16 Conceitos de POO(Programação Orientada a Objeto)

16 conceitos comparativos de POO(Programação Orientada por Objetos)

Classes

Atributos

Métodos

Objeto

Abstração

Estado

Visibilidade

Escopo e Variável

Encapsulamento

Passagem de Parâmetros

Herança

Associação

Dependência

Agregação

Interface

Composições

16 conceitos comparativos de POO(Programação Orientada por Objetos)

1 Conceito - Classes

Uma Classe é uma descrição de um conjunto de objetos que compartilham os mesmos atributos, operações, relacionamentos e semântica.

Características:

Nome – que a diferencia das outras classes

Atributos – propriedades que descrevem um intervalo de Valores que as instâncias da classe podem apresentar.
Abstraem os tipos de dados ou estados que os objetos de uma classe Podem abranger.

16 conceitos comparativos de POO(Programação Orientada por Objetos)

1 Conceito - Classes

Uma Classe é uma descrição de um conjunto de objetos que compartilham os mesmos atributos, operações, relacionamentos e semântica.

Características:

Operações(métodos) – implementam serviços que podem Ser solicitados por algum objeto da classe para modificar o Comportamento. Abstraem algo que pode ser feito com um objeto. Algumas vezes, a chamada a uma operação de um objeto altera os Atributos ou o estado do mesmo.

Definição: Representação de um conjunto de objetos do mundo real.

