Conteúdo: Hierarquia de Herança, Enumeradores e Verificação de Tipo em Execução com Cast e typeof

Introdução ao Problema

Vamos imaginar que você está desenvolvendo um sistema de gerenciamento de animais em um zoológico. Inicialmente, você decide criar uma única classe `Animal` e utilizar uma propriedade string para tipificar o objeto. A identificação do tipo de animal é feita verificando o valor dessa propriedade. No entanto, essa abordagem apresenta alguns problemas significativos.

Implementação Inicial (Incorreta)

Na implementação inicial, todas as propriedades e métodos estão centralizados na classe `Animal`. Cada instância de `Animal` utiliza uma string para indicar seu tipo.

```
public class Animal
{
    public string Tipo { get; set; }
    public string Nome { get; set; }

    public Animal(string tipo, string nome)
    {
        Tipo = tipo;
        Nome = nome;
    }

    public void MostrarDetalhes()
    {
        Console.WriteLine($"Nome: {Nome}, Tipo: {Tipo}");
    }
}
```

Problemas com a Implementação Inicial

1. **Fragilidade**: Verificar o tipo do objeto através de uma string é propenso a erros, especialmente se houver inconsistências de formatação (por exemplo, "Leao" vs. "leão").
2. **Manutenção Difícil**: Adicionar novos tipos de animais requer alterações em várias partes do código onde a verificação de tipo é realizada.
3. **Falta de Especialização**: Todas as características e comportamentos específicos dos diferentes tipos de animais estão ausentes, tornando a classe `Animal` genérica demais.

Implementação Melhorada (Ainda Incorreta)

Na tentativa de melhorar, você cria subclasses para cada tipo de animal, mantendo a propriedade `Tipo` na classe base `Animal`. A verificação do tipo ainda é feita através da string.

```
public class Animal
{
    public string Tipo { get; set; }
    public string Nome { get; set; }

    public Animal(string tipo, string nome)
```

```
    {
        Tipo = tipo;
        Nome = nome;
    }

    public void MostrarDetalhes()
    {
        Console.WriteLine($"Nome: {Nome}, Tipo: {Tipo}");
    }
}

public class Leao : Animal
{
    public Leao(string nome) : base("Leao", nome) { }
}

public class Tigre : Animal
{
    public Tigre(string nome) : base("Tigre", nome) { }
}

public class Elefante : Animal
{
    public Elefante(string nome) : base("Elefante", nome) { }
}
```

Problemas com a Implementação Melhorada

1. **Ainda Fragilidade**: A verificação do tipo através de strings continua sendo suscetível a erros de formatação.
2. **Duplicação de Código**: A lógica para definir o tipo é repetida em cada subclasse.
3. **Verificação Inconsistente**: O uso de strings para verificar o tipo do animal pode levar a problemas de manutenção e legibilidade.

Melhoria com Enumeradores

Para resolver os problemas mencionados, podemos utilizar enumeradores (enum). Enumeradores proporcionam uma maneira mais segura e clara de definir e verificar os tipos de animais.

Objetivo dos Enumeradores:

- **Segurança**: Eliminam a fragilidade associada ao uso de strings para representar tipos.
- **Clareza**: Tornam o código mais legível e fácil de manter.
- **Consistência**: Fornecem uma forma padronizada de definir e verificar tipos de objetos.

Definição do Enum:

```
public enum TipoAnimal
{
    Leao,
    Tigre,
```

```
    Elefante
}
```

Utilização do Enum na Classe Animal:

```
public class Animal
{
    public TipoAnimal Tipo { get; set; }
    public string Nome { get; set; }

    public Animal(TipoAnimal tipo, string nome)
    {
        Tipo = tipo;
        Nome = nome;
    }

    public void MostrarDetalhes()
    {
        Console.WriteLine($"Nome: {Nome}, Tipo: {Tipo}");
    }
}

public class Leao : Animal
{
    public Leao(string nome) : base(TipoAnimal.Leao, nome) { }
}

public class Tigre : Animal
{
    public Tigre(string nome) : base(TipoAnimal.Tigre, nome) { }
}

public class Elefante : Animal
{
    public Elefante(string nome) : base(TipoAnimal.Elefante, nome) { }
}
```

Verificação de Tipo em Tempo de Execução

Além do uso de enumeradores, a verificação de tipo pode ser realizada em tempo de execução utilizando operadores como `is`, `as`, bem como `typeof` e cast explícito.

Usando o Operador `is`:

```
public void IdentificarAnimal(Animal animal)
{
    if (animal is Leao)
    {
        Console.WriteLine($"{animal.Nome} é um Leão.");
    }
    else if (animal is Tigre)
    {
        Console.WriteLine($"{animal.Nome} é um Tigre.");
```

```
    }
    else if (animal is Elefante)
    {
        Console.WriteLine($"{animal.Nome} é um Elefante.");
    }
    else
    {
        Console.WriteLine($"{animal.Nome} é de um tipo desconhecido.");
    }
}
```

Usando o Operador `as` e Cast Explícito:

```
public void IdentificarAnimalComCast(Animal animal)
{
    Leao leao = animal as Leao;
    if (leao != null)
    {
        Console.WriteLine($"{animal.Nome} é um Leão.");
        return;
    }

    Tigre tigre = animal as Tigre;
    if (tigre != null)
    {
        Console.WriteLine($"{animal.Nome} é um Tigre.");
        return;
    }

    Elefante elefante = animal as Elefante;
    if (elefante != null)
    {
        Console.WriteLine($"{animal.Nome} é um Elefante.");
        return;
    }

    Console.WriteLine($"{animal.Nome} é de um tipo desconhecido.");
}
```

Usando o Operador `typeof`:

```
public void IdentificarAnimalComTypeOf(Animal animal)
{
    if (animal.GetType() == typeof(Leao))
    {
        Console.WriteLine($"{animal.Nome} é um Leão.");
    }
    else if (animal.GetType() == typeof(Tigre))
    {
        Console.WriteLine($"{animal.Nome} é um Tigre.");
    }
    else if (animal.GetType() == typeof(Elefante))
    {
```

```
        Console.WriteLine($"{animal.Nome} é um Elefante.");
    }
    else
    {
        Console.WriteLine($"{animal.Nome} é de um tipo desconhecido.");
    }
}
```