IFRN
PROGRAMAÇÃO BÁSICA EM C#

Métodos

Prof. Gilbert Azevedo

Objetivos

- Entender o conceito de modularização
- Modularizar a escrita de programas utilizando métodos
- Conhecer os tipos de passagem de parâmetros
- Entender o escopo de variáveis em um programa

Modularização

- Divisão de um programa em conjuntos de instruções, chamados de sub-rotinas, com os seguintes objetivos
  - Facilitar a escrita do programa
  - Simplificar a leitura e manutenção do código
  - Facilitar a divisão de tarefas
  - Diminuir a repetição de comandos
  - Aumentar a reusabilidade de código
- Na Programação Orientada a Objeto, as sub-rotinas são chamadas de métodos e são escritas dentro de classes

Método

- Conjunto de instruções que realiza uma tarefa específica
  - Calcular uma raiz quadrada, calcular uma média, contar palavras em um texto
- Possui um identificador que é usado na sua chamada (ou invocação)
  - Sqrt, Pow, Parse, WriteLine, ReadLine, Media, ContarPalavras
- Pode ser invocado pelo método *Main*, por outros métodos ou por ele mesmo
  - Recursividade: o método chama ele mesmo

Método – Resumo

- Método é uma sequência de instruções usada para alguma tarefa específica, sendo composto por pelo menos quatro partes:
  - Um identificador: nome utilizado para chamar o método
  - Um tipo de retorno: informa o tipo de dado o método retorna
  - Uma lista de parâmetros: informa os dados que são necessários a execução do método
  - Um corpo: lista as instruções realizadas pelo método

Métodos do Framework

- Muitos métodos estão disponíveis no framework para realizar diversas tarefas.
  - Nestes casos, as instruções do método (corpo) foram escritas pelo programadores do framework.
- Exemplo
  - public static double Sqrt(double d)
  - Método usado para calcular a raiz quadrada de um número real (double)
  - Recebe um parâmetro d, que deve ser real (ou compatível)
  - Retorna como resultado um valor real

Exemplo

- public static double Sqrt(double d)
  - public – Modificador de acesso
  - static – Informa que o método é estático (deve ser chamado com a sua classe, neste caso Math)
  - double – Tipo do retorno: um double com a raiz quadrada de d
  - Sqrt – Identificador do método, utilizado para chamar o método
  - double d – Parâmetro, o valor double que terá sua raiz quadrada calculada
- Chamando o método
  - double x = Math.Sqrt(16);

Dicas sobre Métodos

- Métodos podem possuir o mesmo nome se tiverem listas de parâmetros diferentes (sobrecarga)

```
public int IndexOf(string value)
```

```
public int IndexOf(string value, int startIndex)
```

- Podem não receber parâmetros

```
public static string ReadLine()
```

- Podem não ter retorno, utilizando a palavra-chave *void* no tipo de retorno

```
public static void WriteLine(string value)
```

Escrevendo um Método

- Para escrever um método, é necessário informar:
  - Um tipo de retorno
  - Um identificador
  - Uma lista de parâmetros
  - Um conjunto de instruções

```
[public] [static] TipoRetorno NomeMetodo (ListaParametros)
{
    // Instruções
}
```

Escrevendo um Método

Acesso Tipo Retorno Identificador Lista de parâmetros

```
public static double AreaTriangulo(double b, double h)
{
   double area;
   area = b * h / 2;
   return area;
}
```

Variável local

Resultado retornado

- O resultado deve ser informado com a instrução return.

Chamando um Método

- Para chamar o método, é necessário utilizar seu nome e informar os valores que serão passados aos parâmetros (argumentos)

```
public static void Main(string[] args)
{
    double x = AreaTriangulo(10, 20);
    double y = Math.Sqrt(16);
}
```

- Se os métodos estiverem na mesma classe, não é necessário informar esta classe na chamada do método.

Métodos na Mesma Classe

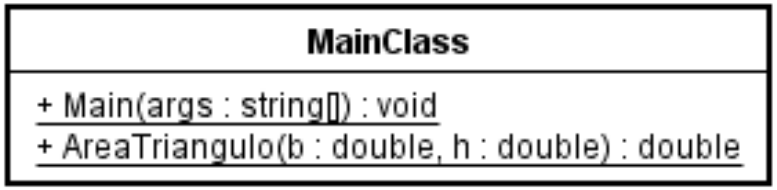
```
class MainClass {
  public static double AreaTriangulo(double b, double h) {
    double area;
    area = b * h / 2;
    return area;
  }
  public static void Main (string[] args) {
    double x, y;
    x = AreaTriangulo(10, 20);
    y = AreaTriangulo(5, 8);
    Console.WriteLine($"Áreas = {x} e {y}");
  }
}
```

Métodos em Classes Distintas

```
class Area {
  public static double AreaTriangulo(double b, double h) {
    double área = b * h / 2;
    return area;
  }
}
class MainClass {
  public static void Main (string[] args) {
    double x = Area.AreaTriangulo(10, 20);
    double y = Area.AreaTriangulo(5, 8);
    Console.WriteLine($"Áreas = {x} e {y}");
  }
}
```

Diagrama de Classes

- Um diagrama de classe da UML é usado para mostrar as classes de um aplicativo
  - Métodos na mesma classe