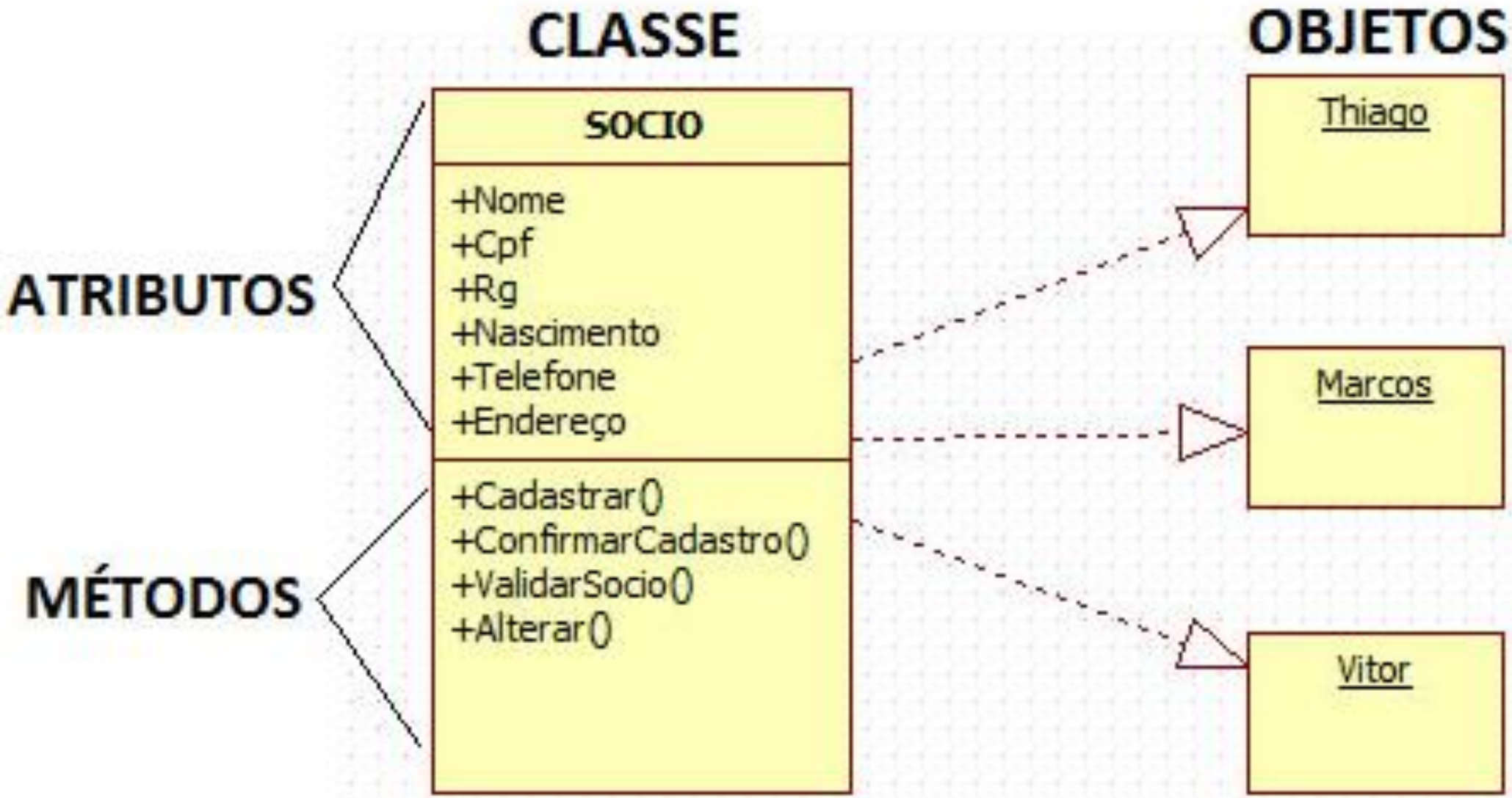


Tabela 1. Classe e Objeto.

Tipos de Classes

Na Programação Orientada a Objetos contamos com diversos tipos de classes como:

- Públicas
- Privadas
- Protegidas,
- Estáticas
- Abstrata
- Seladas
- Herdadas entre outras, sendo que todas possuem recursos que se encaixam ao decorrer da análise do programador.

Pública (Public)

As classes públicas, assim como os atributos e métodos, permitem que qualquer pessoa instancie objetos.
O nome da classe é precedido pelo nível de acesso (public) seguindo pela palavra chave “class”.

```
public class Socio
{
    // Códigos da Classe
}
```

Privada (Private)

Seguindo a mesma lógica dos atributos e métodos, as classes privadas não permitem acesso externo. A sintaxe de criação a seguir.

```
private class Socio
{
    // Códigos da Classe
}
Ou
class Socio
{
    // Códigos da Classe
}
```

Existem duas maneiras para declaração de classes privadas, a primeira utilizando a palavra “private” e a segunda sem nenhuma referência.

Com o exemplo citado fica mais fácil a visualização

Instanciáveis

Este tipo de classe é o mais utilizado, ou seja, toda vez que precisarmos criar um objeto, é necessário instanciá-lo, podendo assim criar vários objetos desta mesma classe.

```
Public class Socio
{
   //atributos estáticos

   private string nome;

   public string Nome
   {
        get { return nome;}
        set { nome = value;}
   }
}
```

Exemplo de utilização:

```
Socio soc = new Socio();
soc.Nome = "Thiago Montebugnoli";
```

Estáticas

Quando utilizamos estes tipos de classes, deveremos por obrigação, possuir todos os atributos como estáticos.
A principal característica destas classes é não permitir realizar a instância de um objeto, ou seja, quando for utilizá-la basta fazer referência aos membros para poder trabalhar com os mesmos

```
Public static class Socio
{
   //atributos estáticos

   private static string nome;

   public static string Nome
   {
        get { return Socio.nome;}
        set { Socio.nome = value;}
   }
}
```

Exemplo de utilização:

```
Socio.nome = "Thiago Montebugnoli"
```

Na sua utilização basta atribuir o valor desejado. Como foi dito anteriormente, a instância já é criada automaticamente quando executamos o programa, podendo ser utilizada em todos os pontos do software. A principal vantagem no uso desta classe é a automatização na criação da instância.

16 conceitos comparativos de POO(Programação Orientada por Objetos)

2 Conceito – Atributos

Atributos de uma classe também conhecido como propriedades, descrevem um intervalo de valores que as instâncias da classe podem apresentar.

Um atributo é uma variável que pertence a um objeto. Os dados de um objeto são armazenados nos seus atributos.

Informações sobre o objeto. Dados que posso armazenar.

16 conceitos comparativos de POO(Programação Orientada por Objetos)

3 Conceito – Métodos

Os métodos são procedimentos ou funções que realizam as ações próprias do objeto.

Assim, os métodos são as ações que o objeto pode realizar.

Tudo o que o objeto faz é através de seus métodos, pois é através dos seus métodos que um objeto se manifesta, através deles que o objeto interage com os outros objetos.

