Perguntas Teóricas

1. Namespaces:

Qual é a finalidade de utilizar o using System; e outros namespaces, como System.Collections.Generic e System.Linq, no início do código? Todos eles são necessários neste caso específico? Justifique.

R: facilitar o acesso a classes e métodos que estão definidos nesses namespaces, evitando a necessidade de escrever o nome completo de cada classe.

2. Atributos protected:

No código, a variável Nome é declarada como protected. Qual é a diferença entre os modificadores de acesso protected e private em C#? Por que o modificador protected foi usado neste exemplo?

R: protected permite que a variável seja acessada por classes derivadas, enquanto private restringe o acesso apenas à própria classe. O modificador protected foi usado neste exemplo para permitir que subclasses, como Cachorro e Gato, acessem a variável Nome, promovendo a herança e a reutilização de código.

3. Construtor e a palavra-chave base:

O que acontece quando o construtor da classe Cachorro utiliza : base(nome)? Por que é necessário invocar o construtor da classe base Animal dessa forma?

R: Quando o construtor da classe Cachorro utiliza base(nome), ele está invocando o construtor da classe base Animal e passando o parâmetro nome. Isso é necessário para inicializar corretamente os atributos da classe base, garantindo que a classe Animal receba as informações necessárias para funcionar corretamente.

4. Polimorfismo e a palavra-chave virtual:

O método FazerSom() na classe Animal é marcado como virtual, e é sobrescrito nas classes Cachorro e Gato. Explique o conceito de polimorfismo e como ele está sendo aplicado neste código.

R: O conceito de polimorfismo permite que métodos com o mesmo nome se comportem de maneira diferente em classes diferentes. No código, o método FazerSom() na classe Animal é marcado como virtual e é sobrescrito nas classes Cachorro e Gato. Isso significa que, ao chamar FazerSom() em um objeto do tipo Animal, o método específico da classe derivada será executado, demonstrando o polimorfismo em ação.

5. Sobrescrita de Métodos:

Qual é a diferença entre override e new ao sobrescrever métodos em C#? Em quais situações o modificador override deve ser utilizado, como visto nos métodos FazerSom() e ExplicarHeranca()?

R: A diferença entre override e new ao sobrescrever métodos em C# é que override substitui um método virtual da classe base, enquanto new oculta um método da classe base. O modificador override deve ser utilizado quando se deseja alterar o comportamento de um método que já foi definido como virtual na classe base, como nos métodos FazerSom() e ExplicarHeranca().

6. Herança e Reutilização de Código:

Explique como o conceito de herança permite a reutilização de código neste exemplo. Como a classe Cachorro e a classe Gato aproveitam a classe base Animal?

R: permite a reutilização de código ao permitir que classes derivadas herdem características e comportamentos de uma classe base. No exemplo, as classes Cachorro e Gato aproveitam a classe base Animal para herdar atributos e métodos, evitando a duplicação de código e promovendo uma estrutura mais organizada.

7. Classe Base e Classe Derivada:

O que acontece se o método FazerSom() não for sobrescrito nas classes derivadas (Cachorro e Gato)? Como seria o comportamento se a chamada fosse feita a partir de um objeto dessas classes?

R: Se o método FazerSom() não for sobrescrito nas classes derivadas Cachorro e Gato, a chamada a partir de um objeto dessas classes resultará na execução do método FazerSom() da classe base Animal. Isso significa que o comportamento padrão da classe base será utilizado, o que pode não ser desejável se as subclasses precisarem de um comportamento específico.

8. Explicação sobre Herança:

O método ExplicarHeranca() descreve o conceito de herança em POO. Qual é a principal vantagem de sobrescrever esse método nas classes Cachorro e Gato, apesar de elas já herdarem o comportamento da classe Animal?

R: A principal vantagem de sobrescrever o método ExplicarHeranca() nas classes Cachorro e Gato, apesar de já herdarem o comportamento da classe Animal, é que isso permite personalizar a explicação do conceito de herança para cada tipo de animal, proporcionando uma descrição mais relevante e específica para cada classe.

9. Polimorfismo em Ação:

Suponha que você tenha a seguinte lista de objetos:

List<Animal> animais = new List<Animal> { new

Cachorro("Rex"), new Gato("Mimi") };

O que acontece se você iterar sobre essa lista e chamar o método FazerSom() de cada objeto? Como o polimorfismo está sendo aplicado nesse cenário?

R: Ao iterar sobre a lista List<Animal> animais = new List<Animal> { new Cachorro("Rex"), new Gato("Mimi") }; e chamar o método FazerSom() de cada objeto, o polimorfismo permite que o método específico de cada classe derivada seja executado. Assim, FazerSom() do Cachorro pode emitir um "au au", enquanto o Gato pode emitir um "miau", demonstrando como o polimorfismo funciona na prática.

10. Sobre a Extensibilidade do Código:

Como você poderia adicionar uma nova classe derivada (por exemplo, uma classe Pássaro) sem modificar muito o código original? Que métodos precisariam ser implementados na nova classe e por quê?

R: Para adicionar uma nova classe derivada, como uma classe Pássaro, sem modificar muito o código original, você precisaria implementar os métodos que são relevantes para a nova classe, como FazerSom(). Isso é necessário para garantir que a nova classe siga a mesma interface e comportamento esperado das classes existentes, mantendo a consistência do código.

11. Instanciação de Objetos:

Explique por que o seguinte código:

Animal animal = new Animal("Genérico");

é válido, mas a criação de objetos diretamente de uma classe abstrata (como se Animal fosse abstrata) resultaria em um erro.

R: O código Animal animal = new Animal("Genérico"); é válido porque a classe Animal não é abstrata, permitindo a criação de instâncias. Se Animal fosse uma classe abstrata, a tentativa de criar um objeto diretamente resultaria em um erro, pois classes abstratas são projetadas para serem subclasses e não podem ser instanciadas diretamente.