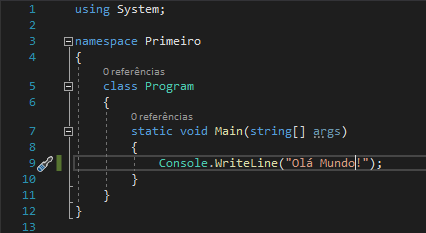
AULA 1 C#

-USING SYSTEM

Referência padro do .net, que é o responsável para realizar as funções, por exemplo, CONSOLE.WRITELINE.



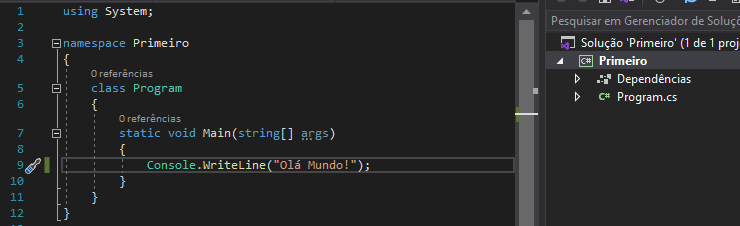
Então lembrar sempre de colocar o using system, senão, as funções no C# não vão funcionar.

-NAME SPACE

Nome do programa, aonde o nome se da quando você cria o projeto, mas você pode trocar para qualquer nome que quiser, apenas por questões de boa praticas.

-CLASS

Colocar sempre o nome das classes, com o nome do arquivo, para não se perder depois, igual no exemplo a seguir:



Se notar dentro do código fonte, existe a CLASS PROGAM, que está vinculado ao nome PROGRAM.C#, do lado direito da imagem, no gerenciador de soluções.

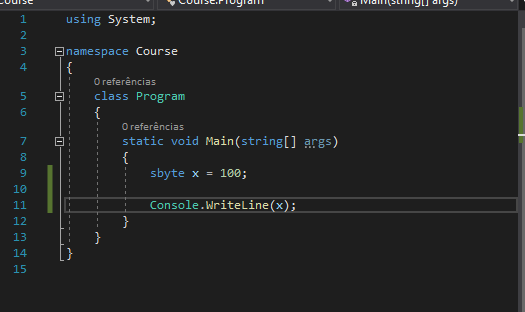
-INDENTAÇÃO AUTOMATICA

Apenas uma forma para deixar todo o código fonte alinhado com o que está dentro das chaves ou não, então para fazer basta apenas aperta a sequência:

CRTL+K+D.

-AULA 2

APLINCANDO O BASICO DE LOGICA DE PROGRAMAÇÃO USANDO C#



No exemplo acima, estamos declarando uma variável, que no caso é x, dizendo que o valor dela é 100, definindo também que essa variável vai ser um sbyte, que pode ter de -128 a 127 de valores dentro dela, tanto negativos, como positivos.

Lembrando que para executar o programa, para que o visual mostre o que estamos pedindo, basta apertar CRTL + F5, que vai abrir uma outra tela mostrando. Nesse exemplo vai apenas mostrar o numero 100.

-DECLARAÇÃO DE VALORES NO C#

INTEIROS – Quando for definir os valores inteiros dentro de uma variável, os tipos de valores mais usado, de valores positivos e negativos, são:

INT – COM 4 BYTES (USADO PARA NUMERO ATE BILHOES)

LONG – COM 8 BYTES (USADO PARA NUMEROS ATE TRILHOES)

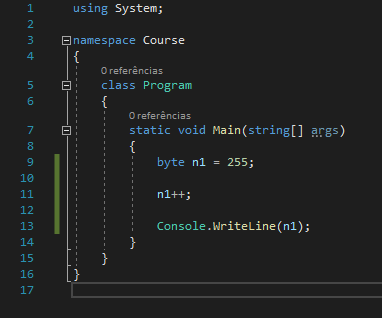
VALORES COM PONTOS OU VALORES COM VIRGULAS – no C# os mais utilizados tanto para números positivos, quanto para negativos, são:

FLOAT – COM 4 BYTES (USADO PARA NUMERO ATE BILHOES)

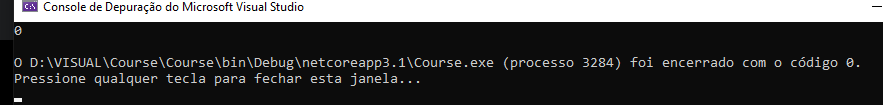
DOUBLE – COM 8 BYTES (USADO PARA NUMETO ATE TRILHOES)

DECIMAL – COM 12 BYTES (USADO PARA CALCULOS FINANCEIROS)

Caso a variável extrapole o limite dela, pode ocorrer o efeito overflow, que seria dar um “erro” na hora de executar o que esta dentro do código fonte, por exemplo:



Nesse caso, foi colocado que o n1 seria uma variável de byte, mas o byte so pode ter números positivos ate 255, passando disso, ele não aceita. Logo a baixo, foi colocado uma função de soma N1++, dizendo que é para fazer uma adição no valor da variável, aumentando mais um. Então, na hora de executar o código fonte, tem que aparecer 256, mas como o byte só tem o limite de 255 valores, isso não será mostrado, fazendo com que o executado mostre o numero zero no resultado.



Então sempre que for fazer um código fonte, lembre-se de sempre olhar o limite dos valores, para não dar errado.

OBSERVAÇÕES:

Se algum dia, for usar o LONG para limitar uma variável com um numero grande, lembre-se de sempre colocar a letra L no final do numero declarado dentro da variável, por exemplo:



Da para ver que no final do número, dentro da variável n4, tem a letra L, especificando que aquele número é uma variável LONG.

Quando for declarar uma varial, e essa mesma for de apenas UM caracter, pode colocar com CHAR, mas lembre-se, quando for texto, por exemplo, colocar apenas a sigla do sexo da pessoa, tem que ser entre ASPAS SIMPLES ‘’. Por outro lado, se for uma variável declarada como texto, UMA STRING, então deve colocar esse texto em ASPAS DUPLAS “”:

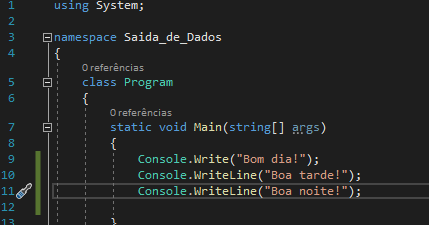


Essa é a diferença quando a variáveis CHAR e STRING, CHAR tem aspas simples, STRING tem aspas duplas.

AULA 3

SAIDA DE DADOS NO C#

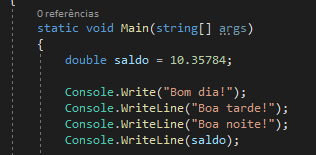
Saída de dados seria basicamente o que o sistema mostra para o usuário, no caso, os comandos que precisamos escrever para que ele nos mostre o que estamos precisando. No caso do C#, o comando responsável para mostrar o resultado ou algum texto, é o comando “CONSOLE.WRITELINE” ou “CONSOLE.WRITE”. A diferença entre os dois, é que quando tive LINE no final, ele faz uma quebra de linha, fazendo com que a mensagem ou resultado fique separada com as informações da página, por exemplo:





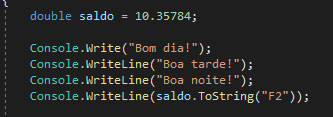
Podem ver que o bom dia e boa tarde estão juntos, pois no código fonte, o bom dia não tem LINE no comando, apenas a boa tarde e boa noite.

Um outro exemplo de saída de dados, seria os de controle de números, no caso, quando existe um numero quebrado, mas tem muita casa depois da virgula, e você quer apenas dois números depois dela, basta fazer desse jeito:



Podem ver que na variável saldo, declarada como Double, ele vai mostrar o resultado do jeito que está escrito, se você colocar console.writeline(saldo). A única coisa que vai mudar, é que ele vai mostrar com vírgula em vez de ponto, pois é o padrão brasileiro que você escolheu na hora de instalar o Visual Studio.

Se na hora de mostrar, você deseja por exemplo, mostrar apenas dois números depois da virgula, em vez de tudo isso, basta apenas fazer o seguinte passo:



Na hora de colocar o Console para mostrar o número, basta adicionar ENTRE PARENTESES O NOME DA VARIAVEL, PONTO, ToString, ABRE PARENTES NOVAMENTE, ENTRE ASPAS DUPLAS ESCREVA F2. Isso vai fazer com que o resultado mostre apenas duas casas depois da virgula.



Lembrando que ele vai arredondar pra mais, no caso no exemplo deu 10,36. Caso queira mais número depois a virgula, basta apenas mudar o f2 pra outro numero no lugar do 2.

Outro exemplo usando esse mesmo de cima, é sobre o ponto ou virgula como separador de números decimais, no caso como dito a cima, no brasil usamos a virgula para separar, mas se caso você opte por ponto na hora de mostrar o resultado, bastar fazer o seguinte passo:

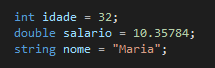




Adicionar o USING SYSTEM.GLOBALIZATION no começo do código fonte, não pode esquecer, senão não funciona. Depois basta colocar o CONSOLE.WRITELINE(SALDO.TOSTRING(“F4”, CULTUREINFO.INVARIANTCULTURE));



Mais um exemplo de saída de dados é o de juntar as informações dentro de um console write, que funciona desse jeito:



Aqui fizemos algumas declarações de variáveis com algumas informações.



Pode notar que o console está tudo dentro de aspas duplas, pois será uma string, e os números dentro de chaves, são a numeração sequencial que você vai colocar as variáveis, que estão no final do código. Então o primeiro é 0, que será a variável nome, depois o 1 que será a variável idade e depois o 2 que será a variável salário. E por fim, vai ficar assim:



Agora vamos dar um up nesse código, em vez de mostrar com varias casas decimais, vamos mostrar apenas com duas casas depois da virgula:





Para fazer com que ele mostre apenas duas casas depois da virgula, basta colocar dentro da chave que você vai mudar, no caso do salário que é o número 2, :F2, que vai ser a quantidade depois da virgula dentro dessa variável. E pode se notar que esse é mais fácil do que o outro jeito para tirar os números depois da virgula, só não pode esquecer de colocar o using globalization no início do código.

Existe outra maneira de colocar essas informações em uma única STRING também, que é a INTREPOLAÇÃO:



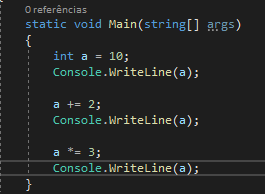
Basta colocar depois do console entre parênteses um sinal de cifrão, depois abre aspas duplas, chaves e dentro dessas chaves, basta apenas colocar direto as variáveis, em vez de numerar qual vai ser primeiro ou não, e depois no salário, para controlar a quantidade de números depois da virgula, basta fazer a mesma coisa, :F2.



Pode ver que tem duas frases iguais, dos dois exemplos, e os dois deram certos.

AULA 4

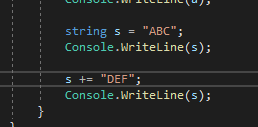
OPERADORES DE ATRIBUIÇÃO





No primeiro exemplo, está querendo dizer que a variável A, que é int., está recebendo o valor de 10. Na segunda linha, está mostrando que o a vai receber ele e mais 2, então como a gente já decidiu na primeira linha que o A vai ser 10, o sistema vai pegar esse 10 e vai acrescentar mais 2, dando o valor de 12. Em sequência ele está dizendo que o A vai receber ele multiplicado por 3, então como o A agora vale 12, ele vai multiplicar esse valor por 3, dando um total de 36.

Da pra fazer também com string, que seria concatenar texto:



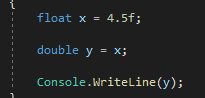
Mesma coisa do exemplo de números, nesse caso estamos definindo que a variável S, vai receber a string ABC, na primeira linha, depois, com o S += “DEF”, quer dizer que essa string vai receber também, o texto “DEF”.

AULA 5

CONVERSÃO IMPLICITA E CASTING

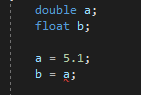
Isso é quando colocamos uma variável de limitação diferente dentro de outra, por exemplo, quando uma variável é definida de FLOAT, que tem 4 bytes para números quebrados, que seriam números com virgulas, e pegamos essa mesma variável e colocamos ela dentro de outra variável, que seria definida por DOUBLE, que tem 8 bytes.

