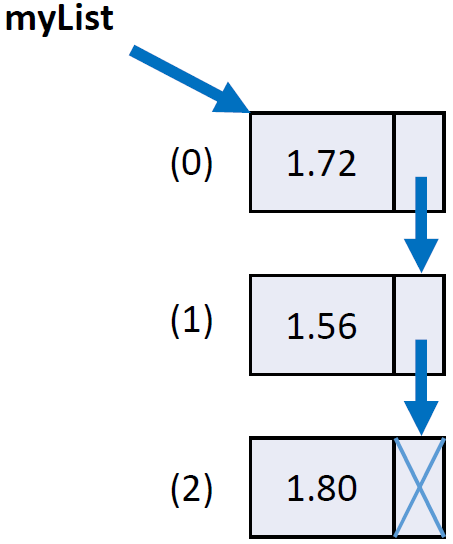
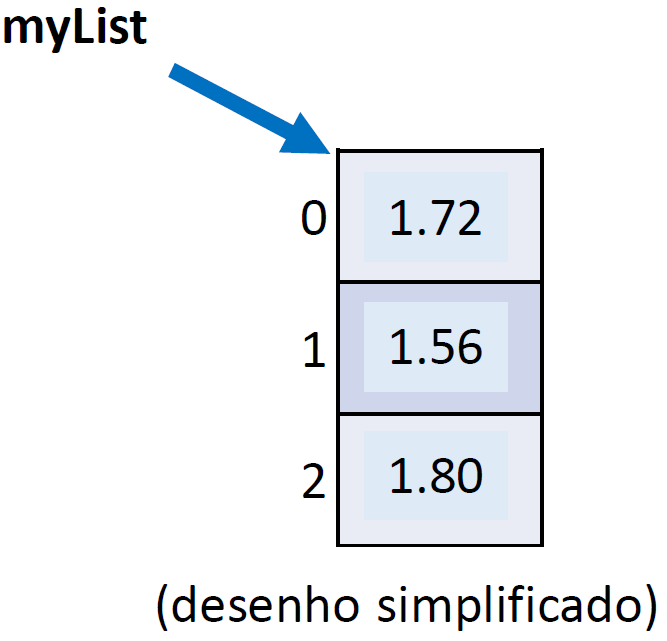
# Listas – list 04/12/2020

## Lista é uma estrutura de dados:

* Homogênea (dados do mesmo tipo).
* Ordenada (elementos acessados por meio de posições).
* Inicia vazia, e seus elementos são alocados sob demanda.
  + Diferente do vetor, que ao instanciar, já precisa informar o tamanho.
* Cada elemento ocupa um "nó" (ou nodo) da lista.



A seta é uma referência para os nodos relacionados. O ultimo nodo não aponta para ninguém.



# Classe: List

Para utilizar, precisamos adicionar uma biblioteca.