Sendo mais conhecidos como: Método Construtor, Métodos Get e Set, Métodos do usuário e Método sobrescrito

```
//Declaração do método Cadastrar(), do tipo "void" e "public" public void Cadastrar()
{
      MessageBox.Show("Cadastrando sócio... ");
}
```

São as funções e Procedimentos dentro da classe.

Realizam operações sobre as informações contidas nos atributos de uma classe.

Os métodos podem ser entendidos como mensagens trocadas entre diferentes objetos.

Assim como os atributos, os métodos também podem ser do tipo público ou privado.

Métodos do tipo “void” podem ser comparados aos procedimentos, ou seja, não retornam nenhum valor.
Para retornar algum dado vemos o exemplo a seguir:

```
//Declaração do método ConfirmarCadastro(), do tipo "bool" e "public" public bool
ConfirmarCadastro(int tipo)
{
      if (tipo == 1)
          return true;
      else
          return true;
}
```

O método criado anteriormente está a retornar um valor do tipo Booleano utilizando a cláusula “return”, que significa retorno.

Construtores e Destrutores

Toda classe criada deverá possuir dois métodos:

- **o Construtor (Construct) que é chamado no momento quando instanciamos a Classe e**
- **o Destrutor (Destruct) quando liberamos o objeto criado por esta classe da memória**

No C# temos o denominado “Garbage Collector”, trocando em miúdos seria um “Coletor de Lixos”.

Ele é responsável pela destruição de todo objeto que não é mais utilizado, sendo um recurso capaz de oferecer uma solução automatizada ao gerenciamento de memória.

```
//Construtor
public Socio()
{
    MessageBox.Show("Objeto Criado com sucesso!");
}
//Destrutor
~Socio()da classe
{
}
```

O método Construtor será invocado quando instanciamos a Classe “Socio” e para fins de aprendizagem utilizei a sintaxe do Destrutor, que normalmente não precisamos de nos preocupar, pois contamos com o recurso “Garbage Collector” citado anteriormente.

16 conceitos comparativos de POO(Programação Orientada por Objetos)

4 Conceito – Objeto

Objetos são instâncias de classes, que determinam qual informação um objeto contém e como ele pode manipulá-la.

É uma entidade capaz de reter um estado (informação) e que oferece uma série de informações (comportamento) ou para examinar ou para afetar este estado.

É através deles que praticamente todo o processamento ocorre em sistemas implementados com linguagens de programação orientada a objetos.

Definição: Elemento encontrado no contexto do sistema a ser desenvolvido

16 conceitos comparativos de POO(Programação Orientada por Objetos)

5 Conceito – Estado

O estado de um objeto é representado pelas variáveis definidas na própria classe.

- Sendo eles:
  Concreto: que existe fisicamente.

- **Abstrato**: é um conceito

16 conceitos comparativos de POO(Programação Orientada por Objetos)

6 Conceito – Escopo e variável

O código que “vê” uma determinada variável é chamada o escopo da variável. Podendo definir uma variável em global ou local.

Assim, o escopo de uma variável global é a classe inteira, e o escopo de uma variável local é o método, ou bloco contido dentro do método, ao qual ela pertence.

As variáveis declaradas fora de qualquer método (usualmente no cabeçalho da classe) e são acessíveis por qualquer método da classe são chamadas globais.

16 conceitos comparativos de POO(Programação Orientada por Objetos)

6 Conceito – Escopo e variável

Muitas vezes, variáveis auxiliares são declaradas dentro de um determinado método, ou até dentro de um bloco menor.

Tais variáveis são chamadas locais. Elas existem somente durante a execução daquele método ou bloco.

16 conceitos comparativos de POO(Programação Orientada por Objetos)

7 Conceito – Passagem de parâmetros

As passagens de parâmetros ocorrem na troca de mensagens através da chamada aos métodos de um objeto por outro objeto.

16 conceitos comparativos de POO(Programação Orientada por Objetos)

8 Conceito – Abstração

Ocorre quando se consegue dar uma identidade e a identidade deve ser única dentro do sistema colando em si as suas propriedades, ex: “Genero” e “Idade”

16 conceitos comparativos de POO(Programação Orientada por Objetos)

9 Conceito – Visibilidade

São distribuídas em três tipos, **private** (privado), **public** (publico) e **protect** (protegido)

Privado: uma função local e um único bloco de código

Público: visível para tudo uma função a ser chamada a qualquer momento

Protegido: esse caso restringe o parâmetro fora da classe, mas ainda acessível as suas subclasses (herança)

16 conceitos comparativos de POO(Programação Orientada por Objetos)

10 Conceito – Encapsulamento

Trata-se do fato de esconder as propriedades, criando uma espécie de caixa preta.