Isso pode acontecer por um motivo, a variável DOUBLE tem espaço para 8 bytes, então pela lógica, ela caberia 4 bytes, que seria o caso da FLOAT, então nesse exemplo está rolando uma conversão implícita de duas variáveis diferentes. Então ficaria assim:



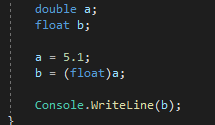
Nesse exemplo, estamos definindo que a variável x vai receber 4.5, e depois que a variável y vai receber a variável x.

Agora vamos declarar uma diferente, para ver o que da.



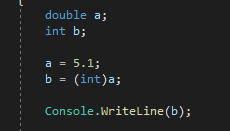
Nesse caso, já da pra ver que o sistema apontou um erro, falando que você esta tentando colocar uma variável dentro de outra, mas esse mesma variável não suporta o tamanho da outra. Então tem que sempre se atentar a isso, e lembrar dos bytes, pois se exceder o tamanho delas, não vai dar certo.

Caso você queira continuar com o processo, você pode, mas no caso vai perder algumas informações, só que pra isso, você terá que fazer a seguinte modificação:



Na variável que vai receber a informação da outra, vai ter que colocar depois do sinal de igual, a declaração de variável que você usou pra mesma, por exemplo, se você declarou que o B vai ser float, dentro dela entre parênteses, você coloca novamente e depois diz que ela vai receber o A mesmo assim. Fazendo desse jeito o sistema entende que você arca com as reponsabilidades e que não tem problema colocar informações de uma variável de 8 bytes dentro de 4 bytes.

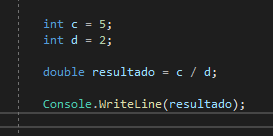
Outro exemplo seria de números quebrados para números inteiros:

Nesse caso estamos tentando colocar um valor quebrado de uma variável com 8 bytes, dentro de uma variável declarada INT com 4 bytes e sem numero quebrados. Ele da o mesmo erro que o de cima, avisando que a compatibilidade é menor. Mas nesse exemplo acontece outra coisa se dermos um F5 e colocarmos int depois do sinal de = :

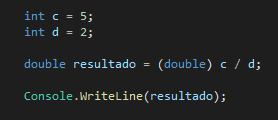
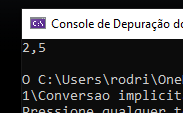


Ele da o resultado de inteiro apenas, sem os números quebrados, pois a variável que está recebendo, não tem a função de mostrar números quebrados, ou números com virgulas. Então vai ter quebra de informações.

Mais um exemplo, dois números inteiros fazendo uma divisão e mostrando em uma variável com 8 bytes e com números quebrados, com virgulas. Nesse exemplo ele não vai mostrar os números depois da virgula, pois tem que fazer o casting, modificar a variável igual nas de cimas para que ele considere também os números depois da virgula, pois como as primeiras variáveis são inteiras, ele continua como padrão, mesmo que você coloque a variável de resultado como double, que seria quebrado:



Então para que ele mostre o resultado em números quebrados, temos que fazer o casting depois do sinal de igual, falando para que o sistema nos mostre o resultado completo da divisão de números inteiros:

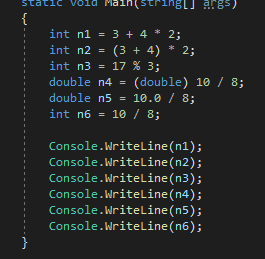


Dando o resultado de 2,5.

AULA 6

OPERADORES ARITMÉTICOS

Como já foi visto na logica de programação no youtube, as questões de operadores aritméticos não mudam muito, apenas algumas expressões de visualização, ou comandos que precisam ser colocados para visualizar o resultado completo. Como de exemplo as questões de números quebrados, com virgulas, no C#, precisa de uma atenção a mais na hora de querer mostrar esses resultados:



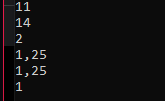
Podem se notar que até a variável N3, as operações são bem parecidas do que aprendemos na logica de programação do VisualG. Agora quando chega na parte do N4 pra frente, da pra notar que existem 3 tipos diferentes com a mesma operação, que seria a divisão.

Cada uma tem um jeito diferente na hora de mostrar o resultado, por exemplo no N4, pode notar declaramos a variável como Double, pois será um resultado de número quebrados, só que se apenas colocar a variável como Double e depois não colocar o casting depois do sinal de igual e antes da operação, o sistema vai entender que você quer um resultado com números inteiros, pois você colocou dois números inteiros para fazer a operação. Então para que ele mostre o resultado com as casas decimais você tem que colocar o casting, que seria o DOUBLE.

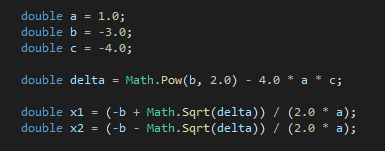
Outro exemplo de resultado é na variável N5, nela não foi colocado o CASTING DOUBLE, mas pode notar que o número 10, está com uma virgula e uma casa decimal valendo 0, isso mostra para o sistema que você quer que ele mostre o resultado com casas decimais, então para isso, basta colocar um ou todos os números com uma casa decimal.

Último exemplo de operação de divisão, é no N6, esse exemplo já diz tudo na hora que você presta atenção na declaração de variável, pode ver que declaramos como INT, então já da pra ter uma noção que o resultado vai ser inteiro e não com casas decimais, mesmo que colocamos um casting, que vai dar errado, e mesmo que colocamos casas decimais nos números, que também da errado, pois como declaramos que a variável vai ser inteira, o sistema consta um erro na hora que colocarmos casas decimais nos números após o sinal de igual.

Então os resultados ficaram assim:



Outra operação que podemos fazer é a equação do segundo grau, que é a de baskara.



Primeiro temos que declarar as variáveis das 3 letras que fazem parte da operação, que são as letras A, B e C. Declaramos elas em Double, pois tem divisão na formula da equação, e quando tem divisão, tem números quebrados.

Depois declaramos o delta, que seria b elevado a 2, menos 4, vezes A e vezes C. Essa declaração de variável delta, vai ser Double também, pelo mesmo motivo de cima. Só que na hora de colocarmos a expressão ELEVADO, podemos usar dentro da variável a função MATH.POW, que seria ela a responsável por colocar a potência em um número dentro do sistema do C#. Então ficaria MATH.POW(B, 2.0), declaramos a função, depois abrimos um parênteses, colocamos a letra que vai ser elevada, que vai receber a potencialização, que nesse caso é o B, virgula, e depois colocamos o valor da potência, que será 2.

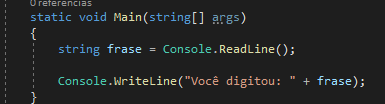
Depois de declarar o delta, vamos para as equações do x1 e x2, que sera um positivo e outro negativo. Declaramos double novamente com x1, depois entre parênteses, menos B, mais raiz quadrada de delta, que no C#, a função da raiz quadrada é MATH.SQRT entre parênteses o valor que vai ser feito a raiz quadrada, que no nosso caso é o delta, fecha parênteses, dividido, abre parênteses, por 2.0, vezes A, fecha parênteses. E para fazer o x2, basta apenas trocar depois a soma pela subtração.

  
E esses são os resultados de Delta, x1 e x2.

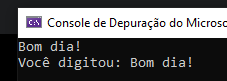
AULA 7

ENTRADA DE DADOS EM C#

Nessa aula, vamos falar sobre a função CONSOLE.READLINE(), que a função responsável por fazer com que o usuário escreva qualquer texto dentro do console, no caso, ele não precisa deixar uma palavra ou um texto pré-definido dentro do código fonte, ele pode deixar pra escrever na hora que executar o programa, isso é muito utilizado quando fazemos um programa para calcular um número com o outro, deixando a pessoa escolher o valor dos números em vez de colocar pré-definidos.



Como no exemplo a cima, declaramos uma variável chamada de frase, como STRING, e depois colocamos o comando console.readline para que o usuário escreva o que quiser dentro do console. Em seguida colocamos um comando para que ele mostre o que o usuário escreveu, com a frase concatenada a outra.

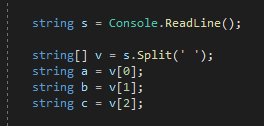


Esse é o resultado final.

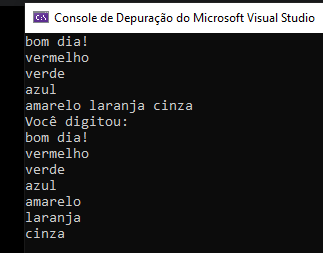
Outro exemplo de entrada de dados, e quando você tem mais de uma palavra na mesma linha, e quer armazenar cada uma delas dentro de variáveis diferentes, por exemplo, você escreveu AMARELO, LARANJA e CINZA, e você quer colocar cada uma delas dentro de uma variável diferente, então temos que fazer alguns passos.

Primeiro, declaramos a variável mestre, que vai ser a variável que vai receber a frase inteira, com todas as palavras de exemplo, STRING S = CONSOLE.READLINE(). Depois, declaramos a outra variável, mas com uma função de VETOR, igual na logica de programação do VISUALG, STRING[ ] V = S.SPLIT(‘ ‘); nessa função estamos abrindo uma variável com vetores chamada de V, que é uma função que vai armazenar vários dados, depois estamos dizendo que o que vai ter dentro dessa variável V vai ser o mesmo que vamos colocar na variável S, e a função SPLIT vai ser responsável em pegar cada informação, dividir essa informações e salvar, mas como ela faz isso? Pode notar que dentro de parênteses, depois da palavra SPLIT, tem ASPAS SIMPLES com espaço, quer dizer que a cada palavra e um espaço, ele salva cada palavra em diferentes lugares dentro do vetor.

Depois temos que declarar cada um desses espaços com palavras dentro de outras variáveis, então faremos assim:

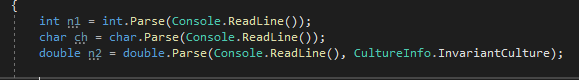


É simples de entender, basta pensar assim, o V[0], quer dizer que a posição 0, dentro do vetor V, vai ser colocada dentro da string A, então agora a string A, tem o mesmo valor que a posição 0 do vetor, e assim por diante, ate chegar na posição 2 e string C.

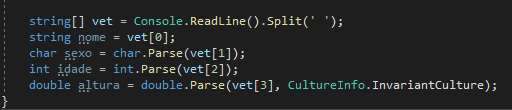


Esse vai ser o resultado final, podem notar que na última linha, antes da frase “VOCE DIGITOU”, as palavras estão todas na mesma linha, já na parte de baixo, que é o resultado final, elas estão uma em cada linha, mostrando que deu certo a questão de armazenamento de vetores, porque cada palavra tem agora uma posição em uma variável diferente.

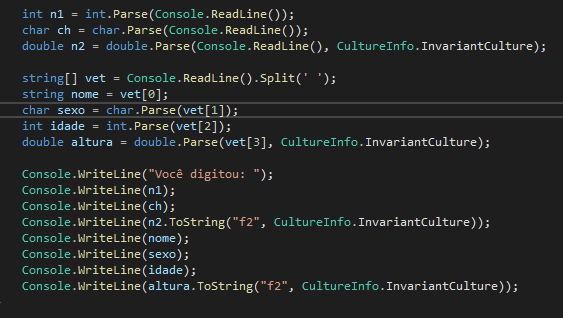
Outros exemplos de entrada de dados:



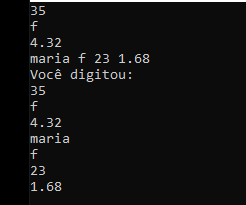
No exemplo a cima, declaramos três tipos de variáveis, uma INT, uma CHAR e uma DOUBLE. Mas antes temos que tomar um cuidado na hora de declarar o CONSOLE.READLINE, pois se colocarmos apenas o comando de console, ele vai dar um erro, pois console é para STRING, então, para que de certo e mostrar o resultado escrito do usuário, temos que converter a variável para a mesma que definimos, então para isso, pode ver que antes de colocar o console, colocamos a variável declarada + .PARSE, isso faz com que converta de STRING para a outra.