Namespace: System.Collections.Generic

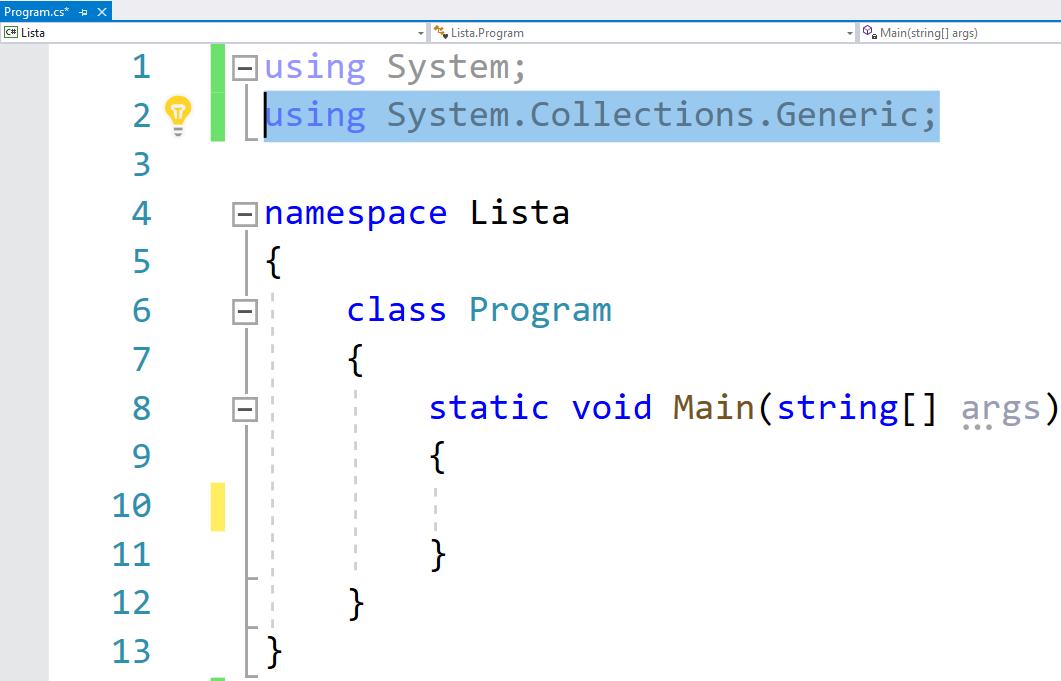
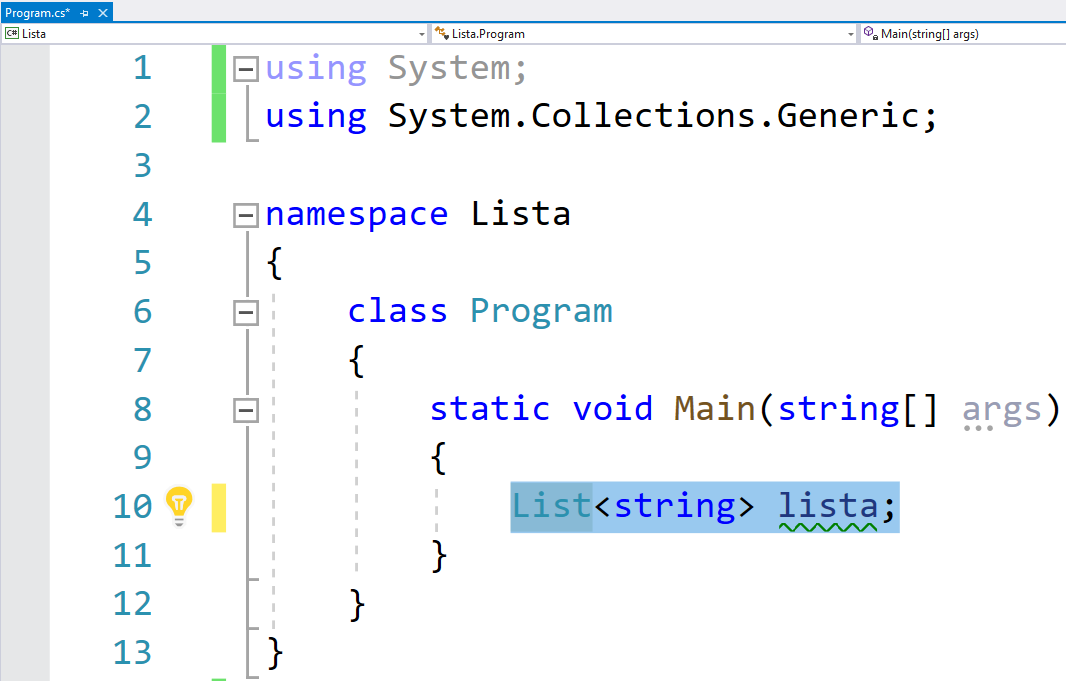
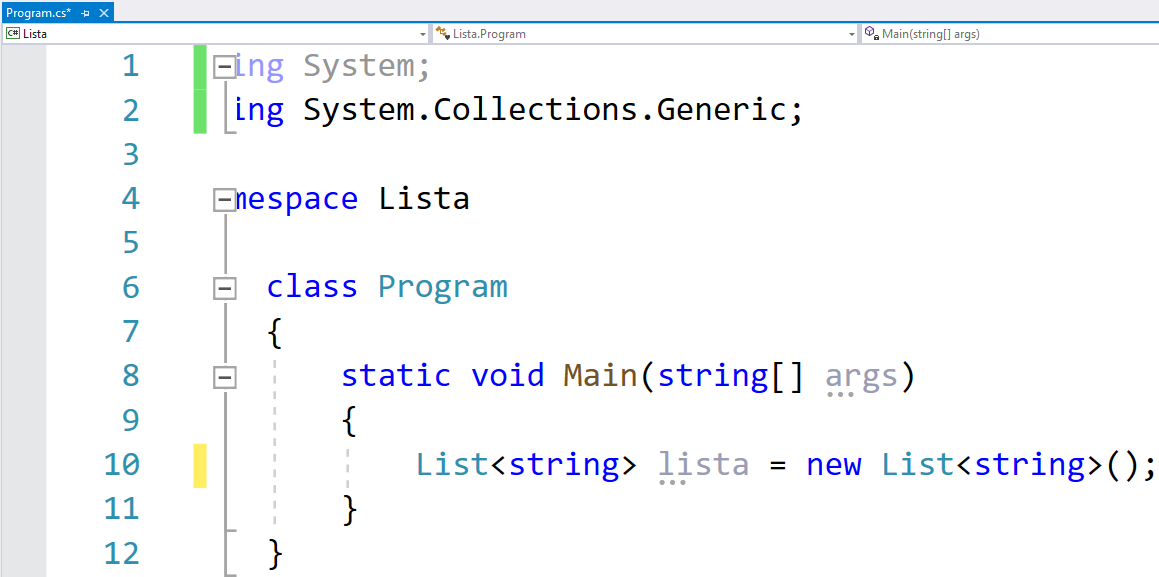


Figura 1. Biblioteca adicionando.

