

C# Essencial



P00 – Exercícios

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

3- 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.

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 ?

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 () 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.

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 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*.

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.

Exercício 5

Para este exercício temos :

1- A classe abstrata Database

2- A classe concreta SQLServer que herda da classe Database

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

Quais alterações podemos fazer para que ele possa chamar o método Conectar da classe SQL Server sem alterar o código na classe Program (Método Main)

Exercício 6

```
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.

Exercício 7

```
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

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.

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.

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 $A = \pi . r^2$

FIM

Exercício 5 - Resposta

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

Main

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

```
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

O método Conectar() da classe abstrata Database está sendo chamado

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)

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()