Mesma coisa na hora de colocarmos mais de 3 palavras, ou várias informações dentro de um vetor, nesse caso serve como a de cima, temos que colocar a declaração da variável junto com .PARSE, para converter na hora de mostrar. Pode notar também que da para colocar tudo em uma linha na hora de começar a criar um vetor, não precisa fazer mais uma declaração, podemos colocar um SPLIT dentro da variável que vai ser o responsável por separar e armazenar as palavras dentro deles.

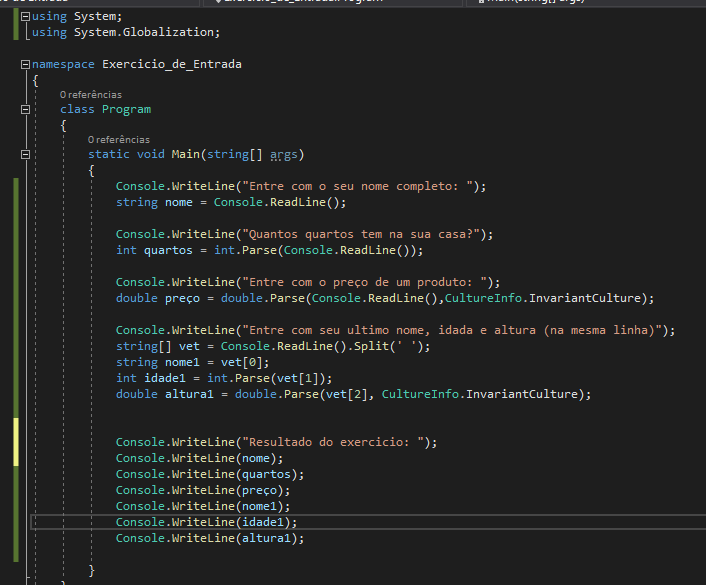


Então nosso código fonte vai ficar assim. Estruturado.

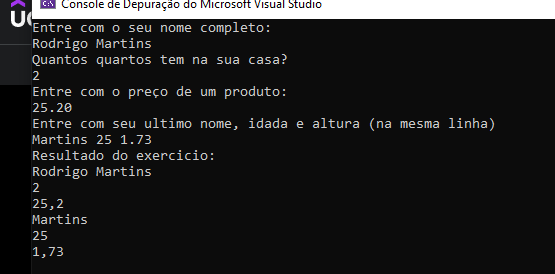


Mostrando esse resultado a cima.

Exercício do exemplo a cima



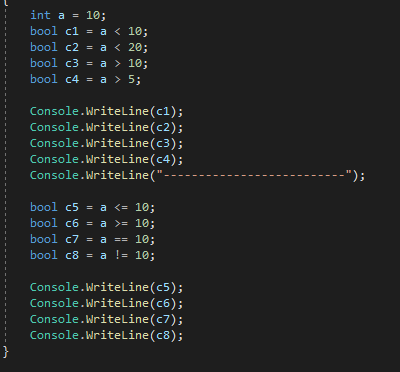
Resultado esperado:



AULA 8

OPERADORES COMPARATIVOS

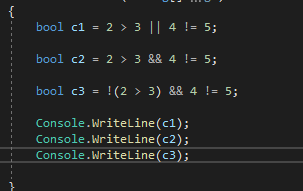
Essa etapa é bem simples, apenas uns comparativos entre numeros, texto, para ver se é falso ou verdadeiro.



Basicamente estamos comparando a variável A, que é inteira, com o valor 10, com algumas outras variáveis BOOL, que são responsáveis por passar essas informações comparativas.

AULA 9

OPERADORES LÓGICOS



Nesse tema, vamos ver as operações de comparação com operadores lógicos, no caso serve para comparar algumas condições com os operadores E, OU e NÃO. O operador OU, é o primeiro da linha, representado na variável C1. Ele está querendo dizer o seguinte, “2 maior que 3 OU 4 diferente de 5”. Então, o sistema vai mostrar no final, se é verdadeiro ou falso.

Nesse caso do operador OU, vai mostrar verdadeiro, pois ele utiliza o método de que se dentro da expressão, ou da condição tiver UM valor verdadeiro, ele considera verdadeiro. Então, se tiver FALSOxFALSO, ele vai considerar falso, se tiver FALSOxVERDADEIRO, ele considera verdadeiro, ele precisa apenas de uma resposta verdadeira, para considerar verdadeira.

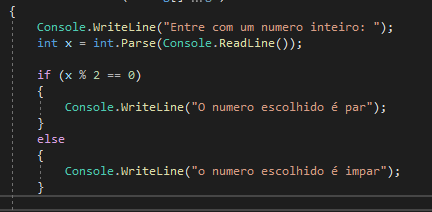
Já na variável C2, estamos com o operador E, representado pelo ‘&&’. Está escrito assim, “2 maior que 3 E 4 diferente de 5”. Nesse caso do operador E, tem que ter as duas expressões na condição como verdadeiro, para ele considerar verdadeiro, senão, ele considera falso tudo. Nesse exemplo da variável, vai ser FALSO, pois a parte que está falando 2 > 3, não é verdadeiro, e pelo motivo de ser operador E, ele considera tudo falso. Para considerar verdadeiro, tem que ter pelo menos os dois verdadeiros, senão, não tem como. Teria então que ser escrito que o 2 < 3, aí sim seria verdadeiro.

Na última variável, C3, está sendo utilizado o operador NÃO, representada pelo sinal ‘!’, na no começo da condição. Ele serve pra anular aquela expressão, por exemplo, na variável de cima, ele diz que 2 é maior que 3, e isso é falso, agora nessa condição nova, ele já diz de cara que NÃO, o 2 NÃO é maior que 3, fazendo com que a expressão toda seja verdadeira.

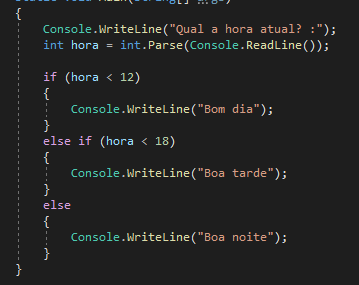
AULA 10

ESTRUTURA CONDICIONAL (IF -ELSE)

Estrutura IF e ELSE, é basicamente o SE e SENÃO da VISUALG, pois é ele que faz a questão de comparação para fazer uma condição ou outra. No caso no exemplo abaixo, que é um código para que o usuário coloque um número, e o sistema fala se esse numero é impar ou par.



Primeiro a gente pede para que o usuário digite um número, com o código CONSOLE.READLINE, salvamos um numero inteiro com a condição PARSE. Depois damos um IF, entre parênteses a conta que ele tem que fazer para chegar no resultado e mostrar se é par ou não, que é a conta de MOOD, que é o resto de uma divisão (%). Depois de declarar esse cálculo, abrimos chaves na linha de baixo ou na mesma linha, depende de como está configurado o seu código fonte, em seguida colocamos a primeira condição, que é se o resultado der 0, aí ele mostrara que o número é par, fecha chaves. Depois abrimos outras chaves na linha de baixo e colocamos o ELSE, que será o contrario da primeira condição, que se o resultado não for 0, então ele vai mostrar que o número é ímpar, pois todo número que for divido por dois e não der zero, é ímpar.

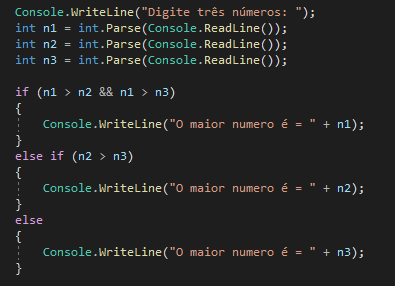


Ultimo exemplo da aula, para testar mais ainda.

AULA 11

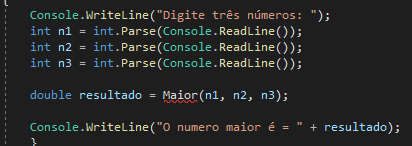
FUNÇÕES (SINTAXE)

Existem vários tipos da gente fazer um código dentro do C# para mostrar alguns resultados diferentes, por exemplo, fazer um código fonte, aonde ele mostra depois de pegar 3 números, qual deles é o maior. Poderíamos fazer isso da seguinte forma:



Pedimos para o usuário digitar 3 números, depois fazemos as condições se e senão.

Mas, podemos também montar um código fonte, aonde criamos essa estrutura como uma função única, para que quando for preciso, não ter que digitar todo esse passo a passo de novo, basta criar uma função nova, para que ela sempre seja chamada caso necessite de novo das informações dela.



Nesse caso a função ainda não existe, mas seria basicamente diminuir toda aquelas informações a cima, por esse simples comando, apenas colocando tudo aquilo, na variável MAIOR, que aí ele faz sozinho, depois declararmos, a função de achar o numero maior entre os 3 digitados. Então vamos lá:

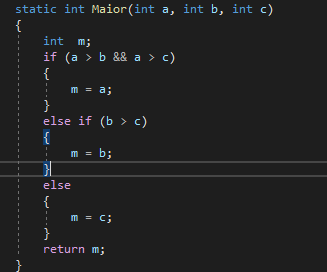
Primeira coisa, temos que abrir outra função principal, igual um código fonte do zero, com a função STATIC VOID. STATIC VOID, quer dizer que a função escrita, vai começar a partir daquele momento, mas nesse nosso caso de função, vai se chamar outra.

Então, quando formos colocar outra função dentro do código fonte, basta escrever STATIC, que vai ser a inicialização de uma função nova, e em seguida a palavra que vai ser a saída dessa função, no caso, é a entrada do valor da variável, qual o tipo de variável que essa função vai receber? No caso desse exemplo, vai ser números inteiros, então ficara assim, STATIC INT, em seguida o nome dessa função, que será MAIOR, então, STATIC INT MAIOR, e depois, entre parênteses, a quantidade de variável que vão ser alocadas dentro delas, STATIC INT MAIOR (INT A, INT B, INT C). Você pode colocar qualquer coisa no lugar das letras, o que não pode esquecer é de colocar em cada um, o tipo de variável, INT, DOUBLE, entre os outros que existem.

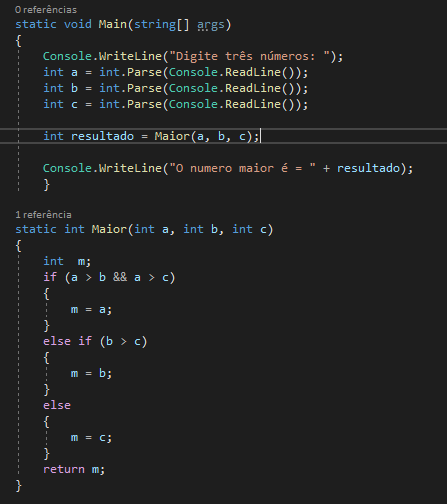
Depois de declarar o nome e colocar a variável, temos que abrir chaves, para que ele entenda que é dali pra frente que vamos digitar o código fonte essencial para a função funcionar, ficara assim:



Daqui pra frente começa a ficar um pouco mais fácil, o que temos que fazer agora, é digitar o código fonte da função, que é a mesmo código da primeira imagem do exemplo:



Declaramos primeiro uma variável INT chamada M, que vai ser a responsável em receber os valores. Depois digitamos os IF e os ELSES do processo da logica, como esta na imagem acima. O que temos que nos atentar é que, temos que colocar no final do código, a palavra RETURN com a variável que queremos retornar na frente, igual na ultima linha da imagem, RETURN M. Quer dizer que, assim que acabar a função, é para o sistema retorna o numero que deu como maior.



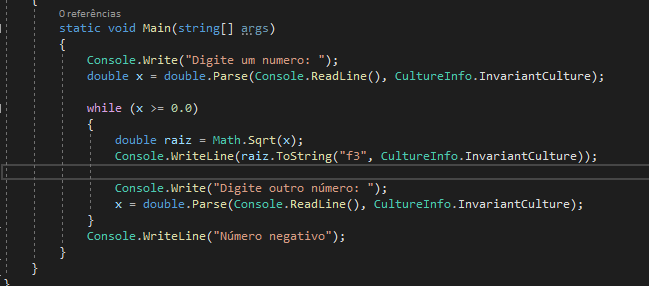
Então o código inteiro vai ficar assim, agora podemos chamar a função MAIOR, sempre que precisarmos, em vez de montar ela toda hora. Fica mais fácil se for um código fonte, aonde vamos teremos vários testes de números maior ou menor.