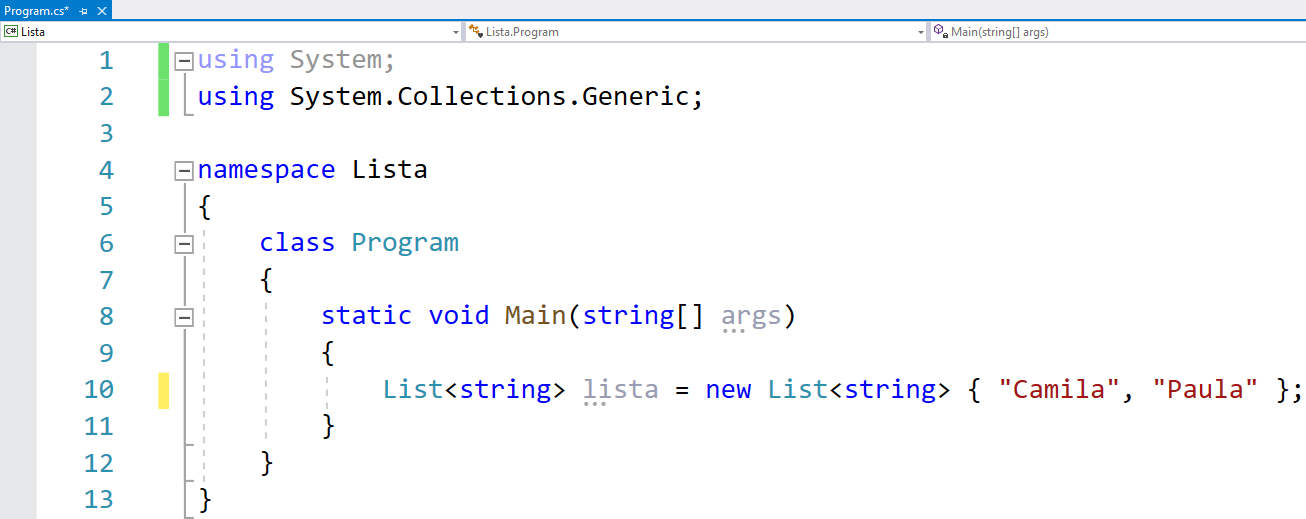
Criando uma lista



Podemos criar e instanciar a lista ao mesmo tempo.



Podemos também criar a lista já adicionando elementos.



Vantagens:

* Tamanho variável.
* Facilidade para se realizar inserções e deleções.

Desvantagens:

* Acesso sequencial aos elementos \* (Precisa percorrer toda a lista para acessar um elemento, o que perde performance).
  + Ressalva para listas otimizadas, que possuem um acesso razoavelmente rápido.

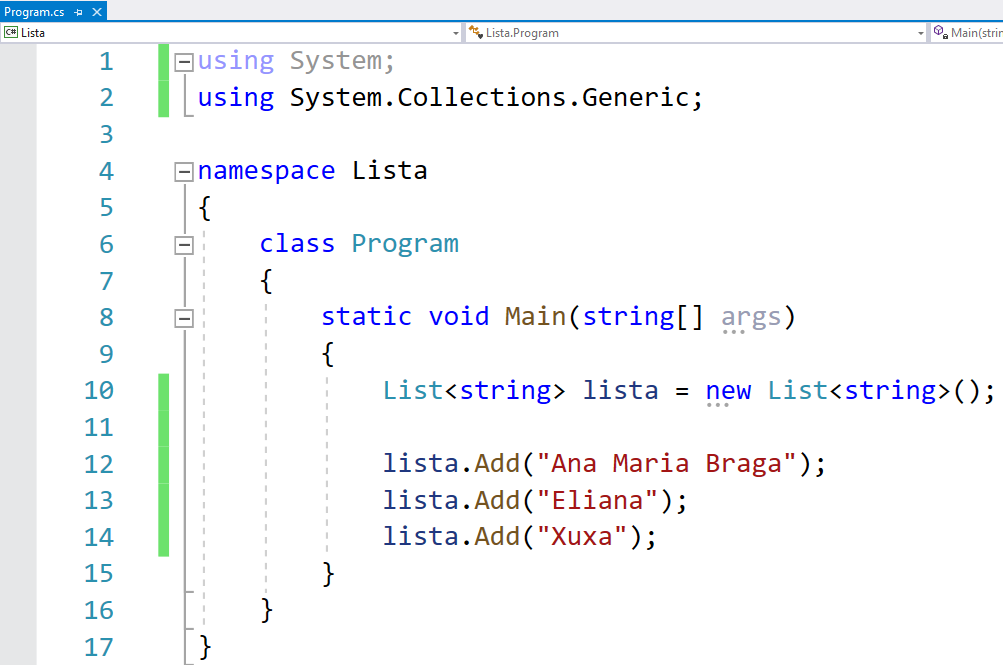
Ou seja, as desvantagens do vetor são corrigidas na lista. Porém, a desvantagem da lista não existe no vetor.