Sempre com os métodos privados ligadas a métodos especiais chamados *getteres* e *setteres*, que irão retornar e “setar” o valor da propriedade, respetivamente.

O encapsulamento evita o acesso direto à propriedade do objeto, adicionando uam outra camada de segurança à aplicação.

```
//Encapsulamento do atributo Público Nome
public string Nome
{
        get
        {
                return nome;
        }
        set
        {
                nome = value;
        }
}
```

Deixamos o atributo “Nome” dinâmico, pois poderemos retornar e atribuir valores dinamicamente com os operadores “Get” e “Set” respetivamente, permitindo criar regras e lógicas para acesso a dados e público permitindo uma visibilidade e acessibilidade externa à Classe.

Encapsulamento

O encapsulamento é o processo de ocultar ou esconder os membros de uma classe do acesso exterior usando modificadores de acesso. O encapsulamento também é chamado de ocultação de informação ou *information hiding*.

O encapsulamento fornece uma maneira de preservar a integridade do estado dos dados. Ao invés de definir campos públicos devemos definir campos de dados privados.

A classe bem encapsulada deve ocultar seus dados e os detalhes de implementação do mundo exterior. Isso é denominado programação *caixa preta*. Usando o encapsulamento, a implementação do método pode ser alterada pelo autor da classe sem quebrar qualquer código existente fazendo uso dela.

Encapsulamento

Um modificador de acesso define o escopo e a visibilidade de um membro da classe. A linguagem C# suporta os seguintes modificadores de acesso:

- **Public**
- **Private**
- **Protected**
- **Internal**
- **Protected Internal**

O modificador de acesso Public

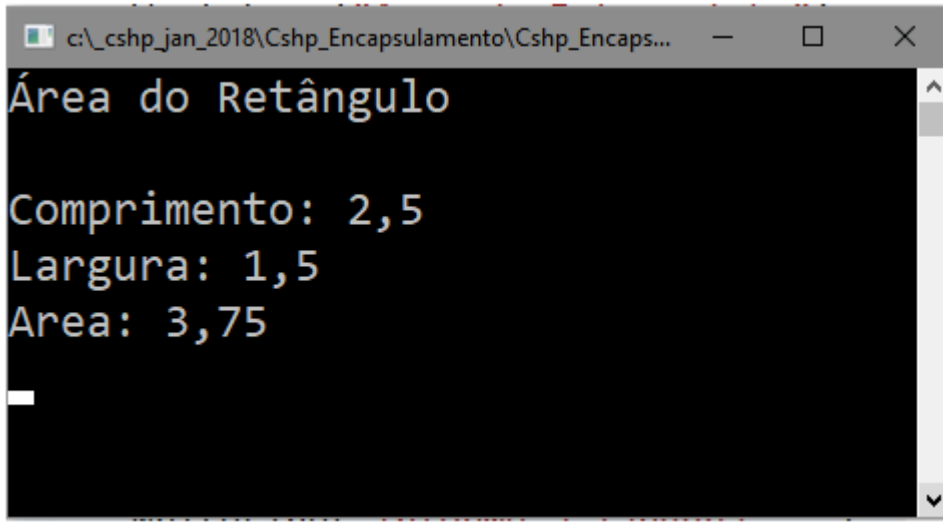
O modificador de acesso **Public** permite que uma classe exponha suas variáveis de membros e funções de membros a outras funções e objetos.
Qualquer membro público pode ser acedido de fora da classe.

```
using static System.Console;

namespace Cshp_Encapsulamento
{
    class Retangulo
    {
        //variáveis membros
        public double comprimento;
        public double largura;

        public double GetArea()
        {
            return comprimento * largura;
        }
        public void Exibir()
        {
            WriteLine("Área do Retângulo\n");
            WriteLine($"Comprimento: {comprimento}");
            WriteLine($"Largura: {largura}");
            WriteLine($"Area: {GetArea()}");
        }
    }
```

```
    class Program
    {
        static void Main(string[] args)
        {
            var r = new Retangulo();
            r.comprimento = 2.5;
            r.largura = 1.5;
            r.Exibir();
            ReadLine();
        }
    }
}
```



