Aula 05 – Herança e Polimorfismo em C#

1. Introdução

A programação orientada a objetos revolucionou o desenvolvimento de software ao introduzir conceitos que aproximam o código do mundo real, facilitando o entendimento e a manutenção dos programas. Entre os principais conceitos avançados desse paradigma estão a herança e o polimorfismo, que permitem maior reutilização, organização e flexibilidade do código.

Nesta aula, vamos estudar em detalhes o que são herança e polimorfismo, entender sua importância, analisar exemplos práticos e propor exercícios para fixação.

2. Herança

A herança é um dos pilares da programação orientada a objetos. Ela permite criar novas classes a partir de uma classe existente, chamada classe base ou superclasse. A nova classe, chamada classe derivada ou subclasse, herda todos os atributos e métodos da classe base, podendo acrescentar novos comportamentos ou modificar comportamentos herdados.

A herança proporciona uma maneira eficiente de reutilizar código, pois evita a repetição dos mesmos métodos e atributos em diversas classes. Além disso, facilita a manutenção e a extensão dos programas, tornando-os mais fáceis de evoluir e adaptar.

2.1. Exemplos de herança

Imagine um sistema de gestão escolar. Podemos criar uma classe base chamada `Pessoa` que armazena informações comuns, como nome e idade. A partir dela, podemos derivar as classes `Aluno` e `Professor`, que herdam os atributos e métodos de `Pessoa` e podem acrescentar características próprias, como matrícula para alunos ou disciplina para professores.

Outro exemplo: em um sistema de veículos, uma classe `Veiculo` pode servir de base para as classes `Carro`, `Moto` e `Caminhao`, que têm comportamentos e atributos semelhantes, mas também características específicas.

2.2. Sintaxe da herança em C#

Em C#, a herança é implementada utilizando o símbolo de dois pontos (`:`) após o nome da classe derivada, seguido do nome da classe base.

Exemplo:

```
class Veiculo {
   public string modelo;
   public int ano;

   public void ExibirDados() {
      Console.WriteLine($"Modelo: {modelo}, Ano: {ano}");
   }
}

class Carro : Veiculo {
   public int portas;

   public void ExibirPortas() {
      Console.WriteLine($"Portas: {portas}");
   }
}
```

No exemplo acima, `Carro` herda todos os membros públicos de `Veiculo`, mas pode acrescentar seus próprios membros.

2.3. Construtores em herança

Ao criar uma instância de uma subclasse, o construtor da superclasse também pode ser chamado. Isso é feito usando a palavra-chave `base` no construtor da subclasse.

```
class Veiculo
{
   public string modelo {get; set;}

   public Veiculo(string modelo)
   {
      this.modelo = modelo;
   }

   public void ExibirDados()
   {
```

```
        Console.WriteLine($"Modelo: {modelo}");
    }
}

class Carro : Veiculo
{
    public int portas {get; set;}

    public Carro(string modelo, int portas) : base(modelo)
    {
        this.portas = portas;
    }

    // Método adicional para exibir informações completas
    public void ExibirCarro()
    {
        ExibirDados(); // chama método da classe base
        Console.WriteLine($"Portas: {portas}");
    }
}

class Program
{
    static void Main()
    {
        Carro c1 = new Carro("Onix", 4);
        c1.ExibirCarro();

        // Para mostrar que herança funciona:
        Veiculo v1 = new Veiculo("Fusca");
        v1.ExibirDados();

        // Também é possível acessar método herdado por c1
        c1.ExibirDados(); // Exibe só o modelo novamente
    }
```

```
}
```

2.4. Vantagens da herança

- Redução de duplicidade de código.
- Facilidade de manutenção: alterações na superclasse refletem nas subclasses.
- Permite criar estruturas hierárquicas próximas do mundo real.

3. Polimorfismo

Polimorfismo é outro conceito fundamental da orientação a objetos. Significa "muitas formas" e refere-se à capacidade que objetos de diferentes classes derivadas têm de serem tratados como objetos de uma mesma superclasse, permitindo que métodos com o mesmo nome executem comportamentos diferentes dependendo do tipo do objeto.

3.1. Polimorfismo de sobrescrita

O polimorfismo mais comum em C# é o de sobrescrita de métodos. Uma subclasse pode sobrescrever um método herdado da superclasse usando as palavras-chave `virtual` (na classe base) e `override` (na classe derivada).

```
class Veiculo {
   public virtual void Mover() {
      Console.WriteLine("O veículo está se movendo.");
   }
}

class Carro : Veiculo {
   public override void Mover() {
      Console.WriteLine("O carro está rodando na estrada.");
   }
}
```

Ao usar referências da superclasse, o método sobrescrito da subclasse é chamado automaticamente:

```
Veiculo v1 = new Veiculo();
Veiculo v2 = new Carro();
v1.Mover(); // Saída: O veículo está se movendo.
v2.Mover(); // Saída: O carro está rodando na estrada.
```

3.2. Vantagens do polimorfismo

- Permite escrever código mais genérico e reutilizável.
- Facilita a manutenção e extensão do programa.
- Torna possível manipular coleções de objetos de diferentes tipos derivados.

4. Exemplo Completo

Veja como herança e polimorfismo trabalham juntos para tornar o código mais flexível:

```
class Pessoa {
   public string nome{get; set;}
   public virtual void Falar() {
      Console.WriteLine("Pessoa falando...");
   }
}

class Aluno : Pessoa {
   public override void Falar() {
      Console.WriteLine("Aluno respondendo presente!");
   }
}

class Professor : Pessoa {
   public override void Falar() {
      Console.WriteLine("Professor explicando a matéria.");
   }
}

// No Main():
Pessoa[] pessoas = new Pessoa[3];
pessoas[0] = new Pessoa { nome = "Carlos" };
pessoas[1] = new Aluno { nome = "Joana" };
pessoas[2] = new Professor { nome = "Marcelo" };
foreach (Pessoa p in pessoas) {
   Console.Write(p.nome + ": ");
   p.Falar();
}
```

Saída esperada:
Carlos: Pessoa falando...
Joana: Aluno respondendo presente!
Marcelo: Professor explicando a matéria.

5. Exercícios Propostos

1. Implemente uma hierarquia de classes `Animal`, `Cachorro` e `Gato`, com um método virtual `EmitirSom()` na base e sobrescrita nas subclasses. Mostre a saída ao chamar os métodos em um vetor de animais.

2. Crie uma classe base `Funcionario` e duas derivadas, `Gerente` e `Vendedor`, cada uma com método sobrescrito para calcular bonus (15% para Gerente e 10% para vendedor). Mostre como o polimorfismo pode ser utilizado nesse contexto.

3 – Criar um classe chamado conta. Esta classe terá as propriedades (atributos) nro_conta e saldo e os métodos depósito (para acrescentar valor a saldo), saque (para retirar valor de saldo, desde que ele não fique negativo) e versaldo (para mostrar o valor de saldo). Depois, crie uma classe chamado conta_especial, que herda a classe conta e acrescenta a propriedade limite, que é um saldo extra acrescido ao saldo. Na classe conta_especial o método saque deve ser reescrito para permitir que haja saque, mesmo que o saldo fique negativo, até o valor de limite.

4 –Instancie um objeto do tipo conta especial e atribua os valores iniciais a esta conta para testar os métodos escritos anteriormente no exercício 3, criando um menu que permita escolher as opções de saque, depósito ou sair.