Curso Fundamentos da POO com C#

Aprenda a utilizar os conceitos da programação orientada a objetos na prática usando a linguagem C#

Herança e Polimorfismo - Exercícios : Respostas

Herança e Polimorfismo - Exercícios

1- Quais os principais tipos de relacionamentos podemos ter entre as classes e qual a principal vantagem eles oferecem ?

Os principais tipos básicos de relacionamentos definidos na programação orientada a objetos são:

- **Associação** : A associação descreve um vínculo que ocorre entre classes

- **Agregação** : É um tipo especial de associação onde as informações de um objeto *(chamado objeto-todo)* precisam ser complementados pelas informações contidas em um ou mais objetos de outra classe.

- **Composição** : Representa um vínculo forte entre duas classes; é um relacionamento **parte/todo**, onde o todo é responsável pelo ciclo de vida da parte. A existência do **objeto-Parte** NÃO faz sentido se o **objeto-Todo** não existir.

- **Herança** : É referida como um relacionamento **"é um" ou 'is a'**. Neste tipo de relacionamento, uma classe herda os membros de outra classe.

A principal vantagem oferecida pelos relacionamentos entre as classes *é a reutilização de código*

Herança e Polimorfismo - Exercícios

2- Descreva as regras usadas para definir os principais tipos de relacionamentos entre as classes.

Os principais tipos de relacionamentos entre classes estão expressos nas regras:

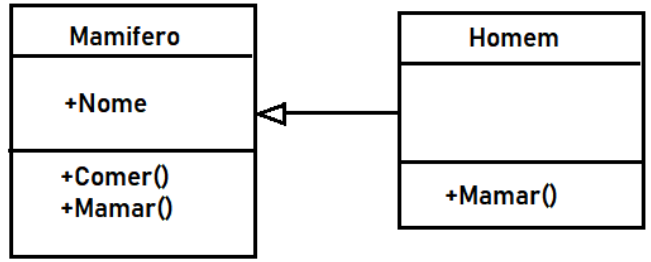
1- **"é um“ ou “is a”** que expressa um relacionamento de **herança**.
Exemplo:
Carro é um Veiculo
Homem é um Mamifero

2- **“tem um"** que expressa um relacionamento de **composição ou agregação.**
Exemplo:
Carro tem um motor (*composição*)
Time tem um ou mais jogadores *(agregação)*

Herança e Polimorfismo - Exercícios

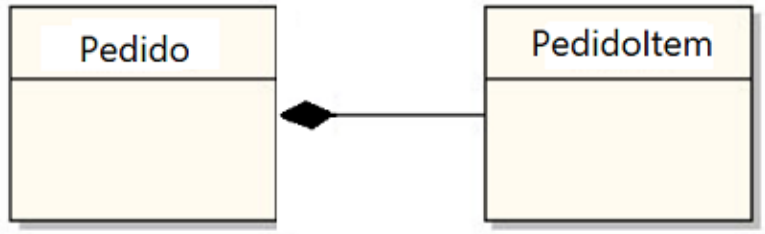
3- Qual a diferença entre herança e composição ?

Herança:



Estende atributos e métodos de uma classe;
Classe Pai, classe Base ou Superclasse é a classe que foi herdada;
Classe Filha ou SubClasse é a classe que herda da classe Pai;
Generalização - Obtém similaridades entre classes e define novas classes. As classes mais genéricas são as classes Pai;
Especialização - Identifica atributos e métodos não correspondentes entre classes distintas colocando-os na classe filha;
Usa a regra : É um;
Ex: Um homem é um mamífero

Composição:



Estende uma classe e delega o trabalho para o objeto desta classe;
Uma instância da classe existente é usada como componente da outra classe;
A partes que o compõem são essenciais e sem elas o objeto não existe;
Usa a regra : Tem um;
Ex: Um pedido tem um Item representando por PedidoItem

Herança e Polimorfismo - Exercícios

4- Quais as vantagens em usar composição sobre a herança ?