O que foi feito

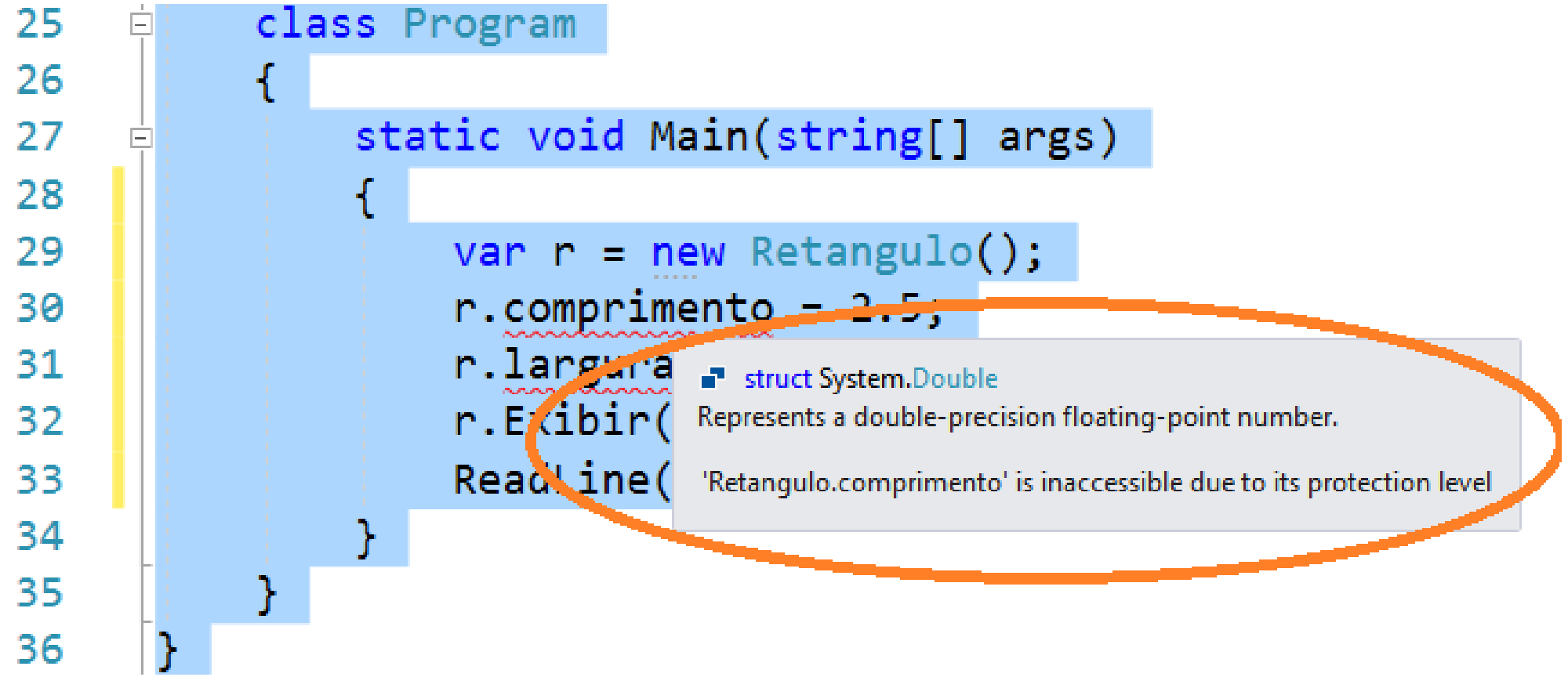
Neste código definimos a classe **Retangulo** contendo dois campos : **comprimento e largura** que foram declarados como públicos.

Dessa forma eles podem ser acessados diretamente a partir do método **Main**() usando uma instância **r** da classe **Retangulo**.

Os métodos **Exibir() e GetArea()** também podem acessar esses campos diretamente pois estão na mesma classe.

Aqui o código não esta encapsulado e pode ser alterado por qualquer programa exterior.

Agora vamos usar o código anterior alterando o escopo dos campos **comprimento e largura** para privados. Fazemos isso usando o modificador de acesso **private**.
Ao fazer isso você verá que os campos não estão mais acessíveis no método **Main**() da classe **Program**, pois o modificador de acesso **private** permite apenas o acesso local aos campos.