Uma curiosidade é, que o sistema já sabe aonde trocar as informações entre parênteses, por exemplo na declaração da função, colocamos as variáveis INT A, INT B e INT C. Na hora que chamamos essa função feita, na parte do código fonte principal, ele troca os valores pelas variáveis declaradas e preenchidas pelo usuário, então, como mostra na imagem, ele vai saber que o n1 é a letra A, o n2 é a letra B e o n3 é a letra C, por isso que quando a gente está criando uma função, podemos colocar o que quisermos dentro dos parênteses, só basta se atentar na hora de colocar as variáveis, e saber também que na hora de começar o código fonte, os números que o usuário colocar, tem que ser INT se estiver declarado como INT.

AULA 12

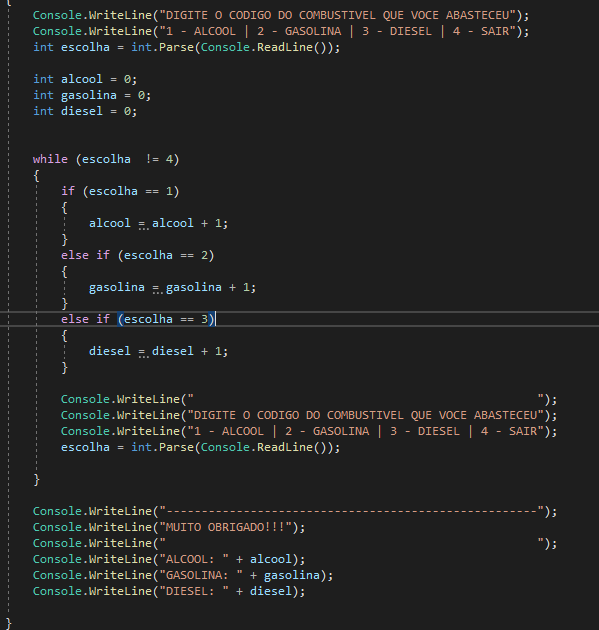
ESTRUTURA REPETITIVA WHILE (ENQUANTO)

É um código fonte aonde criamos uma estrutura que repita o código fonte, enquanto não fazer aquilo que foi declarado como NÃO FAZER. Por exemplo, vamos criar um sistema que peça para que o usuário digite um numero qualquer, assim, fazendo com que o sistema nos imprima o valor da raiz quadrada desse numero digitado. Logo em seguida, ele pediu para que o usuário digite mais um número, e mostre novamente a raiz desse outro número. Esse processo só pode parar, caso o usuário digite um número NEGATIVO, caso ao contrário, ele continua sempre, até o usuário finalizar:



Primeiro a gente declara uma variável, para receber o primeiro número, depois criamos a condição WHILE (ENQUANTO), dizendo que se X >= 0.0 REPITA, abrimos CHAVES como se fosse um IF, e colocamos a função de raiz quadrada em seguida, como MATH.SQRT( ). Em seguida pedimos para que o usuário digite novamente outro número, ainda dentro do WHILE. Fecha as chaves e em seguida, faça um CONSOLE WRITELINE com a informação “NUMERO NEGATIVO”, ele só vai parar e mostrar essa informação, quando o X FOR MENOR QUE ZERO, no caso, quando o usuário digitar um numero negativo.

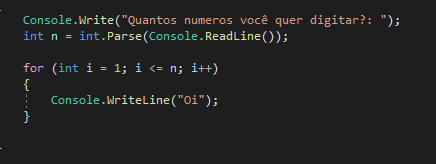
Exercicio de WHILE:



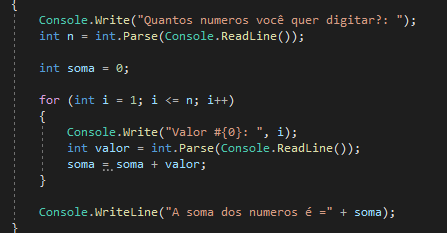
AULA 13

ESTRUTURA REPETITIVA FOR (PARA)

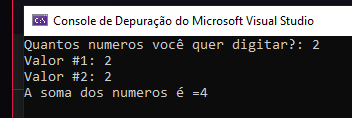
Estrutura FOR, serve bastante para quando você sabe a quantidade de repetições que vai ter dentro de um código fonte, quando você já tem definido quantas vezes aquele mesmo processo vai, ou, tem que ser repetido para um usuário. Por exemplo, vamos fazer um sistema que peça para o usuário digitar a quantidade de vezes que ele quer que apareça na tela:



Nesse exemplo de cima, estamos perguntando para o usuário a quantidade que ele quer digitar, depois lendo uma variável que vai guardar essa informação digitada. Em seguida abrimos uma função FOR, mas preste atenção, quando for abrir uma função FOR, nela tem 3 parâmetros de entrada para fazer rodar. Todas estão entre parênteses. O primeiro parâmetro para fazê-la rodar, é o inicio da função, que a gente denomina com uma variável, que no exemplo a cima, foi i, em seguida damos um valor para essa variável. Então, estamos dizendo que a função FOR, vai ser iniciada com a variável i valendo 1. Depois damos ponto e vírgula e declaramos que essa função só vai continuar enquanto a variável I, for menor ou igual a variável N, que foi declarada no começo da função. E pra finalizar os parâmetros, mostramos também para a função que, enquanto ela rodar, enquanto o i for menor que n, é para o sistema somar ou incrementar +1 na variável I, isso tudo enquanto o i for menor que N.



Mais um exemplo de como fazer uma função FOR. Nesse exemplo estamos pedindo para que o usuário digite a quantidade de números que ele quer, igual o de cima. Só que dessa vez, vamos mostrar no final o resultado da SOMA entre esses números. Primeiro temos que criar uma variável SOMA, antes de começar a função FOR, e essa variável tem que receber 0, pois ela vai iniciar zerada, só depois que vai começar a ser incrementada. Em seguida, digitamos a função FOR, como o exemplo da primeira foto, depois colamos para que o sistema mostre para o usuário a posição do numero que ele vai digitar, por exemplo, NUMERO 1, NUMERO 2, isso vai variar conforme a quantidade que ele escolher na variável N. Depois damos mais uma entrada de dados, com o console readline dentro da variável VALOR, para que o sistema armazene cada valor digitado. E por fim, fazemos com que a variável SOMA, que estava valendo zero, comece a valer ELA MESMA MAIS O VALOR QUE O USUARIO DIGITAR. Por fora das chaves, damos mais um CONSOLE WRITELINE, para mostrar a quantidade da soma dentro da variável SOMA, que ficou armazenando dados durante a execução da função.

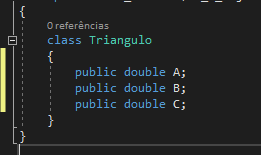


E esse é o resultado que tem que aparecer no final.

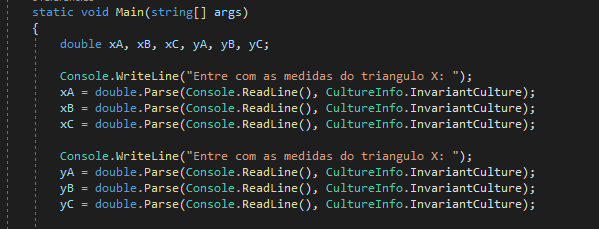
AULA 14

ORIENTAÇÃO A OBJETOS

Para começarmos a usar orientação a objeto, primeiro temos que ter o conceito de classes bem definidas. Classes seria você criar um objeto composto com varias posições ou uma, com funções ou apenas comandos únicos, para depois usar ele no meio de suas variáveis. Por exemplo, quando você cria classe, seja ela simples ou não, você acaba criando um TIPO DE VARIAVEL, que você pode usar ele para denominar ou falar que aquela variável vai receber o mesmo tipo daquela classe. Por exemplo:



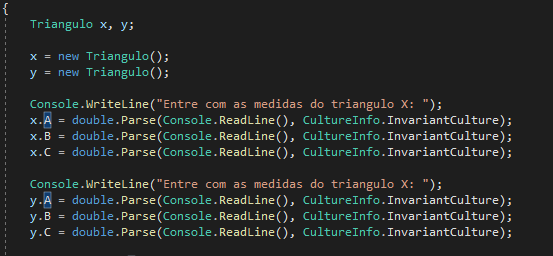
Aqui foi criada uma classe chamada TRIANGULO, que é composta por lados do tipo Double A, B e C.



Seria a mesma coisa se a gente fizer assim, como a imagem a cima. Pode notar que o código fonte fica ate mais limpo, quando usamos classes para definir uma variável. Nesses exemplos, estamos montando um triangulo, então para isso ser feito, de forma que não usamos a orientação a objeto, teríamos que criar 3 variáveis para colocar as informações do triangulo X, e 3 variáveis para colocar as informações do triangulo Y. Só que isso demora muito, e acaba ocupando muito espaço no código fonte, apenas para definir uma variável e criar um objeto por exemplo. Então para ser mais rápido, no C#, conseguimos definir todas essas informações, em uma única classe. Como mostra a primeira imagem. Definimos a classe inteira como TRIANGULO e com os lados A B C, em seguida, em vez de declarar variáveis com todos esses lados, vamos apenas declarar variáveis com a classe TRIANGULO, então vai ficar assim:

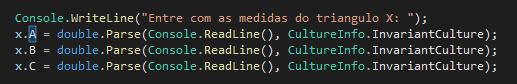


Mostrando que agora o X e o Y, vão ser triângulos, e automaticamente, vão ter 3 lados como um triângulo.



Notem que agora não existem varias variáveis que definem o triangulo, e apenas duas que fazem parte do triangulo inteiro. Então definimos duas variáveis, com todas os elementos deu triangulo. Não pode esquecer que, quando for definir elas, colocar sempre um NEW e o nome da variável, como mostra nas primeiras linhas da imagem a cima. Isso faz com que o sistema entenda que você está colocando X para receber uma nova variável, ou uma nova incrementação contendo as informações da classe TRIANGULO, e com o a variável Y é a mesma coisa, Y = NEW TRIANGULO();

Agora, para fazer com que o sistema saiba qual informação colocar em cada parte, basta fazer o seguinte:



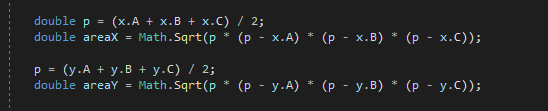
Nesse exemplo, estamos pedindo para o usuário colocar todas as medidas do triangulo X, então para fazer com que o sistema coloque de forma certa, dentro dos espaços definidos na CLASSE TRIANGULO, temos que usar primeiro o X que é a variável que estamos usando agora, junto com o ponto e depois o local que vamos colocar, como por exemplo, na primeira linha do console na imagem, X.A, isso quer dizer que estamos pedindo para que o sistema coloque, ou salve, as informações daquela linha dentro da coluna A na variável X, e assim por diante. Então só basta colocar a variável, o ponto e o local.

Lembre-se que essa mudança tem que ser feita em todo o código fonte, senão, ele não fara sentido. Pois tudo o que usar como variável X, vai ter diferença pela posição, ate mesmo na hora da formula, caso tiver.

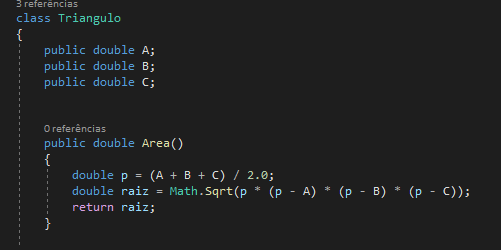
Outra informação importante, a classe é a definição do tipo, por exemplo, vai ser uma classe chamada triangulo, composta por lados, seja ela Double ou outras. Já objetos, são as instancias contidas dentro dessas classes, que no exemplo do triangulo, é o triangulo X e o Y, são denominadas como instancias, quando usa as informações e são declaradas com o tipo da classe. já os dados contidos dentro dos locais A,B e C, denominado no código fonte para salvar as informações que o usuário coloca, por exemplo, é chamada de ATRIBUTOS ou CAMPOS.