# Inserindo elementos na lista: Add, Insert

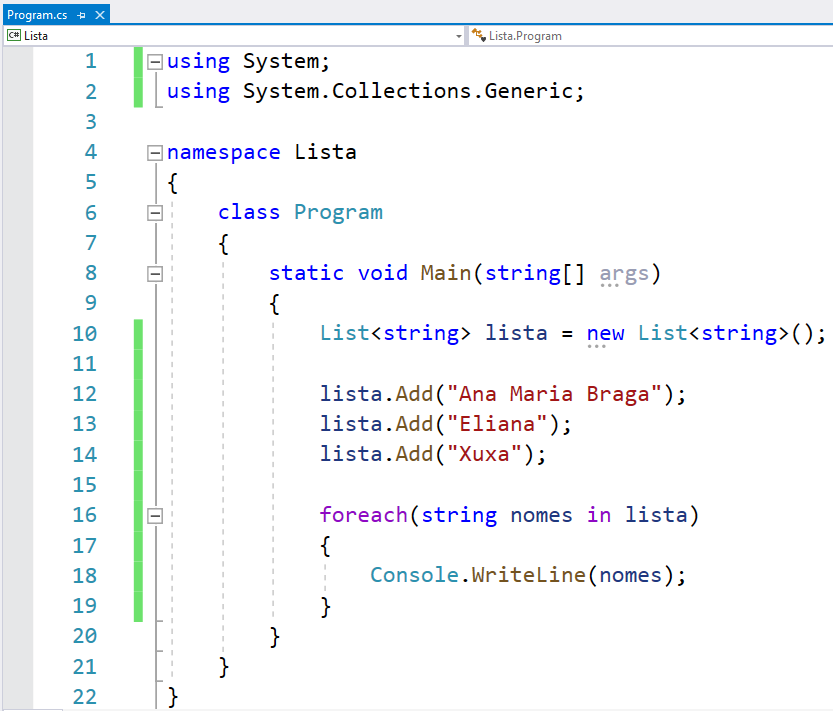
O método Add adiciona na mesma ordem inserida.

Sintaxe:

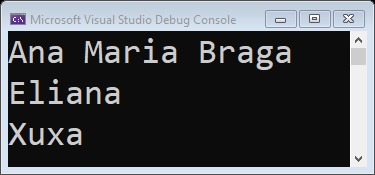
nomeDaLista.Add(“valor”);



Vamos testar com um foreach



Ok:

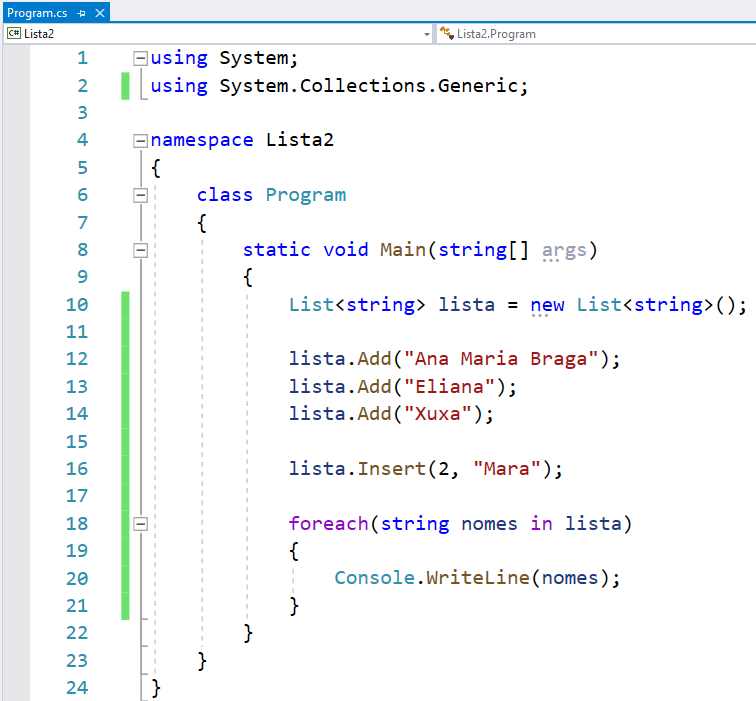


# Método Insert.

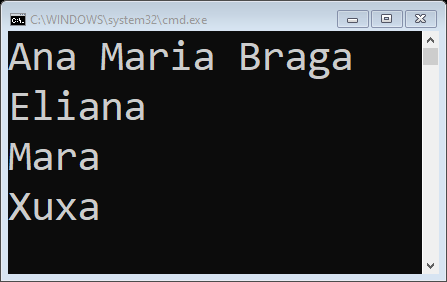
Com esse método, é possível inserir o elemento na posição desejada.



A primeira instrução é a posição, e a segunda o valor.

