Programação Orientada à Objetos

Aula 07
Encapsulamento e Modificadores de Acesso

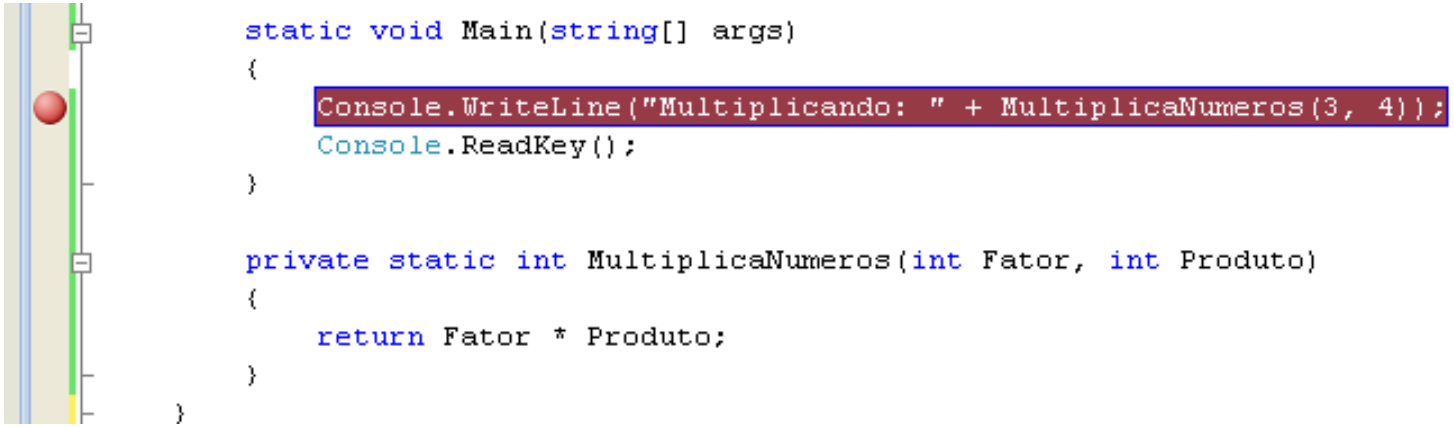
Henrique Poyatos
henrique.poyatos@bandtec.com.br

Depuração

- O processo de depuração consiste em procurar (e se possível encontrar) erros em um programa previamente escrito ou codificado.
- Através do Visual Studio poderemos adicionar pontos de parada (Breakpoint) para verificação de código, executar o programa passo a passo (Step Into e Step Over) além de possibilitar a visualização de valores de variáveis em tempo de execução.
- Teclas de atalho utilizadas
  - Breakpoint – F9 - Adiciona / Remove um Ponto de Parada
  - Step Into – F11 - Avança para a próxima instrução a ser executada, seja na própria função ou rotina em que a execução se encontra, seja em outra rotina qualquer.
  - Step Over – F10 - Avança para a próxima instrução a ser executada, dentro da própria função ou rotina em que a execução se encontra.
  - Quick Watch – Shift + F9 - Mostra o conteúdo de uma variável ou uma equação previamente selecionada
  - Next Statement – Ctrol + Shift + F10 - Avança ou retrocede a execução para a instrução selecionada.

Breakpoint

- Para adicionando ou remover um breakpoint : pressione F9 na linha desejada. A linha ficará destacada.

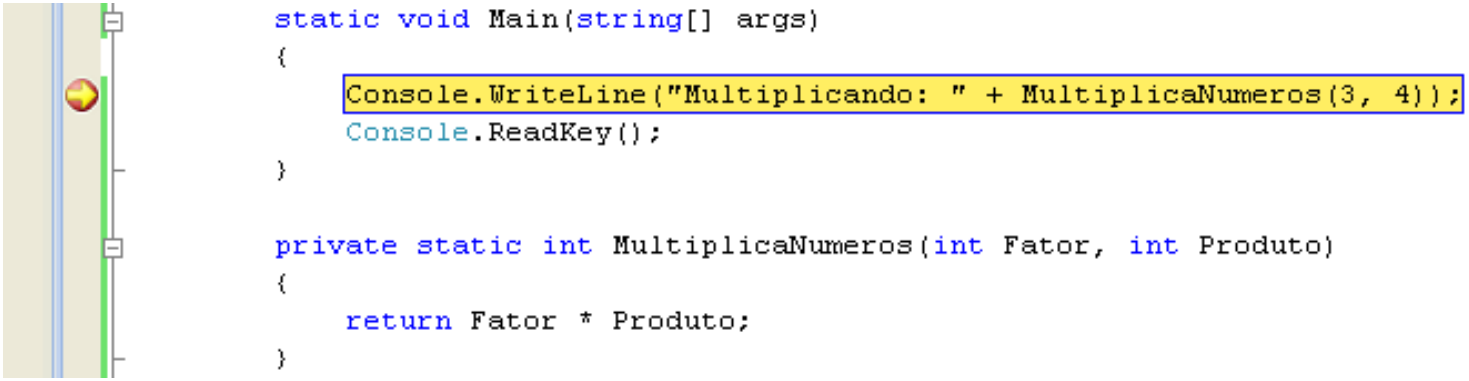


```
        static void Main(string[] args)
        {
            Console.WriteLine("Multiplicando: " + MultiplicaNumeros(3, 4));
            Console.ReadKey();
        }

        private static int MultiplicaNumeros(int Fator, int Produto)
        {
            return Fator * Produto;
        }
    }
```