Conseguimos também, dentro das classes, colocar as funções ou as contas que são usadas para chegar no valor da área, igual no exemplo do triangulo.

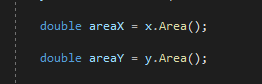
Em vez de ficar um código grande, com todas as contas escritas, podemos fazer assim:



Essa são as contas que precisamos fazer para achar a raiz e a área dos triângulos, mas como podemos observar, fica muito extenso, pois temos duas variáveis para analisar e dizermos qual o valor da área delas. Pois mesmo que seja apenas uma conta, ou duas, temos que fazê-las duas vezes, pois se trata de dois triângulos no exemplo. Então, para facilitar ainda mais, conseguimos colocar essa conta dentro da classe TRIANGULO, dizendo para o sistema que, quando executar as variáveis, sendo duas ou mais, ele apenas vai usar essa conta em apenas UM TRIANGULO, o que vai mudar, vai ser na criação de outras variáveis com a mesma classe, mudando apenas o nome. Em outras palavras, não vai ser necessários fazer DUAS CONTAS IGUAIS, apenas uma dentro da classe e utilizar ela dentro de várias variáveis.

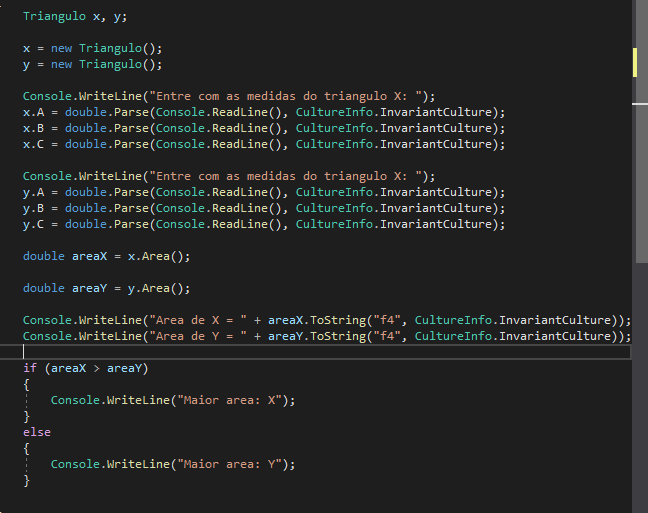


Esse imagem a cima representa o jeito que a conta vai ficar dentro da classe, lembrando que, se for usar algumas funções dentro do código fonte da classe, tem que colocar no começo dele o USING SYSTEM, para o sistema conseguir puxar as funções, caso ao contrário, dará um erro.

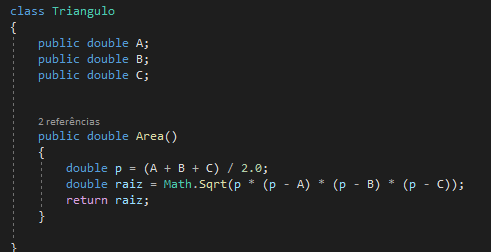


E é assim que ficara no código fonte principal, bem mais limpo e bem mais fácil de visualizar. Podem notar que usamos dentro da variável Double areaX, o nome do triangulo, que é a variável X, indicando que aquela expressão, ou função, vai ser usada no triangulo X que é a função de AREA, escrita no código fonte da classe, que se repete na variável Y, mudando apenas o nome para não dar erro.

O código final ficara assim:



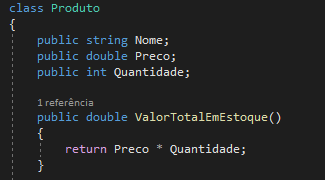
Com a classe assim:



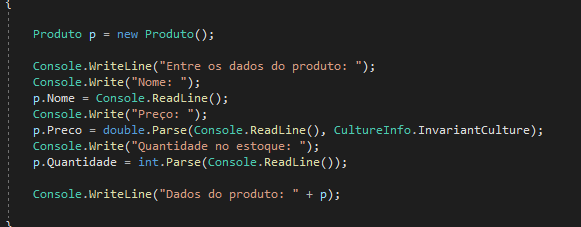
Outro exemplo agora.

Vamos elaborar um código aonde o problema é o seguinte, temos um estoque, queremos dar entrada nele de produtos, com quantidade e preço do mesmo, usando classes também. Mas tem uma coisa diferente, queremos que na hora que ele for mostrar essas informações, que ele nos mostre tudo, nome, valor unitário, quantidade e valor total.

Então vamos lá:

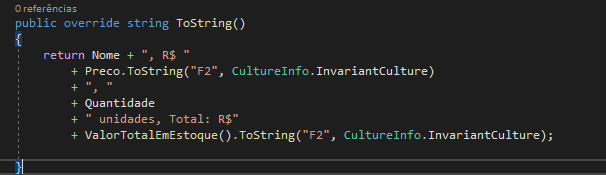


Primeiro definimos a classe, com os atributos e métodos iniciais. Nesse caso vai ser uma entrada de dados de um produto, com nome, preço e quantidade, e no final, queremos que ele nos mostre a quantidade total, que é um dos métodos que vai ser utilizado.

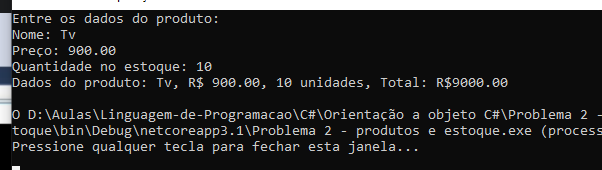


Depois, voltamos para o código fonte e damos inicio ao problema, definindo uma variável P, com a classe produto. Depois pedimos para o usuário digitar os atributos que tem dentro da classe produto, como nome, preço, do jeito que está na classe, lembrando sempre de colocar o nome da variável com a posição que vai ser preenchida na classe.

Depois de colocar os comandos, reparem que tem uma console no final, pedindo para que o sistema mostre a frase “dados dos produtos” concatenado com a letra P, que é a variável da nossa classe. Se dermos um F5 com essa expressão, o sistema vai nos mostrar apenas o nome da classe, que é produto, mas o que queremos aqui, é que ele nos mostre tudo que foi adicionado dentro da classe conforme cada campo definido na variável P, então pra isso, temos que utilizar uma função TOSTRING.



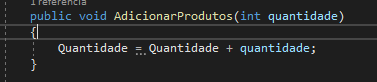
Depois de declararmos dentro da classe, os atributos e alguns métodos, podemos utilizar esse método do TOSTRING, para nos mostrar todas as informações quando colocarmos o nome da variável em um console. Primeiro colocamos como PUBLIC, depois damos o comando OVERRIDE, que está dizendo que essa função está vindo de outro lugar de dentro do sistema, que ela já existe, em seguida colocamos o tipo de retorno de dados, que vai ser uma STRING e colocamos o nome dela, como já existe, vai ser TOSTRING (), lembrando que isso se faz para converter e nos mostrar uma determinada informação em STRING, então tudo que tiver número, texto, vai ser convertida e mostrada como STRING, para não dar erro na hora da execução. Abrimos chaves para acrescentar os comandos e colocamos dentro dela um RETURN, para retornar todas as informações que vão ser acrescentadas. Depois começamos a concatenar cada informação desejada, nome, preço, quantidade, algumas palavras concatenadas entre aspas e no final, concatenamos um método que criamos dentro da classe, VALOR TOTAL EM ESTOQUE, ele vai mostrar a quantidade total de cada produto no final da execução.



Então se executarmos o programa, e acrescentar tudo certinho dentro dos campos, no final vai aparecer assim, como mostra na imagem a cima.

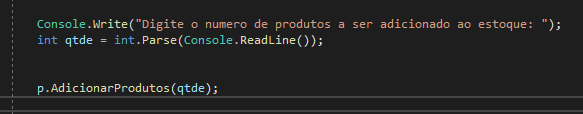
Agora vamos fazer com que o sistema peça para o usuário acrescentar mais informações, no caso, aumentar a quantidade do produto já cadastrado.

Primeira coisa, temos que colocar outro método dentro da nossa classe, que agora vai se chamar ADICIONAR PROTUDOS:



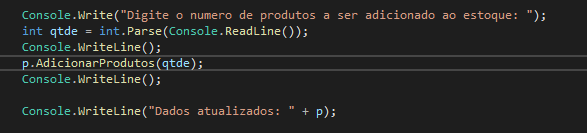
Como na imagem a cima. Pode reparar que na hora de criar o método, usamos a palavra VOID, isso que dizer que esse método não vai nos retornar um valor especifico depois de executar o programa, ele vai apenas fazer uma alteração dentro do campo quantidade, que está vinculado com os campos do produto. Pode reparar também, que dessa vez, entre parênteses tem um parâmetro de entrada, que no caso, vai ser o parâmetro que vamos usar para modificar o campo já existente. Isso acontece quando temos que colocar um campo que ainda não existe dentro da classe, por isso colocamos dentro dos parênteses. Então depois de fazer todos esses preenchimentos, abrimos chaves, e colocamos as variáveis ou campos que vão ser modificados ou alterados durante o processo, que no caso, é o campo de Quantidade que é um atributo do produto, mais a quantidade que é um parâmetro de entrada, que está dentro dos parênteses.

Depois de preencher e declarar cada coisa no código fonte da classe, colocamos essas informações dentro do código fonte principal:

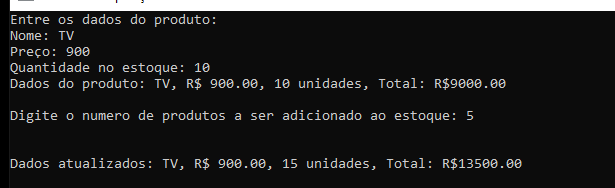


Como podem ver, estamos pedindo para o usuário digitar a quantidade que vai ser adicionada ao estoque. Logo em seguida, salvamos essa informação em uma variável nova, declarada como INT. Em seguida, colocamos a operação matemática que vai ser responsável em somar o estoque inicial com a quantidade que está sendo colocada agora, na função P. ADICIONAR PRODUTOS (QTDE). O QTDE entre parênteses, quer dizer que ele vai ser o novo parâmetro de entrada, igual no código fonte da classe, então o sistema vai usar ele para fazer a soma da quantidade existente mais a quantidade que o usuário colocou.

Em seguida, temos que pedir para o sistema imprimir na tela de novo, para ver se deu certo:

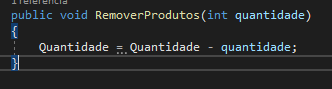


Então vai ficar assim, com uma nova chamada no final, para ver os dados atualizados.

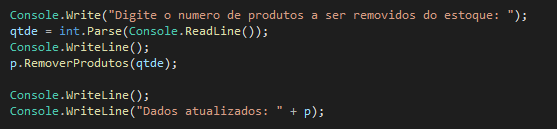


Podem ver que ele já somou a quantidade do estoque e o total também, pois a do total está vinculado com a quantidade, então se uma muda um, muda o outro automaticamente.

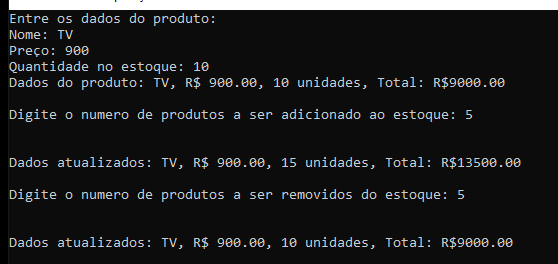
Agora para finalizar, vamos colocar outra conta para que o usuário retire algumas unidades do estoque, é bem parecido com o adicionar ao estoque:



Basta criar outro método na classe produto, agora com o nome de REMOVER PRODUTOS, com parâmetro de entrada como quantidade também, e na hora da conta, em vez de colocar como soma, basta apenas trocar o sinal e colocar como subtração, igual na imagem a cima.



Depois volte para o código fonte principal, peça para o usuário digitar a quantidade de produtos que vão ser retirados do estoque, utilize a mesma variável INT QTDE para salvar essa informação, e depois coloque o método que foi criado na classe para fazer a remoção e usar a variável QTDE como parâmetro de entrada. No final, faça a chamada para imprimir o resultado novamente, para ver se deu certo.