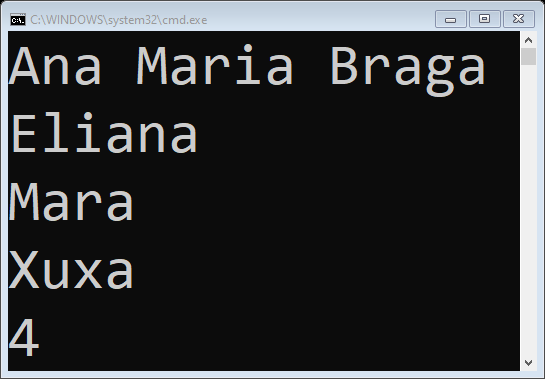


Ok



# Count – Tamanho da lista





# Find – Encontrar primeiro

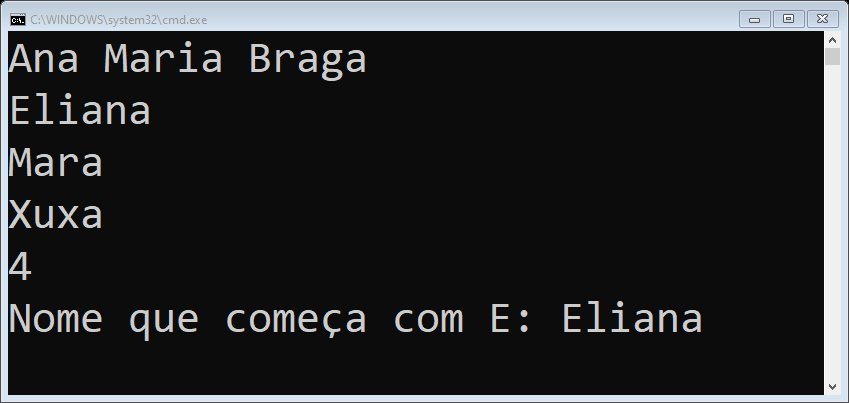
Encontrar o primeiro que satisfaça um predicado:

Exemplo: Vamos buscar a primeira pessoa que começa com a letra E

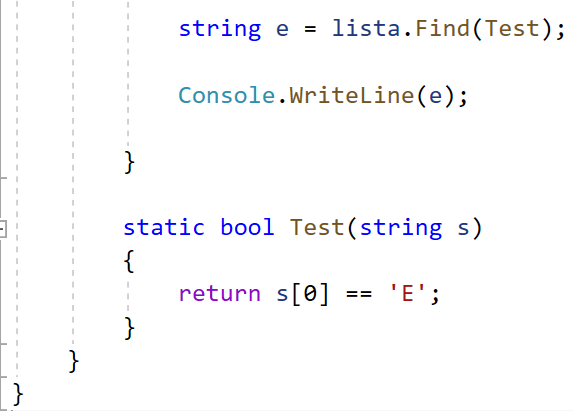


O Find espera um predicado como argumento.

Predicado é uma função anônima ou função lambida. (Veremos posteriormente).



Uma outra forma de realizar uma busca, seria através da criação de uma função:



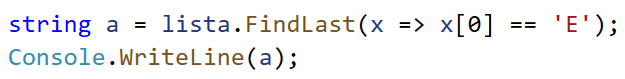
# FindLast – Encontrar ultimo

Encontrar o ultimo que satisfaça um predicado:

Vou acrescentar um outro nome com a letra E para testar.

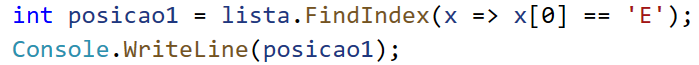


Agora executando

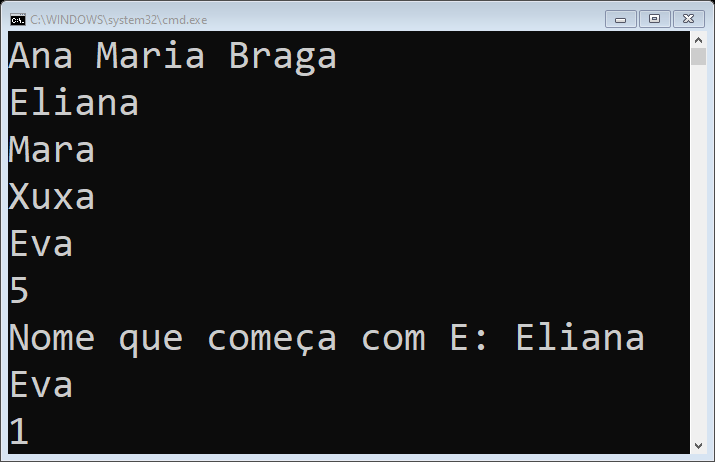


# FindIndex – Encontrar a primeira posição

Encontrar a primeira posição que satisfaça um predicado:



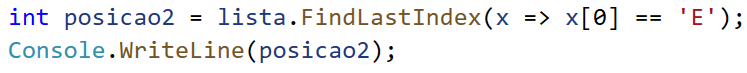
Observando:



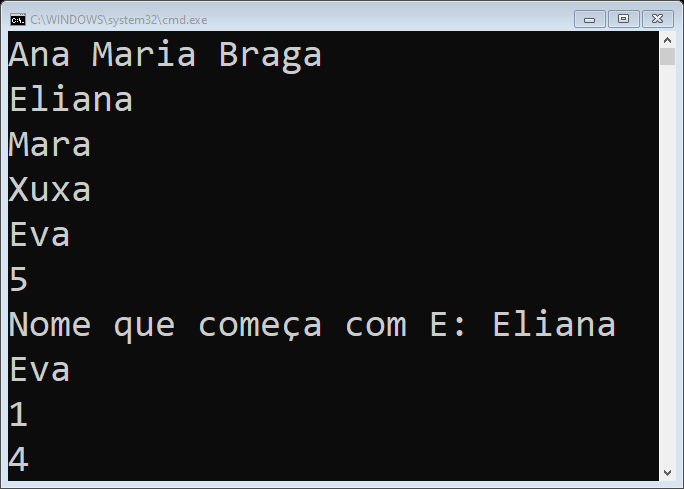
Sim, Eliana está na posição 1 da lista.

# FindLastIndex – Encontrar a última posição

Encontrar a última posição que satisfaça um predicado:



Observando:



Eva está na posição 4 da lista.

# FindAll – Filtrar a lista

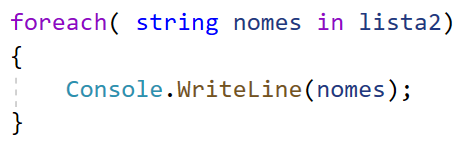
Filtrar a lista com base em um predicado:

Exemplo:

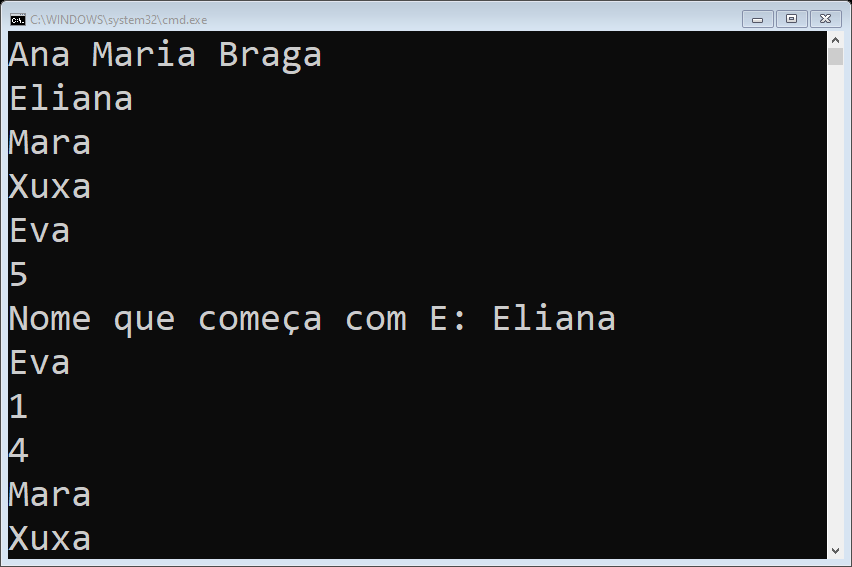
Quero os nomes que tenham 4 letras. Para isso, precisamos declarar uma outra lista para receber os resultados da primeira lista.



Um foreach para exibir a lista2



Observando:



Ok, a Mara e a Xuxa possuem 4 letras.

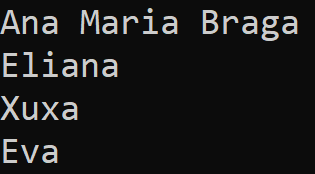
# Removendo elementos da lista

# Remove()

O remove espera uma string como parâmetro. Exemplo: Vamos remover a Mara.



Observando:



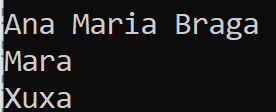
# RemoveAll()

O remove tudo que satisfaz o predicado.

Exemplo. Vamos remover todos que começam com a letra E.



Observando:



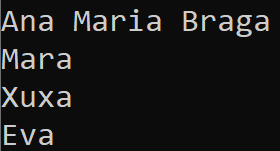
# RemoveAt()

Remove o elemento na posição específica.

Exemplo. Vamos remover o elemento da posição 1.



Observando:



A Eliana foi removida.

Obs. Ao tentar remover alguém que não existe, simplesmente não acontece nada.

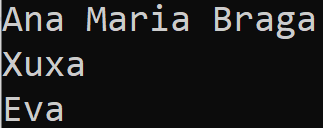
# RemoveRange()

Remove os elementos de uma faixa. Ele espera como predicado uma posição e uma contagem.

