***Teoria de Métodos***

Os métodos têm uma função especifica dentro de um projeto. Nos projetos com alguma dimensão torna-se imprescindível dividir o código em subprogramas, de modo a facilitar a tarefa de programar. Isto permite o reaproveitamento do código.  
Como membros de uma classe, os métodos podem ter valor de retorno ou não, ou seja, podem devolver um valor para a linha de comando que o chamou ou apenas executar um conjunto de instruções. Os benefícios destas técnicas são claros: Código mais percetível, mais fácil de conceber e de manter.  
A capacidade de passar argumentos e retornar valores é opcional e depende do que se quer que o método faça.  
Os métodos podem levar um modificador que indica a sua visibilidade para o exterior da classe (private, protected, public, internal, e protected internal)

Ex:

[modificador] [tipo de retorno] [nome do método] ( [argumentos] )

{

[instruções]

}

Os argumentos dos métodos declaram valores que podem ser passados a partir da instrução que os chamou. Nesse sentido, os argumentos correspondem a declarações de variáveis do próprio método.

Ex de um método que devolve a data do dia seguinte:

Public DateTime ProxDay()

{

//adiciona-se um dia à data atual

DateTime ProximoDia=DateTime.Now.AddDays(1);

Return ProximoDia;

}

Os métodos também podem ser chamados a partir de expressões. Por exemplo: MessageBox.Show(“Dia de pagamento é” + ProxDay().ToString());

***Argumentos opcionais***

Num método podem especificar-se argumento como opcionais. Isto significa que se trata de argumentos que podem, ou não, ser passados. Para declarar um argumento como opcional temos de inicializar o argumento com um determinado valor inicial.

***Passar argumentos para métodos***

Um argumento é um valor passado para um método com base no qual ele pode executar cálculos, determinar comportamentos, entre outros.

Ex:

Public double AreaRetangulo(double lado1, double lado2)

{

double area = lado1 \* lado2;

return area;

}

Quando se pretende chamar ou invocar o método anterior, o programador terá de indicar, para além do nome, os respetivos argumentos, como:

Double\_areaFinal = AreaRetangulo((double) txtLado1.Text, (double)txtLado2.text)

Sempre que o método é chamado, o programador tem de disponibilizar os valores especificados na definição do método e na mesma ordem. Por outro lado, os argumentos passados devem ser do mesmo tipo de dados dos argumentos definidos. Se um método espera um valor do tipo double, não pode ser passado um valor do tipo string.

***Mecanismos de passagem de argumentos***

Passagem de argumentos por valor- É o mecanismo assumido por definição. Neste caso é passada uma cópia do conteúdo das variáveis para o método;

Passagem de argumentos por referência- Aqui não é enviado o conteúdo da variável, mas o seu endereço de memória. Isto significa que o método pode alterar permanentemente o conteúdo da variável passada.

Ex: private double PorReferencia(ref double val1, ref double val2)

***Número indeterminado de argumentos***

Quando não se sabe quantos argumentos são necessários, faz-se uso da palavra Params aquando da definição do método. Desta forma, é possível aceitar um número desconhecido de argumentos.

Ex: private int calcular (params int [] valores)

Cálculo auxiliar:

int[] vetor = new int[10];

int maior = vetor.Max();

int menor = vetor.Min();

double med = vetor.Average();

***Exercício***

Exercício que calcula a soma de um conjunto de valores indicados pelo utilizador. Todos os valores introduzidos são enviados para uma listBox através do botão adicionar.

Elabore um método calcular com parâmetros indeterminados eu realize a soma de todos os valores introduzidos (use o foreach).

O botão calcular envia os valores do list box para um array e apresenta a soma dos números (invocação do método calcular).