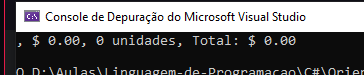
E esse é o final do problema.

AULA 15

CONSTRUTORES

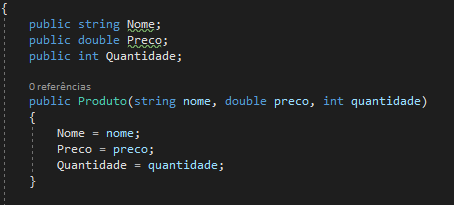
A utilização de construtores pode variar com suas responsabilidades, mas agora, vamos ver os construtores como uma função aonde o usuário ou programador é obrigado a iniciar o sistema, ou programa, com as informações iniciais digitas. Para isso, vamos utilizar o mesmo problema da aula anterior, que é do estoque.

Vamos dizer que é meio estranho começar o programa com um produto aonde ele não tem nome, nem preço e nem quantidade. Por algum motivo o problema está aparecendo assim:



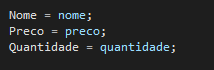
Isso quer dizer que não existe um construtor responsável para que o sistema impeça essa forma de exibir as informações do produto. Então, para evitar que o usuário imprima essas informações vazias, vamos criar um construtor para que o sistema mostre para o programador, que ele tem que colocar as informações necessárias para iniciar o programa:

Para isso, devemos criar esse construtor dentro da nossa classe, que nesse exemplo é a classe PRODUTO. Um construtor, por padrão, tem que ser criado depois dos atributos do objeto:

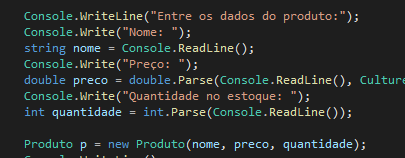


Podem notar que primeiro vem os atributos NOME, PRECO e QUANTDADE, e depois vem o construtor. O construtor tem que estar com o mesmo nome da classe, para ele identificar que vai se tratar da mesma coisa, do mesmo codigo, que aquele construtor está vinculado a classe. Entao para criar ele, primeiro temos que colocar como PUBLIC, para ele ser utilzado em outros lugares do codigo, como o mesmo codigo fonte principal, depois, colocamos o nome da nossa classe, PRODUTO, e depois colocamos entre parenteses, os parametros de entrada do nosso construtor. Em outras palavras, os parametros de entrada vao ser o mesmo atributos do produto, mas lembre-se de sempre colocar os tipos desses atributos juntos aos nomes deles. Entao dentro do parenteses vai ficar assim STRING NOME, DOUBLE PRECO E INT QUANTIDADE. Lembre-se tambem que, quando estamos dando a informação dos parametros de entrada, temos que colocar os nomes dos atributos com letra minusculas, por padrao do C#.

Em seguida, abrimos chaves e começamos a dar inicio ao nosso construtor. Essa parte é bem simples, apenas temos que relacionar um parametro de entrada, que esta dentro dos parenteses, com os atributos do produto, lembrando que nessa parte, os atributos, com a primeira letra maiscula, vem primeiro recebendo os parametros de entrada em minusculos:



Depois de finalizar o construtor, voltamos para o código fonte para notar algumas mudanças. Como criamos ele com parâmetros de entrada para nome, preço e quantidade, não é mais necessário colocar a nossa classe antes de todo o código fonte, e sim, temos que colocar depois que o usuário digitar as informações. Vai ficar assim:

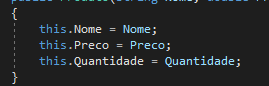


Antes de criarmos o construtor, a nossa classe vinha primeiro que tudo, para darmos os nomes das variáveis junto com a classe, então, antes a variável STRING NOME era P. NOME e assim por diante. Como já mostramos para o sistema, dentro da classe, que o STRING NOME, vai ser a mesma coisa que P. NOME, então não precisamos fazer isso novamente no código fonte, apenas relacionar igual no código da classe, na parte do construtor, entre parênteses. Pois fazendo isso, o sistema sabe quem é quem. Então primeiro o usuário digita uma informação dentro de uma variavel auxiliar, e depois essa variável é trocada para a outra variavel que está dentro do construtor. Então STRING NOME, é apenas uma variavel temporária que vai ser trocada pela variavel NOME, na hora de mostrar o resultado. Pode notar que essas variáveis principais, estão dentro da classe PRODUTO P.

AULA 16

PALAVRA THIS

Serve para diferenciar uma variavel para um parâmetro de entrada. Por exemplo na hora de criar um construtor de 3 parâmetros, na linguagem C#, na hora de colocar os parâmetros de entrada entre parênteses, você precisa iniciar as palavras lá dentro com letras minúsculas, caso você esqueça ou queira colocar com letra maiúsculas, você precisa acrescentar a palavra THIS na hora de vincular uma coisa na outra, para que o sistema entenda e defina quem é quem:

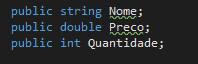


Primeiro é o campo dos atributos e depois o campo dos parâmetros de entrada, então se tiver THIS, o sistema entende que é o atributo do objeto, se não tiver o THIS, ele vai entender que é o parâmetro de entrada.

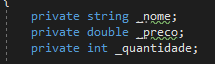
AULA 17

ENCAPSULAMENTO

Seria basicamente você proteger um atributo para que o usuário do sistema não consiga alterar ele na hora de executar ou mexer no código fonte. Por exemplo:

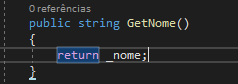


O atributo quantidade, ele esta como PUBLIC, isso quer dizer que qualquer um pode ir no código fonte, e alterar o valor dele, pois até o momento ele está valendo zero. Então quando um atributo se encontra como PUBLIC, você pode alterar ele dentro do código sem nenhuma restrição. Mas se caso você queira mudar isso, que ele apenas altere o valor desse atributo, na hora de executar uma função para isso, basta apenas colocar PRIVATE no lugar do PUBLIC. Mas atenção, a partir do momento que fizer isso, os nomes dos atributos que estão com PRIVATE, vão ter que começar por padrão com um sinal de \_ na frente e com a primeira letra em minúsculo:

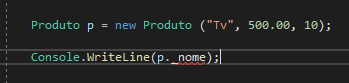


Depois disso, tem que trocar todos os nomes dentro da classe, que contem esses atributos, tanto nos métodos como nas funções, para dar certo depois no código fonte.

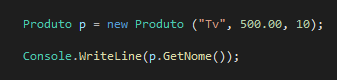
Agora, na hora de colocar esses atributos no código fonte, não vai ter como, pois o programador que está utilizando colocou eles como privados. Então para conseguir puxar as informações desses atributos no código fonte, terá que usar o método GET. Para isso, basta criar um método com as seguintes informações:



A palavra GET, significa que você quer puxar uma informação dentro de uma variavel ou atributo, então, como de exemplo, estamos fazendo com o atributo NOME, que no caso, se a gente tentar puxar para o código fonte com o nome atribuído a ele na classe, não vai dar certo. E é aí que o método GET entra. Ele serve para “substituir” e mostrar a informação que tem dentro daquele atributo, mas, não deixa que o usuário mude essa informação.

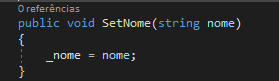


Pode notar que na hora de puxar as informações, dentro do código fonte no projeto principal, ele mostra o atributo com um erro, dizendo que por causa da segurança imposta no atributo, ele não deixa você pegar ou fazer qualquer alteração das informações dele. Entao para isso, você tem que dar um GET no código da classe, criar um método, fazendo com que ele mostre a informação contida dentro desse atributo que agora esta privado.



Pronto, agora ele deixa você usar as informações contidas dentro do atributo \_nome, com o método GETNOME, pois você vinculou uma coisa a outra.

Agora se quiser alterar o mesmo atributo, com outras informações, com ele ainda em método PRIVADO, basta usar o método SET, que significa, alterar um atributo pelo código fonte, mesmo que ele esteja privado:



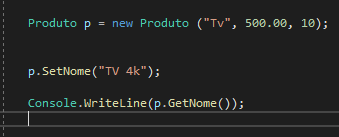
Primeiro colamos como VOID, pois ele não vai nos retornar nenhum resultado diferente ao executar, e sim apenas alterar o que já estava dentro do atributo, depois, damos o nome do método de SETNOME, entre parênteses o tipo do parâmetro de entrada usado para a alteração do atributo e o nome do parâmetro de entrada, fecha parênteses, abre chaves e coloca o atributo principal, que é o atributo que esta como PRIVADO, \_nome, recebendo o parâmetro de entrada usado na criação desse SET, desse método.

Agora vamos tentar fazer a modificação desse atributo pelo código:



Se colocarmos assim, vamos notar que não vai dar certo, pois estamos tentando utilizar um atributo que está privado, igual no exemplo acima, que era no GET.

Para que ele faça a alteração, temos que digitar assim:



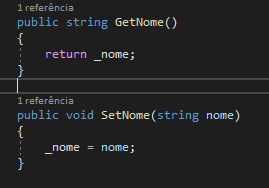
Com o nome do método utilizado no SET, entre parênteses a nova frase ou palavra que vai conter dentro do atributo privado \_nome.

AULA 18

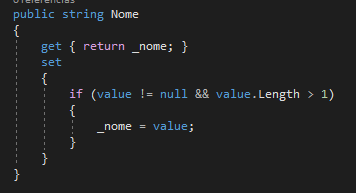
PROPERTIES

Incrementação de encapsulamento.

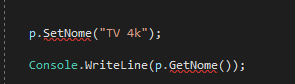
No exemplo da aula anterior, aprendemos que podemos fazer com que nossos atributos sejam privados, fazendo com que o usuário não consiga trocar ou imprimir aquelas informações contidas dentro de cada campo dos atributos, a menos que se utilize o método GET e SET. Mas na aula anterior, aprendemos de forma manual de como fazer cada um, aprendemos o conceito de cada e como utilizar dentro do código fonte principal. Nas propriedades, vamos fazer a mesma coisa, mas, de forma mais estruturada e limpa, para que o sistema e código não fique com ar de bagunça:



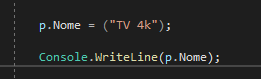
Então em vez de usarmos os dois exemplos acima, GET e SET, um separado do outro, podemos unir esses dois métodos em um apenas, fazendo as seguintes alterações:



Primeiro criamos um método publico de tipo STRING, com o nome NOME, abrimos chaves, e dentro delas colocamos o método GET, depois abrimos outras chaves na frente do GET, e dentro dessas chaves colocamos que o GET tem que retornar as informações do ATRIBUTO PRIVADO \_NOME. Sem seguida, na linha de baixo, colocamos o método SET, e abrimos outra chave, e dentro dessa chave, colocamos um IF, essa condição IF, vai falar para usuário que só ser aceita a troca ou alteração do atributo NOME, se as informações colocadas forem diferentes de 0 e que contenham caracteres maior que 1. Então a palavra VALUE significa que ele vai pegar o nome de parâmetro de entrada, que é a informação que o usuário vai colocar para trocar as informações e colocar, alterar, dentro do atributo \_NOME. Depois entre chaves novamente, se toda aquela condição for verdadeira, ele faz com que o atributo \_NOME receba o valor que foi colocado no parâmetro de entrada.



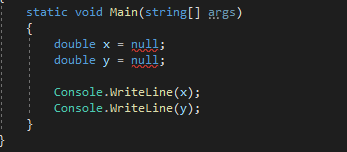
Agora no código principal, como mostra na imagem a cima, não precisa colocar com GET na hora de buscar as informações do atributo, só precisa colocar o nome da propriedade com os métodos já dentro delas, que no caso é o nome p.NOME, que dentro dele já tem os métodos GET e SET já preá definidos.