Executando um Programa

- Ao executar o programa (F5) e a execução identificar o ponto de parada, a execução irá “parar” no breakpoint esperando que a continuação seja feita.

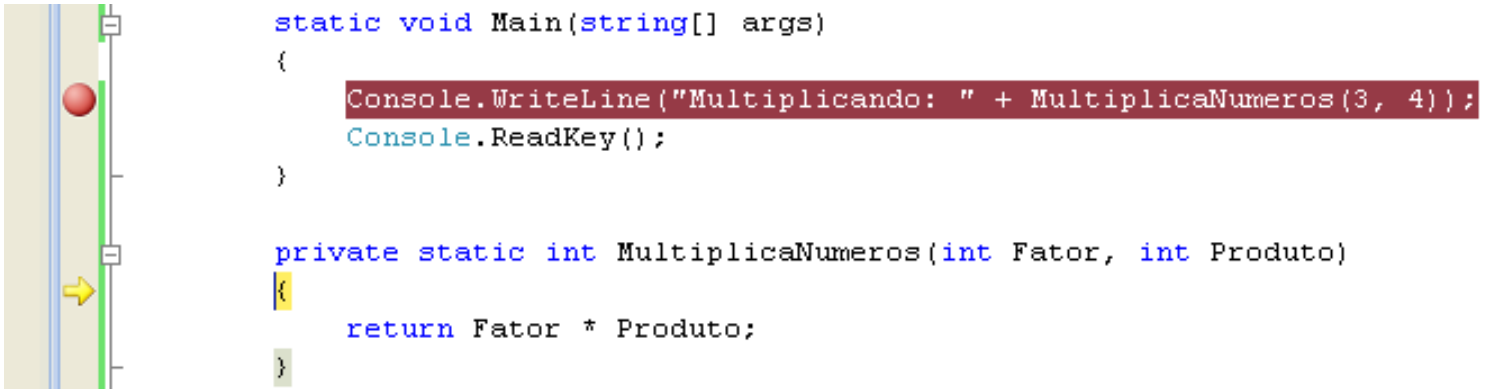


```
static void Main(string[] args)
{
    Console.WriteLine("Multiplicando: " + MultiplicaNumeros(3, 4));
    Console.ReadKey();
}

private static int MultiplicaNumeros(int Fator, int Produto)
{
    return Fator * Produto;
}
```

Step Into e Step Over

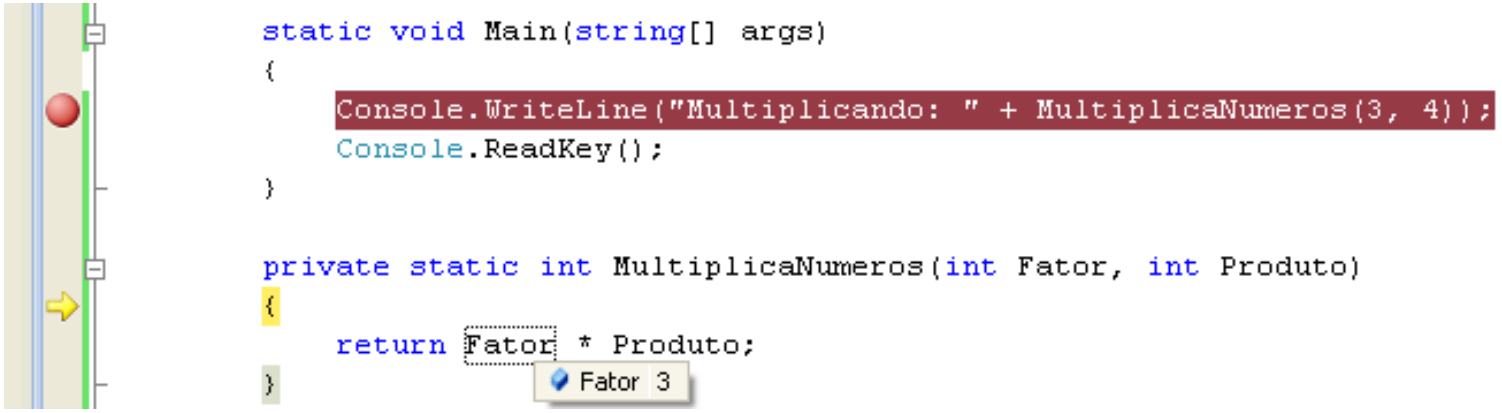
- Ao continuar a execução pressionando F11 a depuração “entra” dentro da função MultiplicaNumeros. Caso seja pressionado F10 ao invés de F11, o depurador executará a rotina porém seguirá na próxima da função que está sendo executada.



