5 – Abstração em POO com C#

A abstração é um dos recursos que tornam o paradigma de programação orientada a objetos tão útil e interessante de se trabalhar em projetos consideravelmente complexos. Classes e métodos abstratos, em POO, servem para moldar uma estrutura comum para entidades (classes) de um programa que compartilham algumas características, mas que diferem em outras.

Suponha que seja necessário criar várias classes que possuem algumas características e comportamentos em comum, mas que ao mesmo tempo possuem suas particularidades, o que impede de se trabalhar com uma única classe. Nessas situações, podemos criar uma classe superior que sirva como molde para as demais, definindo sua estrutura base e que tenha, ou não, métodos já implementados por padrão.

Para os métodos cuja execução seja comum a todas as classes filhas, pode-se implementá-los ainda na classe abstrata, e todas as classes filhas herdarão o mesmo comportamento. Já aqueles métodos que serão implementados de formas distintas nas classes que os herdarão, podem ser declarados apenas como uma assinatura, sem corpo, para que cada classe filha os implemente como for mais adequado.

Os métodos que devem ser implementados ainda na classe mãe não devem ser marcados como abstratos, pois essa definição é apenas para os métodos que serão implementados pelas classes filhas. Também é importante destacar que classes abstratas não podem ser instanciadas, elas servem apenas como molde para herança por classes filhas. E todas as classes abstratas devem ser, obrigatoriamente, escritas na classe filha.

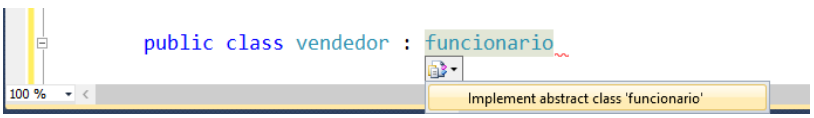
Em C#, a declaração de classes e métodos abstratos é feita com a inserção da palavra reservada *abstract.* Veja o exemplo da classe abstrata funcionário:

```
public abstract class funcionario
        {
           //estes atributos valerão para todos
            public string matricula { get; set; }
            public string nome { get; set; }

            //este método vale para todos
            public void mostra_funcionario()
            {
               Console.WriteLine("Matrícula: {0} - Nome: {1}",this.matricula,this.nome);
            }

            //este método será abstrato e deve ser
            //declarado nas classes filhas
            public abstract double comissao();
        }
```

Note que, como foi explicado anteriormente, o método abstrato **comissao()** não possui corpo implementado, apenas assinatura. Ele deverá ser escrito na classe filha. Se o programador não escrever, o compilador dará erro e não irá compilar o código.



Note que o próprio programa avisará, como um traço azul abaixo do nome da classe abstrata, que existem métodos a serem implementados. Se clicar, ele já montará a estrutura necessária básica para a implementação.

```
public override double comissao()
            {
                throw new NotImplementedException();
            }
```

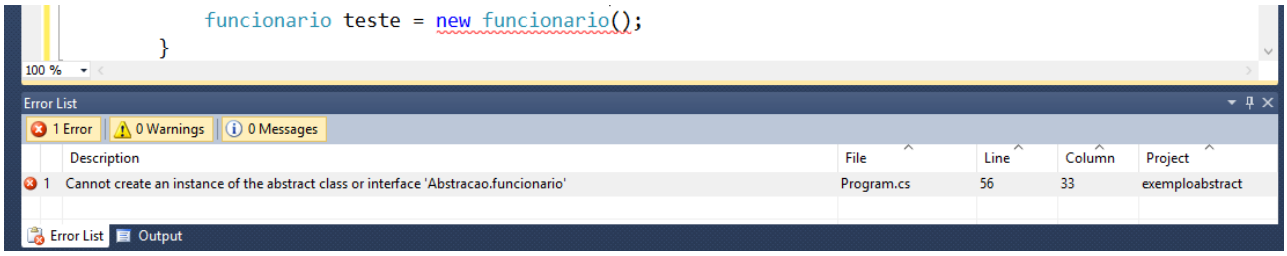
Basta agora ao programador escrever o código desejado, por exemplo:

```
public override double comissao()
            {
                return Convert.ToDouble("10,5");
            }
```

A classe filha, ao herdar da classe abastrata, só irá implementar os métodos que foram definidos como abstract, pois os métodos que já foram implementados na classe mãe já têm seu comportamento definido e este será herdado pela classe filha sem que seja necessário nenhuma codificação extra.