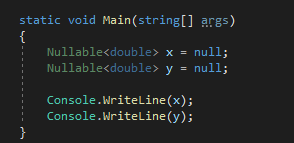
AULA 19

NULLABE

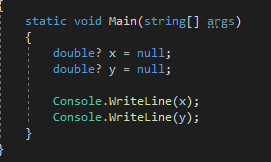
Quando você quer fazer um campo tipo número, ou tipo valores, como um campo opcional, no caso, para que o usuário do sistema coloque se ele achar necessário. Mas lembre-se, por padrão o sistema do C# não deixa isso acontecer, se você tentar declarar uma variável, do tipo valor, seja INT ou DOUBLE, ele vai dar um erro:



Para que você consiga, com que a variável seja opcional para o usuário preencher, basta fazer da seguinte forma:



Ou também:



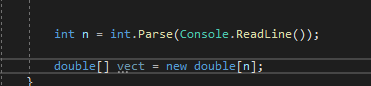
Tanto no exemplo numero 1 como no número 2, dar certo, basta você escolher o mais fácil para fazer, isso vai fazer com que o preenchimento da variável seja opcional pelo usuário e vai fazer com que o sistema aceite uma variável tipo valor receber um resultado nulo, ou um resultado que seja igual a nada.

AULA 20

VETORES

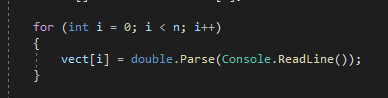
Vetores são mais utilizados não logica de programação, igual na linguagem do VISUAG. São resumidamente, uma caixa que você abre com varias campos de uma vez só, unidimensional e alinhado. Voce pode colocar a quantidade que quiser dentro um vetor, basta apenas depois, alocar dentro de cada posição, a sua informação correta. Lembrando que um vetor sempre começa com a posição 0, independentemente do tamanho dele.

Para inicializar, ou criar um vetor, basta fazer o seguinte comando no C#:



Esse comando esta querendo dizer um seguinte, abra um vetor, lembrando que para iniciar ele, ou criar, tem que colocar junto com o tipo do vetor, o abre colchetes, para o sistema saber que o que vai ser criado é um vetor e não uma variável comum. Depois, de o nome do seu vetor, nesse caso acima é VECT, igual, new double, pois tem que mostrar que vai ser um novo vetor do mesmo tipo da variável, e entre colchetes novamente, você coloca a quantidade de campos que ele vai ter, nesse caso, como a gente declarou que o usuário ia decidir, na variável N, colocamos entre colchetes a variável N.

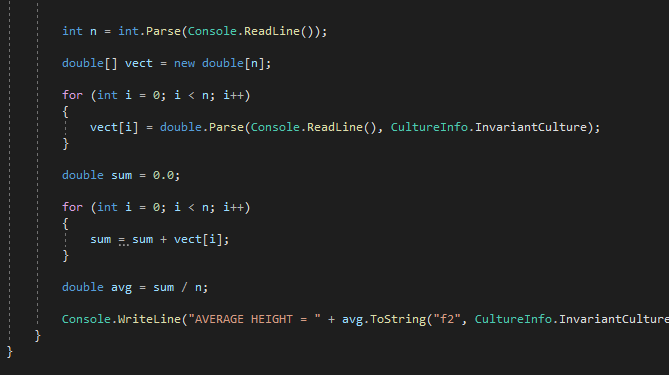
Agora, para acessar e colocar as informações em cada campo certo com um contador automático, temos que utilizar a função FOR. Ela vai ser responsável para acrescentar ou criar um contador para que ele coloque as informações dentro de cada campo do vector:



Basicamente estamos dizendo que, FAÇA COM QUE A VARIAVEL I INICIE COM O VALOR ZERO, E ENQUANTO ELA FOR MENOR QUE N, SALVE O VALOR DE N DENTRO DA POSIÇÃO I, E ASSIM, ATE DAR O LIMITE DE N.

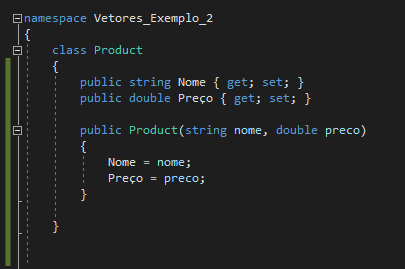
Depois temos que fazer com que o sistema some todas as informações de todos os campos dos vetores, para por fim, conseguir tirar a média.

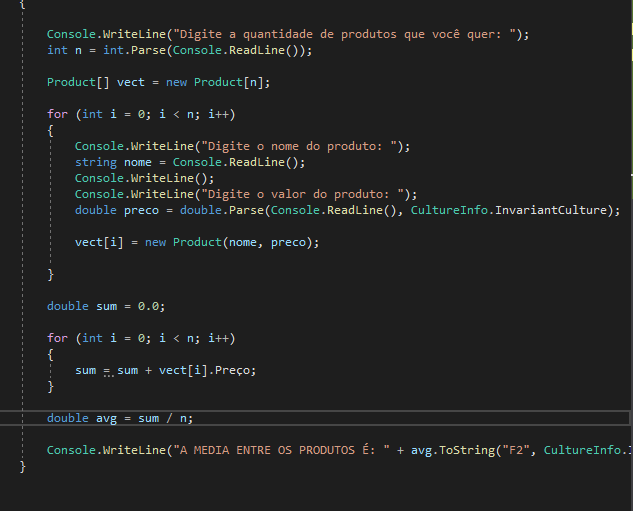
Então basicamente vai ser criado outro contador com a função FOR, para que ele percorra dentro do vetor já criado, só que agora, somando as informações que o usuário digitou:



Primeiro declaramos outra variável, que vai ser responsável em pegar todas informações e armazenar dentro dela, lembrando que ela tem que iniciar com zero. Depois, damos inicio a função FOR, fazendo com que o I INICIE COM 0 NOVAMENTE, E ENQUANTO ELE FOR MENOR QUE N, FAÇA ELE RECEBER SEMPRE MAIS 1. DEPOIS, FAÇA COM QUE A VARIAVEL SUM, QUE ESTÁ VALENDO ZERO, SOME O VALOR DELA COM O A INFORMAÇÃO QUE ESTÁ DENTRO DO CAMPO I, NO VETOR VECT. EM SEGUIDA, NA VARIAVEL AVG, FAÇA COM QUE A VARIAVEL SUM SEJA DIVIDIDA PELO TANTO DE POSIÇÃO EXISTENTE NO VETOR VECT.

Outro exemplo de vetores, agora utilizando ele para salvar mais uma informação, com classes feitas e construtores feitos:

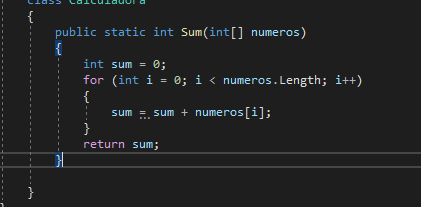




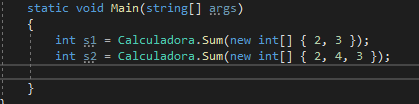
AULA 21

FUNÇÃO PARAMNS

Uma função que mostra para o sistema que você deseja colocar uma quantidade variável de informação. No caso de um vetor, você consegue colocar uma quantidade ilimitada de números, por exemplo. Em vez de fazer uma classe com um vetor e uma variável estática, aonde você inicia o vetor com N posições, você pode dizer para o compilador que você quer iniciar ele com essas N posições, mas utilizando apenas o comando PARAMS.

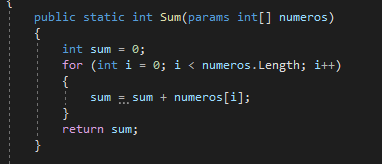


Nesse exemplo a cima foi criada uma classe chamada CALCULADORA. Aonde utilizamos um vetor dentro de uma variável estática. No caso o vetor se chama números do tipo INT. Depois iniciamos a variável estática SUM valendo 0. Em seguida, iniciamos a alimentação do vetor, sendo que a variável I recebe 0 de começo, enquanto ela for menor que a variável números, faça ela receber sempre o I mais 1. Depois dentro de chaves, fazemos o calculo de soma da variável SUM com o vetor números, recebendo os valores de cada posição. No final, retornamos a variável SUM.

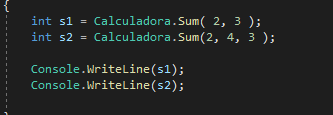


Para testarmos e vê se dar certo o calculo no programa, basta criar uma variável, instanciar a classe Calculadora que já está com um membro estático chamado SUM, que dentro desse membro estático, é aonde vamos colocar um novo vetor, só que com os parâmetros de entrada declarados entre chaves, nesse caso, os números nas duas linhas. Esse modo deixa o compilador um pouco cheio de informação, tendo que ficar escrevendo sempre uma variável nova, caso queira fazer uma soma com mais de um número.

Agora, existe uma função dentro do C# que faz com que o compilador aceite qualquer limite de variável, ou limite de números dentro de um vetor:



Basta colocar a palavra PARAMNS antes de declarar o vetor.

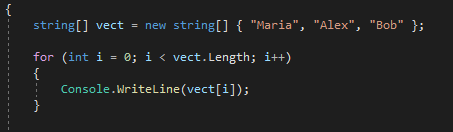


Fazendo isso, você não precisa instanciar um novo vetor sempre que for fazer um cálculo, basta apenas colocar os valores como parâmetros de entrada.

AULA 22

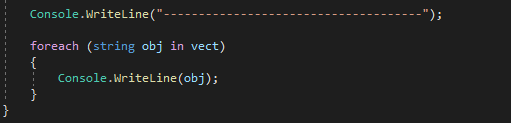
SINTAXE FOREACH

É função existe basicamente para imprimir algumas informações de dentro dos vetores de forma mais simples. Utilizando essa função, não precisa fazer um código tao grande fazendo com que ele incremente de um em um, dentro uma variavel, as informações contida dentro de um vetor. Seria basicamente assim:

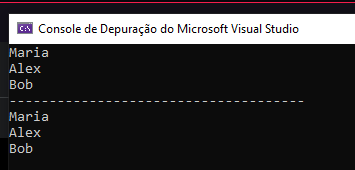


Nesse exemplo acima, estamos utilizando o sistema que aprendemos nas ultimas aulas. Fizemos uma variável STRING[ ], que vai ser um vetor, depois declaramos um NEW VETOR STRING, com as informações que vai ter dentro dele, entre chaves. Em seguida criamos um contador para que ele armazene as informações desse vetor dentro de uma variável chamada i. Então, faça com que a variável i seja iniciada com o valor zero, e enquanto ela for menor que o tamanho do vetor, faça ela receber todas as informações dele e salvar em si mesmo. Depois, mostre no console o as informações contidas do vetor dentro da variável i.

Esse seria um método para se fazer essa função, agora vamos utilizar a função FOREACH.



Notem que até o código fica mais curto e mais limpo. Então basicamente estamos dizendo assim para o sistema, para cada STRING OBJ (OBJ é uma variavel temporária, como se fosse a variável i que utilizamos dentro do exemplo do contador), contida dentro do vetor, imprima os valores de OBJ. Fica mais fácil fazer assim.

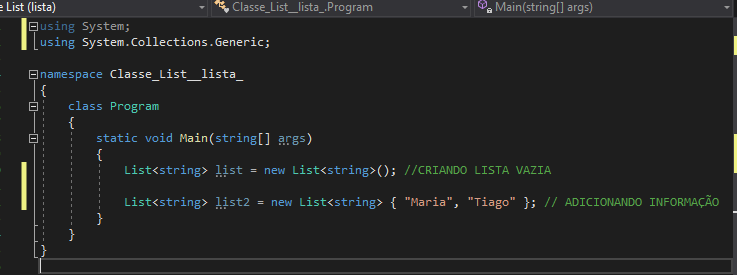


AULA 23

LISTAS

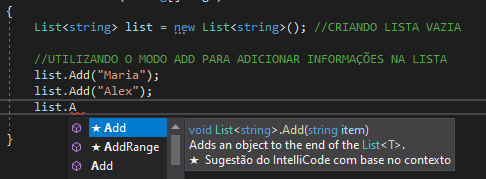
As Listas são estrutura de dados parecidas com as dos vetores, homogenias, ordenadas, mas a diferença é que nos vetores, você tem que iniciar eles, já as listas, são iniciadas vazias, os elementos são alocados sob demanda, conforme o sistema vai rodando o código fonte.