- Step Into - F11 – Avança o depurador para a próxima instrução, independente se a próxima instrução é da mesma rotina ou de outra rotina.
- Step Over - F10 – Avança o depurador para a próxima instrução da rotina que está sendo executada..

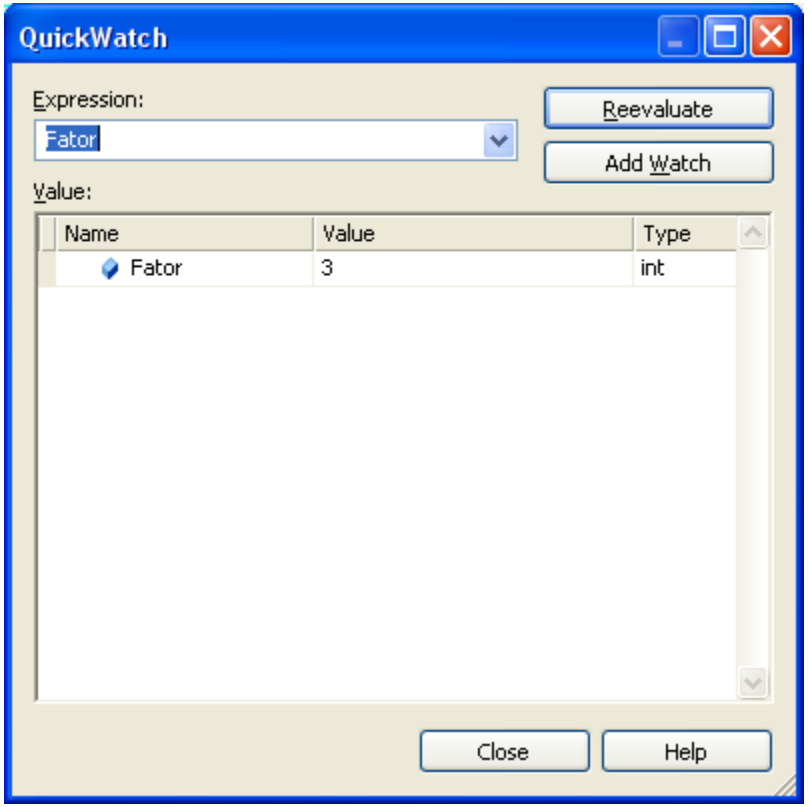
Verificando valores

- É possível averiguar o valor de uma variável simplesmente passando o mouse acima dela.



Verificando valores – Quick Watch (Shift+F9)

- Pode-se também, uma vez que o cursor está em algum ponto de uma variável acionar o Quick Watch através de Shift + F9. Através do Quick Watch pode-se modificar o valor da variável em tempo de execução.



Encapsulamento e Modificadores de Acesso

Atributos e Métodos são encapsulados e possuem os quatro tipos de **modificadores de acesso**:

public – significa que o atributo/método é público, ou seja, pode ser acessado e modificado por qualquer um.

private – apenas a classe que o possui (atributo/método) pode acessá-lo

protected – os atributos e métodos são acessíveis pela própria classe e por toda sua árvore genealógica – ou seja, filhos, netos e demais descendentes ligados por herança podem acessá-lo (atributo/método)

internal – todos dentro do projeto podem acessar. É o padrão quando nada é informado.

Encapsulamento e Modificadores de Acesso

Para prover o acesso quando necessário e evitar manipulações por engano, métodos públicos são criados para manipular indiretamente os atributos.

Estes métodos são costumeiramente chamados de

Setters (métodos que inserem valores nos atributos)

Getters (pegadores, métodos que recuperam o valor a partir dos atributos)

Getter & Setter Tradicional

```
class Cliente
{
    private float saldo;

     //Getter tradicional
    public float GetSaldo()
    {
        return this.saldo;
    }

    //Setter tradicional
    public void SetSaldo(float value)
    {
        this.saldo = value;
    }
}
```

Getter & Setter Simplificado (só C#!)

```
class Cliente
{
    private int codigo;

    public int Codigo
    {
        get { return this.codigo; }
        set { this.codigo = value; }
    }
}
```

Getter & Setter
MAIS Simplificado (só C#!)

```
class Cliente
{
    private bool ativo; //private
    public bool Ativo { get; set; }
}
```

Encapsulamento e Modificadores de Acesso

```
private string name;    // atributo
public string Name    // accessors
{
    get
    {
        return name;
    }
    set
    {
        name = value;
    }
}

//Utilização
e1.Name = "Joe"
Console.WriteLine(e1.Name);
```