Ao criar uma instância da classe filha, podemos acessar os dois métodos que foram definidos na classe abstrata. E como esperado, notamos que o método que foi implementado na classe abstrata funciona como esperado na classe filha, e da mesma forma ocorrerá para quaisquer outras classes que herdarem dessa classe mãe. Já o método abstrato, só terá seu retorno definido nas classes filhas, representando um comportamento que é individual de cada classe.

Na hora de instanciar as classes, apenas as classes filhas poderão ser instanciadas. A classe funcionário, neste caso, por ser abstrata, não será instanciada. O compilador dará erro se tentar fazer isso, veja na imagem:



Criaremos as demais classes que não serão abstratas:

```
class Program
    {

        public abstract class funcionario
        {
            public string nome { get; set; }
            public string matricula { get; set; }

            public void mostra_funcionario()
            {
                Console.WriteLine("Matrícula: {0} - Nome: {1}", this.matricula, this.nome);
            }
```

```
        public abstract double comissao();

    }

    public class vendedor : funcionario
    {
        public override double comissao()
        {
            return 10.5;
        }
    }

    public class atendente : funcionario
    {
        public override double comissao()
        {
            return 5;
        }
    }

    public class entregador : funcionario
    {
        public double gorjeta { get; set; }

        public void valorgorjeta()
        {
            Console.Write("Qual o valor da gorjeta: ");
            this.gorjeta = Convert.ToDouble(Console.ReadLine());
        }

        public override double comissao()
        {

            return 0;
        }
    }
```

Vamos implementar nosso programa principal Main() para execução do exemplo:

```
static void Main()
{
   double vendas = 1000;
   vendedor paulo = new vendedor();
   atendente maria = new atendente();
   entregador igor = new entregador();
   paulo.nome = "Paulo Andrade";
   paulo.matricula = "000001";
   maria.nome = "Maria Aparecida da Silva";
   maria.matricula = "000002";
   igor.nome = "Igor José";
   igor.matricula = "000003";
   paulo.mostra_funcionario();
   Console.WriteLine("Vendas: R$ {0} - Valor da Comissão: R$ {1}", vendas, (vendas *
paulo.comissao() / 100));
   maria.mostra_funcionario();
   Console.WriteLine("Vendas: R$ {0} - Valor da Comissão: R$ {1}", vendas, (vendas *
maria.comissao() / 100));
   igor.mostra_funcionario();
```

```
    Console.WriteLine("Vendas: R$ {0} - Valor da Comissão: R$ {1}", vendas, (vendas *
igor.comissao() / 100));
    igor.valorgorjeta();
    Console.WriteLine("Igor ganhou R$ {0} de gorjeta.", igor.gorjeta.ToString());
    Console.ReadKey();
}
```

EXERCÍCIO

Crie uma classe abstrata chamada FormaGeometrica, que possui como métodos abstratos: area e perimetro.

Implemente classes filhas chamadas TrianguloRetangulo, Circulo e Retangulo, que herdam a classe abstrata FormaGeometrica.

Triângulo possui os atributos base e altura e sabemos, pelo exercício da aula anterior, que é possível ser calculado um atributo private chamado hipotenusa, que será utilizado para o cálculo do perímetro. (área=base*altura/2) (perímetro = base+altura+hipotenusa)

Círculo tem um atributo chamado Raio, que será usado para calcular a área do círculo (PI * R^2) e seu perímetro (2 * PI * R).

Retangulo possui um atributo chamado base e outro chamado altura. (área=base*altura) (perímetro=2 * base + 2 * altura).

Faça um programa que instancie as classes TrianguloRetangulo, Circulo e Retangulo, leia os atributos necessários de cada figura, calcule e mostre a área e o perímetro de cada figura.