Exemplo. A partir da posição 1, vou eliminar 2 elementos.



Observando:



Removeu a Eliana e a Mara.

# Código de completo com a busca criando função:

using System;

using System.Collections.Generic;

namespace Lista2

{

class Program

{

static void Main(string[] args)

{

List<string> lista = new List<string>();

lista.Add("Ana Maria Braga");

lista.Add("Eliana");

lista.Add("Xuxa");

lista.Insert(2, "Mara");

foreach(string nomes in lista)

{

Console.WriteLine(nomes);

}

Console.WriteLine(lista.Count);

string e = lista.Find(Test);

Console.WriteLine(e);

}

static bool Test(string s)

{

return s[0] == 'E';

}

}

}

# Código completo da aula:

using System;

using System.Collections.Generic;

namespace Lista2

{

class Program

{

static void Main(string[] args)

{

List<string> lista = new List<string>();

lista.Add("Ana Maria Braga");

lista.Add("Eliana");

lista.Add("Xuxa");

lista.Add("Eva");

lista.Insert(2, "Mara");

foreach(string nomes in lista)

{

Console.WriteLine(nomes);

}

Console.WriteLine(lista.Count);

string e = lista.Find(x => x[0] == 'E');

Console.WriteLine("Nome que começa com E: " + e);

string a = lista.FindLast(x => x[0] == 'E');

Console.WriteLine(a);

int posicao1 = lista.FindIndex(x => x[0] == 'E');

Console.WriteLine(posicao1);

int posicao2 = lista.FindLastIndex(x => x[0] == 'E');

Console.WriteLine(posicao2);

List<string> lista2 = lista.FindAll(x => x.Length == 4);

foreach( string nomes in lista2)

{

Console.WriteLine(nomes);

}

// lista.Remove("Mara");

// lista.RemoveAll(x => x[0] == 'E');

// lista.RemoveAt(1);

lista.RemoveRange(1,2);

Console.WriteLine("--------------------");

foreach (string nomes in lista)

{

Console.WriteLine(nomes);

}

Console.ReadKey();

}

}

}

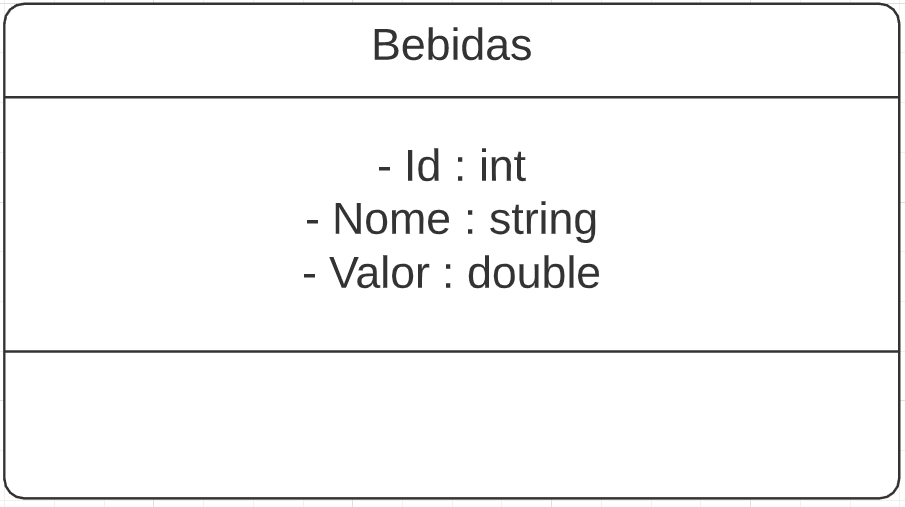
# Exercício de fixação:

1) Crie uma lista com 10 livros.

2) Utilize cada um dos métodos estudadas hoje.

3) Utilizando POO. Crie um programa que solicite dados para bebidas.

Execute esse projeto:

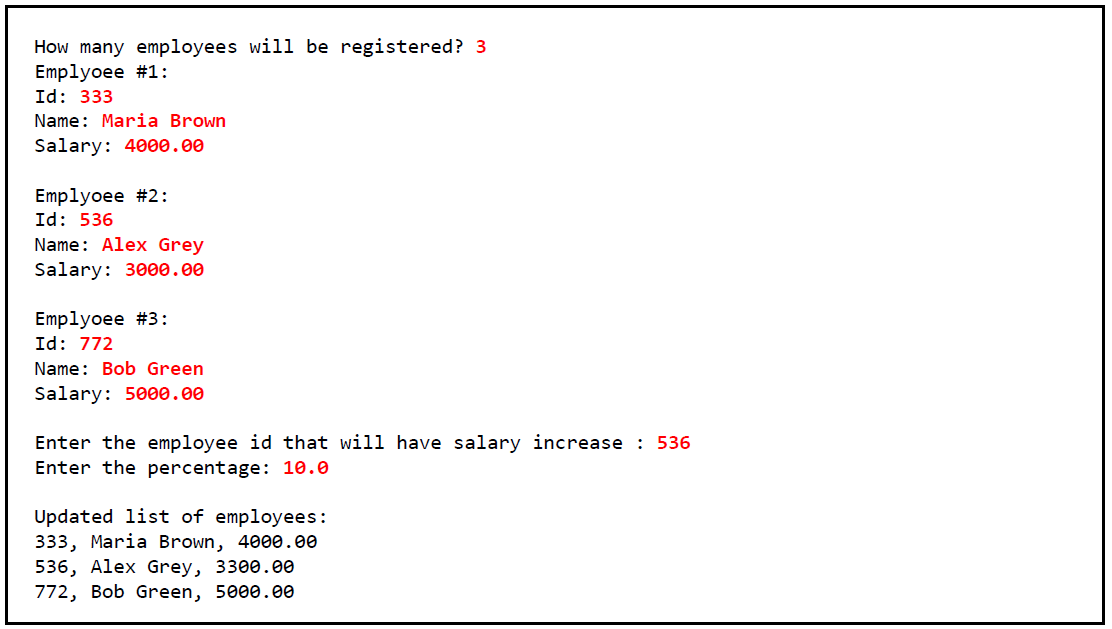


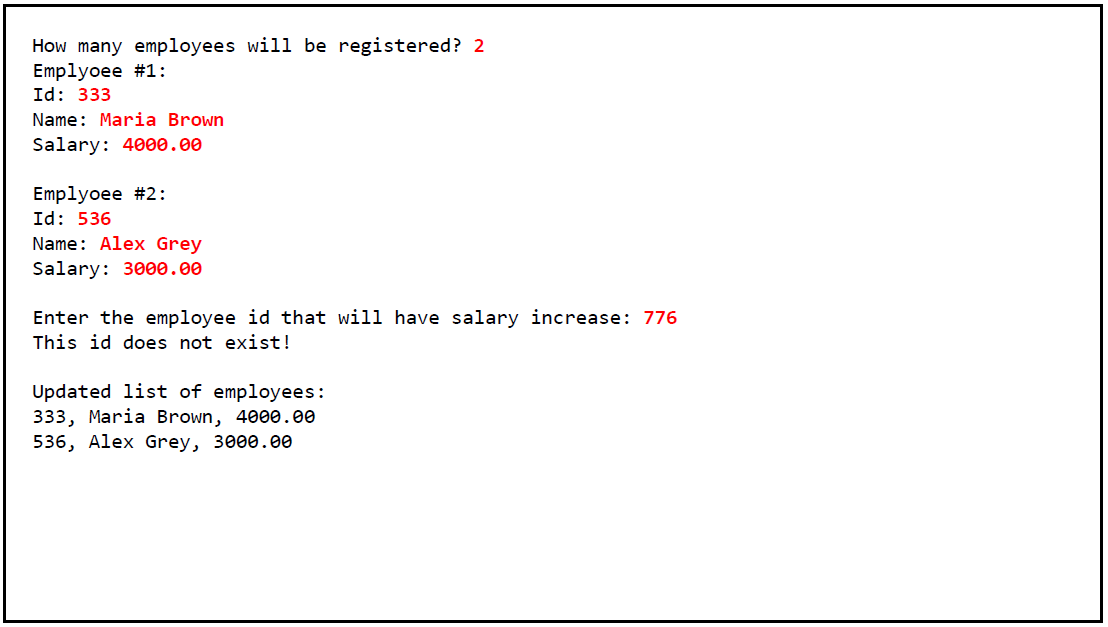
3) **(DESAFIO)** Fazer um programa para ler um número inteiro N e depois os dados (id, nome e salario) de N funcionários. Não deve haver repetição de id.

Em seguida, efetuar o aumento de X por cento no salário de um determinado funcionário. Para isso, o programa deve ler um id e o valor X. Se o id informado não existir, mostrar uma mensagem e abortar a operação. Ao final, mostrar a listagem atualizada dos funcionários, conforme exemplos.

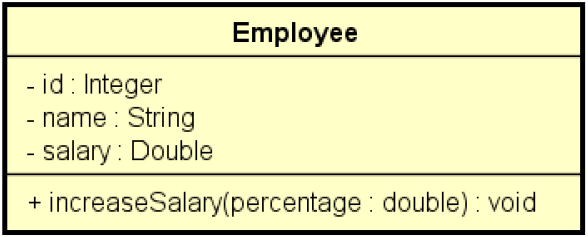
Lembre-se de aplicar a técnica de encapsulamento para não permitir que o salário possa ser mudado livremente. Um salário só pode ser aumentado com base em uma operação de aumento por porcentagem dada.

(exemplo na próxima página)





Desenho da classe:



# Referência:

Docs Microsoft – List <T> Class – <https://docs.microsoft.com/en-us/dotnet/api/system.collections.generic.list-1?redirectedfrom=MSDN&view=net-5.0> Acessado dia 27 de novembro de 2020.