Os campos agora somente podem ser acessados pelos métodos **GetArea() e Exibir()** da classe **Retangulo**. Esses métodos são públicos e podem ser acessados no método **Main**().

Como então aceder aos valores de comprimento e largura ?

Podemos declarar um método público chamado **InformarValores()** e permitir que o valores sejam informados e atribuídos a esses campos:

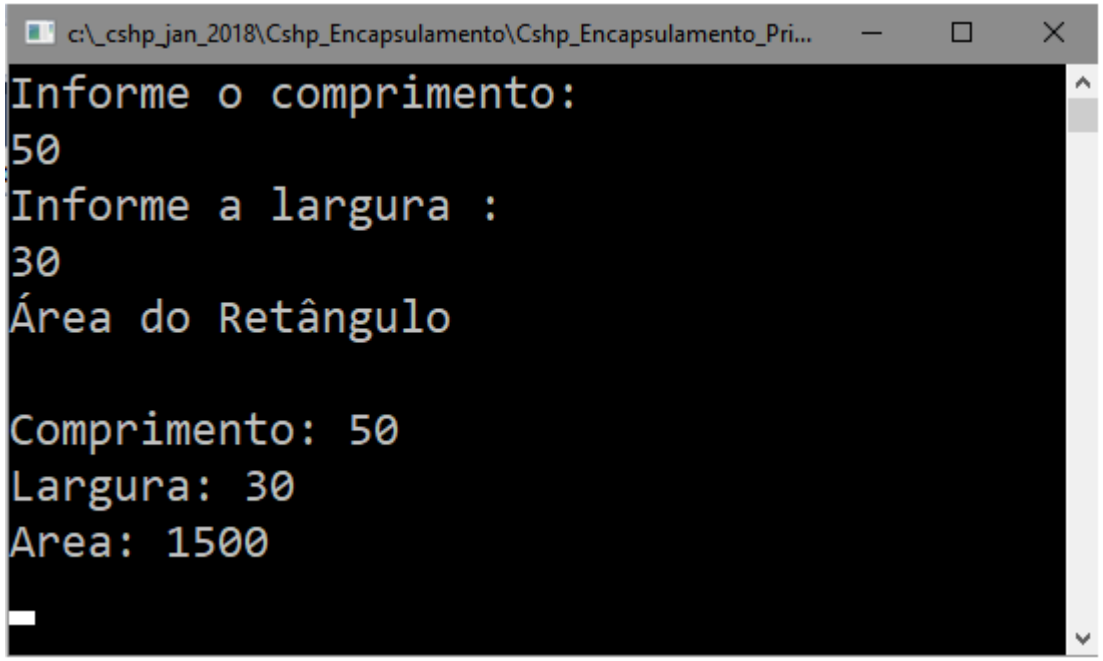
```
using static System.Console;

namespace Cshp_Encapsulamento
{
    class Retangulo
    {
        //variáveis membros
        private double comprimento;
        private double largura;

        public double GetArea()
        {
            return comprimento * largura;
        }
        public void Exibir()
        {
            WriteLine("Área do Retângulo\n"
            WriteLine($"Comprimento: {com
            WriteLine($"Largura: {largura}");
            WriteLine($"Area: {GetArea()}");
        }
```

```
        public void InformarValores()
        {
            WriteLine("Informe o comprimento: ");
            comprimento = Convert.ToDouble(Console.ReadLine());
            WriteLine("Informe a largura : ");
            largura = Convert.ToDouble(Console.ReadLine());
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            var r = new Retangulo();
            r.InformarValores();
            r.Exibir();
            ReadLine();
        }
    }
}
```

Executando o projeto iremos obter o seguinte resultado:



Essa implementação é mais robusta pois oculta o valor dos campos **largura e comprimento** permitindo que eles sejam acedidos somente pelo método **InformaValores**().

Criando propriedades Públicas

Podemos melhorar o código definindo duas propriedades públicas **Comprimento e Largura** que permitem acessar o valor dos campos **comprimento e largura.**

Na definição das propriedades podemos incluir uma lógica não permitindo que valores menores que zero sejam incluídos, se isso ocorrer lançamos uma exceção.

Removemos também o método **Exibir()** da classe **Retangulo** que estava com a responsabilidade de exibir o resultado e usava para isso recursos da interface do usuário.