Usando herança:

1- Violamos um dos pilares da orientação a objetos: o encapsulamento, pois os detalhes da implementação da classe Pai são expostos nas classes Filhas;
2- Violamos um dos princípios básicos das boas práticas de programação : manter o acoplamento entre as classe fraco, visto que as classes filhas estão fortemente acopladas à classe Pai e alterar uma classe Pai pode afetar todas as classes Filhas;
3- As implementações herdadas da classe Pai pelas classes Filhas não pode ser alteradas em tempo de execução;

Usando composição :

1- Os objetos que foram instanciados e estão contidos na classe que os instanciou são acessados somente através de sua interface;
2- A composição pode ser definida dinamicamente em tempo de execução pela obtenção de referência de objetos a objetos de do mesmo tipo;
3- A composição apresenta uma menor dependência de implementações;
4- Na composição temos cada classe focada em apenas uma tarefa (princípio SRP);
5- Na composição temos um bom encapsulamento visto que os detalhes internos dos objetos instanciados não são visíveis;

Herança e Polimorfismo - Exercícios

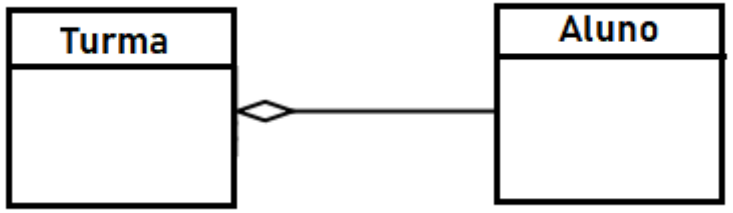
5- Qual a diferença entre composição e agregação ?

A diferença entre **composição e agregação** tem relação com a existência dos objetos e é apenas conceitual.

Tanto a composição como a agregação são associações que representam uma relação TODO-PARTE.

Na **composição** a parte não existe sem o todo.
Exemplo:
Nota Fiscal é composta por **itens da Nota**
Carro tem um **motor**
Se o objeto TODO não existir a parte não faz sentido

Na **agregação** a parte existe independente do todo.
Exemplo:
Turma e Aluno
Time e Jogador
Se o objeto TODO não existir a parte continua existindo.



Herança e Polimorfismo - Exercícios

6- Qual a diferença entre interface e classe abstrata ?

Classe Abstrata

- É uma classe **não pode ser instanciada**
- P*ode ser herdada* e geralmente serve como *classe base* para outras classes.
- Pode conter métodos abstratos e métodos comuns.
- Pode possuir construtores, propriedades, indexadores e eventos.
- Não pode ser estática (*static*). Uma classe abstrata não pode ser selada (*sealed*).
- Pode herdar de outra classe abstrata.

Interface

- Contém somente a assinatura dos métodos sem implementação;
- Todos os membros são públicos;
- Não pode ser instanciada;
- Uma classe pode implementar mais de uma interface;
- Não pode conter um construtor;
- Quando uma classe herda de uma interface ela tem que implementar todos os métodos da interface

Herança e Polimorfismo - Exercícios

Exemplo entre interface e classe abstrata

```
abstract class MaquinaDeLavar
{
  public MaquinaDeLavar()
  {
    // Codigo para iniciar o objeto.
  }

  abstract public void Lavar();
  abstract public void Enxaguar(int tamanhoCarga);
  abstract public long Secar(int velocidade);

  public void  Centrifugar()
  {
     Console.WriteLine("Centrifugando...");
  }
}
```

```
Interface IMaquinaLavar
{
    void Lavar();
    void Enxaguar(int tamanhoCarga);
    long Secar(int velocidade);
    void Centrifugar();
}
```

Herança e Polimorfismo - Exercícios

7- O que é o acoplamento entre classes ? Dê um exemplo e explique.

- Acoplamento é o nível de dependência/conhecimento que pode existir entre as classes;
- Uma classe com **acoplamento fraco** não é dependente de muitas classes para fazer o que ele tem que fazer;
- Uma classe com **acoplamento forte** depende de muitas outras classes para fazer o seu serviço;
- Uma classe com **acoplamento forte** é mais **difícil de manter**, de entender e de ser reusada;

```
public class A
{
        private B _b;

        public A()
        {
            _b = new B();
        }
}
```

Neste exemplo temos um **forte acoplamento** entre a **classe A e a classe B** pois quem controla a criação de uma instância da classe **B** é a classe **A**.

