Console.ReadKey();

class Pai
{
    Filha ofilha = new Filha();

    public Pai()
    {
        Console.WriteLine("Construtor Pai");
    }
}

class Filha : Pai
{
    public Filha()
    {
        Console.WriteLine("Construtor Filha");
    }
}
```

The image shows a screenshot of the Microsoft Visual Studio Debug Console. The window has a title bar with the text 'Microsoft Visual Studio Debug' and standard window controls. The console output is as follows:
Stack overflow.
Repeat 9637 times:

at Pai..ctor()
at Filha..ctor()

at Program.<Main>\$(System.String[])
The text is displayed in a monospaced font on a dark background.

O processo será encerrado devido a um **StackOverflowException**.

Ao criar uma instância da classe Filha() o seu construtor será chamado, e, como classe Filha herda da classe Pai, a classe Filha vai executar a classe Pai onde temos o código para criar a instância da classe filha a partir da classe base.

Ao fazer isso será chamado o construtor da classe Filha novamente que vai retornar para a classe Pai e o processo vai se repetir até que não exista mais memória para ser alocada na Stack ocorrendo um erro de Stack Overflow.

Exercício 9

Apresente um cenário onde a utilização de interfaces seria a única solução possível. Explique e dê um exemplo de código.

Um cenário onde o uso de interfaces é a única solução possível seria um cenário onde temos que uma classe precisa herdar de mais de uma classe ou interface, caracterizando a herança múltipla de interface.

Na linguagem C# uma classe não pode herdar de mais de uma classe porém pode herdar de várias interfaces. Neste caso o uso de interfaces seria indispensável.

Veja exemplo de código a seguir.

Exercício 9 - Resposta

```
MastroCreditCard cartao = new MastroCreditCard();

cartao.ValidarCartao();
cartao.IniciarTransacao();
cartao.StatusTransacao();

Console.ReadKey();

interface ICartaoCredito
{
    void ValidarCartao();
}
interface ITransacao
{
    void IniciarTransacao();
}

class MastroCreditCard : ICartaoCredito, ITransacao
{
    public void ValidarCartao()
    {
        Console.WriteLine("Validando cartão...");
    }
    public void IniciarTransacao()
    {
        Console.WriteLine("Iniciando transação...");
    }
    public void StatusTransacao()
    {
        Console.WriteLine("Transação Completada");
    }
}
```

Temos aqui um possível cenário onde o uso de interfaces seria indispensável

A classe **MastroCreditCard** precisar implementar a *validação do cartão e a inicialização da transação* que devem ser definidos em classes ou interfaces diferentes, e com isso vai precisar herdar das duas classes ou interfaces.

Como não podemos herdar de duas classes a solução é criar duas interfaces (*ou uma classe e uma interface*) e fazer com que a classe **MastroCreditCard** herde das duas interfaces (*ou herde da classe e implemente a interface*).

Exercício 10

Usando os conceitos de polimorfismo escreva um programa para calcular a área do **Quadrado, Triângulo e do Circulo**

Fórmulas das áreas

- Quadrado -> **$A = \text{lado} * \text{lado}$**
- Triângulo -> **$A = (\text{base} * \text{altura})/2$**
- Circulo -> **$A = \text{Pi} * \text{raio} * \text{raio}$** ou $A = \pi . r^2$

Resposta :

```
public class Forma
{
    public virtual double Area()
    {
        return 0;
    }
}
```

```
public class Circulo : Forma
{
    private double Raio { get; set; }
    public Circulo(double raio)
    {
        Raio = raio;
    }
    public override double Area()
    {
        return (Math.PI * Raio * Raio);
    }
}
```

```
public class Quadrado : Forma
{
    private double Lado {get;set;}
    public Quadrado(double lado)
    {
        Lado = lado;
    }
    public override double Area()
    {
        return (Lado * Lado);
    }
}
```

Exercício 10 - Resposta

```
public class Triangulo : Forma
{
    private double Base { get; set; }
    private double Altura { get; set; }
    public Triangulo(double b, double h)
    {
        Base = b;
        Altura = h;
    }
    public override double Area()
    {
        return (0.5 * Base * Altura);
    }
}
```

```
using Resolucao9;

Forma c = new Circulo(3.5);
Console.WriteLine($"Área do círculo de raio 3.5. A = {c.Area()}");

Forma q = new Quadrado(5.5);
Console.WriteLine("Área do quadrado = {0}", q.Area());

Forma t = new Triangulo(3.0, 5.0);
Console.WriteLine("Área do triângulo = {0}", t.Area());

Console.ReadKey();

Console.WriteLine("\n ---- Outra solução ----- \n");
var formas = new List<Forma>()
{
    new Circulo(3.5),
    new Quadrado(5.5),
    new Triangulo(3.0,5.0)
};
foreach(var forma in formas)
{
    Console.WriteLine(forma.Area());
}
Console.ReadKey();
```

FIM