1) Como ficou o código

```
using System;
using static System.Console;

namespace Cshp_Encapsulamento_Private
{

    class Retangulo
    {
        private double comprimento;
        private double largura;

        public double Comprimento
        {
            get { return comprimento; }
            set
            {
                if (value < 0)
                {
                    throw new ArgumentException("O valor do comprimento não pode ser menor que zero");
                }
                else
                {
                    comprimento = value;
                }
            }
        }
```

2) Como ficou o código

```
    public double Largura
    {
        get { return largura; }
        set
        {
            if (value < 0)
            {
                throw new Exception("O valor da largura não pode ser menor que zero");
            }
            else
            {
                largura = value;
            }
        }
    }

    public double GetArea()
    {
        return Comprimento * Largura;
    }
}
```

3) Como ficou o código

```
        public double GetArea()
        {
            return Comprimento * Largura;
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            var r = new Retangulo();
            try
            {
                WriteLine("Informe o comprimento: ");
                r.Comprimento = Convert.ToDouble(Console.ReadLine());
                WriteLine("Informe a largura : ");
                r.Largura = Convert.ToDouble(Console.ReadLine());
                WriteLine($"Area do Retangulo: {r.GetArea()}");
            }
            catch (ArgumentException argEx)
            {
                WriteLine($"Erro : {argEx} ");
            }
            ReadLine();
        }
    }
}
```

Neste código temos o encapsulamento aplicado de forma que o código esta mais aderente às boas práticas.

Agora a classe **Retangulo** tem somente uma responsabilidade: calcular a área do retângulo.

Nota: *Podemos tornar a propriedade **somente leitura** não definindo a propriedade **set**.*

```
public double Comprimento
{
    get { return comprimento; }
}
```

Vantagens de programar desta forma:

- Podemos criar as referências as campos apenas quando formos usá-los;
- Podemos verificar ou definir restrições na atribuição/obtenção de valores das propriedades;
- As propriedades permitem um acesso controlado aos campos;
- Como os estado da classe depende dos valores dos campos, usando propriedades podemos assegurar que valores inválidos não sejam atribuídos aos campos;

Que outros motivos para preferir usar propriedades públicas ao invés de campos ???

Campos(fields) não podem ser usados em interfaces
Campos(fields) não permitem ser validados diretamente
A implementação do databingind é feita pela vinculação de propriedades e não de campos.

Encapsulamento

16 conceitos comparativos de POO(Programação Orientada por Objetos)

11 Conceito – Associação

Sempre utilizada para relacionar duas classes sendo que os objetos podem se comunicar. Diria que seria como se uma conhecesse a outra.

16 conceitos comparativos de POO(Programação Orientada por Objetos)

12 Conceito – Agregação

Sendo visto que uma agregação indica que uma das classes do relacionamento é uma parte ou está contida em outra classe.

Sendo assim pode se dizer que seria uma união de classes.

Para formar uma única resposta!

16 conceitos comparativos de POO(Programação Orientada por Objetos)

13 Conceito – Composições

Seria um relacionamento com características todo por parte onde existe um entendimento entre todas as partes desta forma, se o todo não existir, as partes também não existirão.

Um exemplo de composição é o pé:

- Um pé é composto por dedos.
- Os dedos compõem o pé.
- Não há lógica em existir um dedo sem o pé, porém pode-se ter um pé sem um ou mais dedos.

16 conceitos comparativos de POO(Programação Orientada por Objetos)

14 Conceito – Herança

Uma característica bem pensada como uma família.

Imaginemos :

- Uma criança está a herdar as características dos pais
- Os pais herdaram algo dos avós, o que faz com que a criança também tenha características de seus avós, e a sucessivamente.

16 conceitos comparativos de POO(Programação Orientada por Objetos)

15 Conceito – Dependência

A dependência deixa visto que uma mudança na especificação de um elemento pode alterar o valor do elemento dependente.

16 conceitos comparativos de POO(Programação Orientada por Objetos)

16 Conceito – Interface

Pode-se dizer que a interface seria um contrato entre a classe e o mundo exterior.

Quando uma classe implementa uma interface, se compromete a fornecer o comportamento publicado por esta interface.

As classes ajudam a definir um objeto e seu comportamento e as interfaces que auxiliam na definição dessas classes.

As interfaces são formadas pela declaração de um ou mais métodos, os quais obrigatoriamente não possuem corpo.