A classe A é responsável por obter uma referência a classe **B**.

Qualquer alteração que ocorra na classe **B** vai afetar diretamente a classe **A** pois esta possui uma referência a classe **B**.

Herança e Polimorfismo - Exercícios

8- O que é polimorfismo ?

O polimorfismo é considerado o terceiro pilar da programação orientada a objetos.

A palavra polimorfismo vêm do Grego e significa *'muitas formas'*.

Polimorfismo é um princípio a partir do qual as classes **derivadas** de uma única classe **base** são capazes de invocar os métodos que*, embora apresentem a mesma assinatura*, comportam-se de maneira diferente para cada uma das classes derivadas.

Polimorfismo é a habilidade de enviar uma mensagem a um objeto sem saber qual o tipo do objeto.

No C# podemos ter obter o polimorfismo usando herança e também a sobrecarga de métodos.

Herança e Polimorfismo - Exercícios

9- Em um projeto C# temos a interface e classes descritas abaixo:

```
class ConectarBancoDados
{
    public void Conectar()
    {
        Console.WriteLine("Conectar banco de dados");
    }
}
```

```
class Transacao : ConectarBancoDados, ITransacao
{
        public void Executa()
        {
            Console.WriteLine("Processando Transação...");
        }
}
```

```
interface ITransacao
{
        void Executa();
}
```

1 - Crie uma instância da classe **Transacao** e execute os métodos.

2 - Qual o resultado final esperado ?

3 - Você precisa que a classe **Transacao** utilize o método **Imprimir**(). Você tem duas opções :

a- Criar a interface **IRelatorio** e definir o contrato com este método ou
b- Criar uma classe **Relatorio** implementando o método.

Qual opção você usuária ? Justifique a sua resposta

Herança e Polimorfismo - Exercícios

Definição do código no método **Main**:

```
static void Main(string[] args)
{
    Transacao t = new Transacao();
    t.Conectar();
    t.Executa();
    Console.ReadLine();
}
```

Como não existe *herança múltipla* na linguagem C# a opção seria criar a interface **IRelatorio** e implementar esta interface na classe **Transacao**.

```
Interface IRelatorio
{
    void Imprimir();
}
```

```
class Transacao : ConectarBancoDados, ITransacao , Irelatorio
{
    ....
    public void Imprimir(string texto) { Console.WriteLine(texto);}
}
```

Herança e Polimorfismo - Exercícios

10- Crie uma classe **Animal** com a propriedade **Tipo**, um construtor onde podemos definir o *tipo de animal* e um método **Mover**() que iremos usar para simular o movimento de um animal.

A seguir crie uma classe **Peixe** e uma classe **Ave** que devem usar o construtor da classe **Animal** para definir o seu tipo e *devem implementar o método **Mover**() característico à sua espécie (peixe nada e ave voa);*

A classe **Animal** não poderá ser instanciada e vai servir como classe base para as demais classes.

Comente quais os recursos e princípios envolvidos na solução deste exercício .

Herança e Polimorfismo - Exercícios

```
abstract class Animal
{
      public string Tipo { get; set; }
      public Animal(string tipoAnimal)
      {
         Tipo = tipoAnimal;
      }

      public abstract void Mover();
 }
```

```
class Peixe : Animal
{
      public Peixe(string tipoAnimal) : base(tipoAnimal)
      { }

      public override void Mover()
      {
         Console.WriteLine("Nadando 15 metros");
      }
}
```

```
class Ave : Animal
 {
      public Ave(string tipoAnimal) : base(tipoAnimal)
      {}
      public override void Mover()
      {
         Console.WriteLine("Voando 20 metros");
      }
 }
```

```
static void Main(string[] args)
{
         Animal animal1 = new Ave("Pato");
         animal1.Mover();
         Animal animal2 = new Peixe("Tubarão");
         animal2.Mover();
         Console.ReadLine();
 }
```