  - Métodos em classes distintas

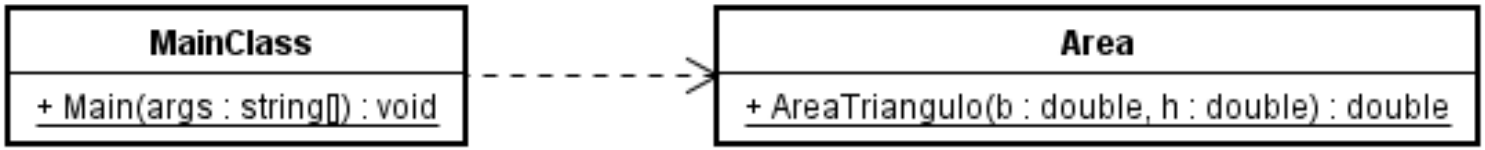
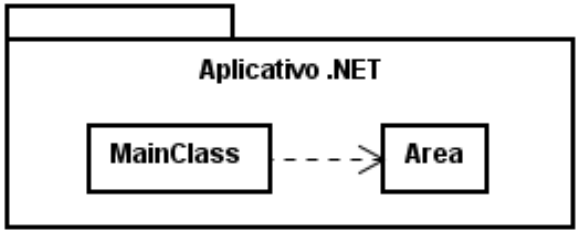
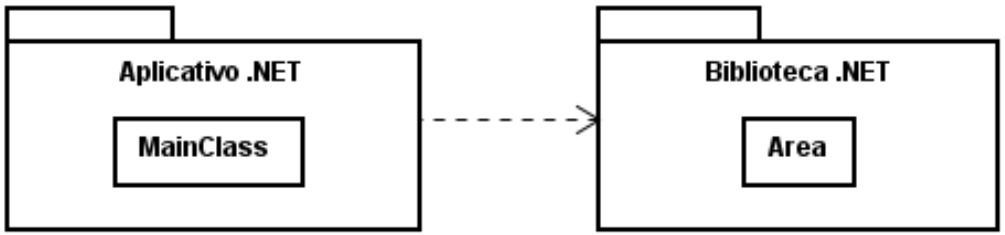


Diagrama de Pacotes

- Um diagrama de pacotes da UML é usado para mostrar os projetos de uma solução: Arquitetura do Software
  - Classes no mesmo projeto



  - Classes em projetos distintos



Passagem de Parâmetros

- A passagem de parâmetros para um método pode ser informada de três formas distintas:
  - Passagem de Parâmetros por Valor
    - Forma padrão mais comumente utilizada. Não utiliza nenhuma sinalização no parâmetro.
  - Passagem de Parâmetros por Referência
    - Os parâmetros são sinalizados com o prefixo *ref*
  - Passagem de Parâmetros de Saída
    - Os parâmetros são sinalizados com o prefixo *out*

Passagem de Parâmetros por Valor

- Quando o método é chamado, os valores são copiados na memória e modificar o parâmetro, não altera o argumento.

```
public static void Metodo1(int a, int b) {
  int c = a;
  a = b;
  b = c;
}
public static void Main (string[] args) {
  int x = 10, y = 20;
  Metodo1(x, y);
  Console.WriteLine($"{x} e {y}"); // Escreve 10 e 20
}
```

Passagem por Referência

- Quando o método é chamado, os valores são compartilhados na memória e modificar o parâmetro, altera o argumento.

```
public static void Metodo2(ref int a, ref int b) {
  int c = a;
  a = b;
  b = c;
}
public static void Main (string[] args) {
  int x = 10, y = 20;
  Metodo2(ref x, ref y);
  Console.WriteLine($"{x} e {y}"); // Escreve 20 e 10
}
```

Parâmetros de Saída

- Semelhante aos parâmetros passados por referência. Mas, neste caso, os argumentos não podem ser constantes e devem ter seu valor definido no método.

```
public static void Metodo3(out int a, out int b) {
  a = 10;
  b = 20;
}
public static void Main (string[] args) {
  int x, y;
  Metodo3(out x, out y);
  Console.WriteLine($"{x} e {y}"); // Escreve 10 e 20
}
```

Escopo Local

- O escopo define onde um identificador é válido.
- Variáveis definidas dentro de um método possuem escopo local e são chamadas variáveis locais.

```
class Exemplo01 {
  public static void Metodo1() {
    int x = 0; // Variável Local
  }
  public static void Metodo2() {
    x = 10; // Erro - Fora do Escopo
  }
}
```

Escopo da Classe

- Variáveis definidas dentro de uma classe podem ser vistas em todos os seus métodos.

```
class Exemplo02 {
  static int x;
  public static void Metodo1() {
    x = 0; // Ok - Escopo da Classe
  }
  public static void Metodo2() {
    x = 10; // Ok - Escopo da Classe
  }
}
```

Referências

- Microsoft Visual C# 2010 – Passo a passo, John Sharp, Bookman, 2010
- UML 2 – Uma Abordagem Prática, Gilleanes T. A. Guedes, Novatec, 2018
- Métodos (Guia de Programação em C#)
  - https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/classes-and-structs/methods
- Namespaces (Guia de Programação em C#)
  - https://docs.microsoft.com/pt-br/dotnet/csharp/programming-guide/namespaces/

Fim

- Tarefa
  - Questionário
- Próxima Aula
  - Introdução à POO