Para colocar ela dentro de um código fonte, ela tem que ser chamada dentro de uma namespace, que nesse caso é a USING SYSTEM.COLLECTIONS. GENERIC, fazendo essa colocação de namespace no começo do código, a classe LIST acaba aparecendo.

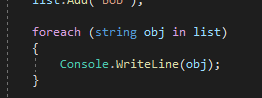
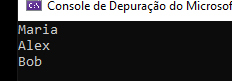


Então basicamente para criar, instanciar e adicionar as informações dentro de uma lista, basta seguir como o exemplo a cima. Primeiro você coloca a classe LIST, dentro de maior e menor você tem que colocar o tipo dessa lista, no caso do exemplo, vai ser uma lista STRING. Depois você instancia essa lista, usando o NEW LIST<STRING> (), como uma classe normal, e em seguida você decide se já vai acrescentar as informações nela, ou não.

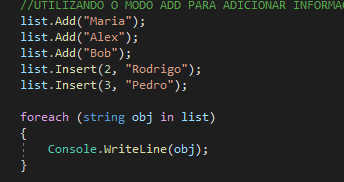
Um dos modos para acrescentar as informações dentro da lista, a gente viu no exemplo a cima, que é quando a gente cria e já adiciona entre chaves. Mas existe alguns modos para isso com a lista, como no exemplo a abaixo, utilizando a função ADD.



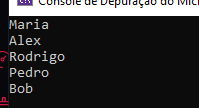
Basta então colocar a classe LIST.ADD, e depois entre parênteses, colocar a informação que desejar, que no caso do exemplo, foi uma STRING, então, tem que colocar entre aspas duplas. Depois para ver se deu certo, basta utilizar o comando FOREACH, que aprendemos na aula anterior:



Tem também a função INSERT. A diferença do INSERT para o ADD, é que com ele, você consegue colocar na posição que preferir, diferente do ADD, que coloca na posição de sequencia pré definida.



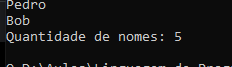
Basta digitar LIST.INSERT, abre parênteses, e dentro dos parênteses muda a questão de colocar junto com a informação, o local que essa informação vai ficar. Então com mostra no exemplo, primeiro colocamos o numero da posição e depois a informação.



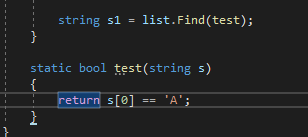
Lembrando que as contagens de posição começam do zero em diante.

Dentro da lista, conseguimos também saber o tamanho dela com o código LIST.COUNT, basta apenas chamar ele junto com um console, sem ter que colocar código fonte grande:



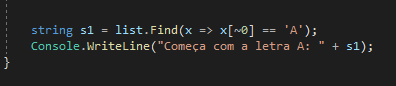


Agora outra função que existe na lista, é a função FIND. Ela é responsável em trazer algum tipo de informação usando predicado, é que uma função dentro da função FIND. Por exemplo, desses nomes registrados, vamos procurar o primeiro que aparece que começa com a letra A:



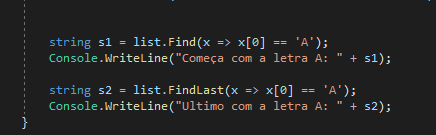
Esse é um exemplo da função FIND, usando uma função criada do zero dentro do código fonte. Então primeiro criamos uma função, aonde declaramos que essa função vai ser do tipo BOOL, chamada TEST, aonde ela vai receber como parâmetro de entrada, uma variável STRING S, aonde ela vai retornar na posição 0 da nossa lista, o nome que começar com a letra A. depois criamos uma variável de STRING chamada S1 recebendo a função LIST.FIND recebendo de parâmetro de entrada a função que criamos TEST. Então isso serve para mostrar que a função FIND, também pode receber funções criadas dentro do código fonte.

Mas podemos fazer dessa forma também, com a expressão LAMBDA.



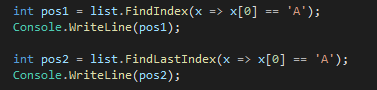
Que tem a mesma finalidade da função test. Então estou dizendo que, dado a variável X eu quero que ele mostre essa variável na posição X, caso ela comece com a letra A.

Tanto como começar, existe também a função para pegar a última informação, que seria a função LIST.FINDLAST. Pelo nome já identificamos que ela vai nos trazer, dentro do parâmetro de entrada, a ultima informação.



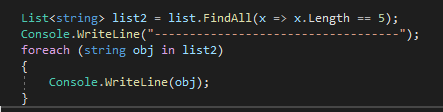


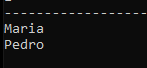
Outra função dentro da FIND, é saber a POSIÇÃO dentro da lista. Vamos fazer duas funções, para achar posição da primeira informação aonde ela começa com a letra A e a ultima informação que começa com a letra A, lembrando que essas duas procuraram e mostram posição e não as informações completas, então elas vão ser INT.



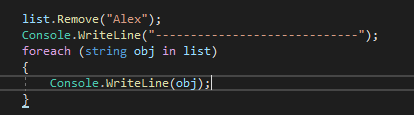
Basta usar a LIST.FINDINDEX e LIST.FINDLASTINDEX.

Funções para filtrar a lista. Buscar como exemplo, os nomes que tem apenas 5 caracteres, para isso, temos que criar outra lista para que ela traga as informações da outra lista já criada:

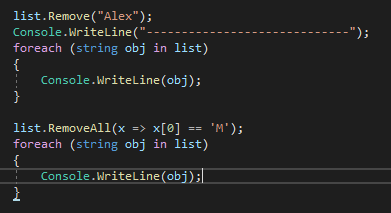


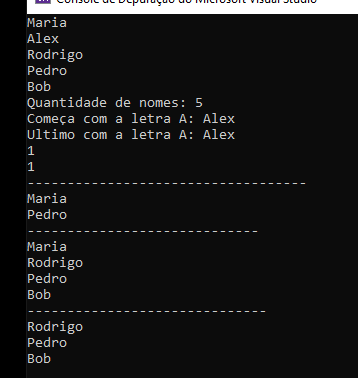


Para remover itens da lista, basta apenas colocar o código LIST.REMOVE e entre parênteses a informação que gostaria de tirar:



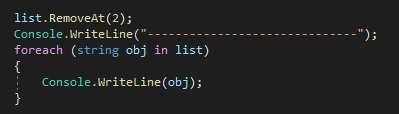
Temos também a função REMOVEALL, que remove uma informação determinando um predicado, no caso, vamos pedir para o sistema remover quem começa com a letra M:





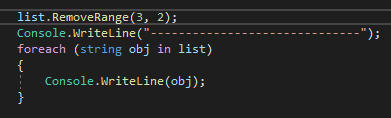
Pode notar que ele removeu o ALEX como está no LIST.REMOVE e ele removeu a MARIA, pois na FIND.REMOVEALL, colocamos para que ele retirasse todos que começassem com a letra M.

Conseguimos também remover pela posição dentro da lista, qualquer informação, com o comando REMOVEAT:



Basta colocar, depois de declarar a função, entre parêntese, a posição que quer retirar.

Pra finalizar, temos a última função de remoção de lista, que é a função LIST.REMOVERANGE:



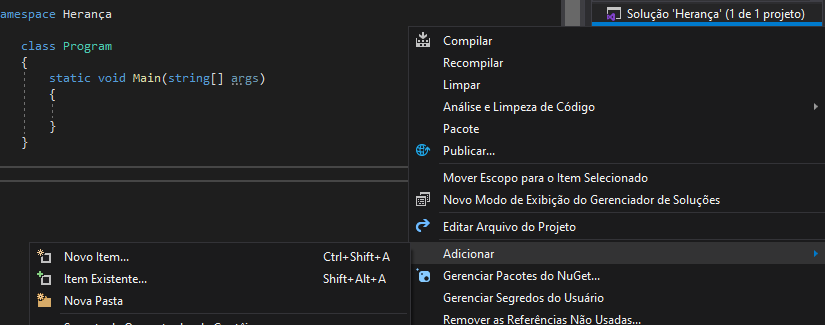
No caso ela fala como parâmetro de entrada o seguinte, quero remover a partir da posição X, N informações, que no caso do exemplo a cima, está dizendo que é para remover a partida da posição 3, 2 informações.

AULA 24

HERANÇA E POLIMORFISMO

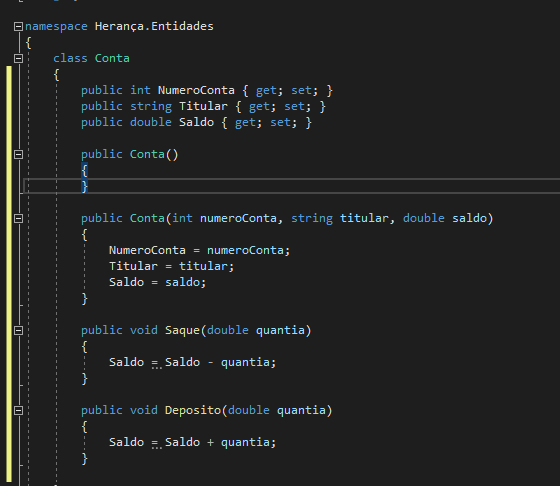
Herança seria nada mais do que pegar as informações de uma classe já criada e usar elas dentro de uma outra classe, otimizando o tempo na hora de criar outra classe parecida, o que pode fazer de diferente, é acrescentar outros atributos ou outros métodos. Fazendo isso, a classe que serve de padrão para utilização dos atributos e métodos, acaba passando a ser a SuperClasse, e a classe que recebe as informações, acaba passando a ser a SubClasse.

Para criar uma herança entre uma classe e uma subclasse, basta apenas fazer os seguintes passos, vamos supor que queremos fazer duas contas de um banco, uma conta para o titular e uma conta aonde o banco vai ter acesso.

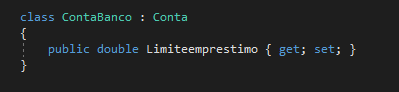
Primeiro de tudo, para criar uma herança, temos que criar uma nova pasta dentro projeto: 

Para isso, basta clicar com o botão direito no nome do projeto, ir em adicionar e procurar o item NOVA PASTA.

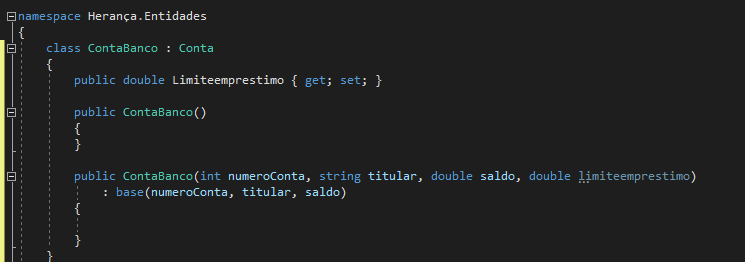
Depois o processo é o mesmo, clicar nessa pasta com o botão direito, adicionar e criar as classes necessárias, lembrando que uma vai ser a conta do cliente e outra a conta do banco.



Basicamente é igual as demais classe, não muda nada, lembre-se que essa vai ser a Superclasse, a classe principal utilizada para alimentar a subclasse. Podem notar que nela não mudou nada, criamos os atributos, os construtores e os métodos, agora vamos criar a nossa subclasse:

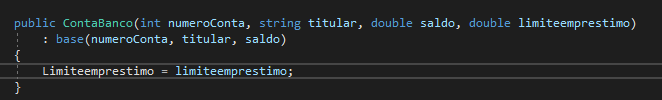


Para fazer com que o sistema entenda que você quer utilizar essa classe como uma subclasse de outra, utilizar as informações já criadas em outra classe, basta apenas colocar, depois do nome da sua subclasse nova, o ponto : e na frente o nome da Superclasse, que na imagem mostra como CLASSE CONTABANCO : CONTA. Pronto, apenas esse detalhe para que o sistema passe a entender que você quer que ele utilize todas as informações da classe principal.



Na imagem a cima, podemos notar que já tem os construtores padrão, que são aqueles em branco, sem parâmetro de entrada, para o usuário conseguir iniciar o sistema com eles vazios e um novo tipo de construtor, que é o construtor com as informações migradas da SuperClasse.

Se você tentar colocar as informações dentro dos construtores direto pelo atalho, pela função pronta do sistema, ele só vai mostrar o primeiro atributo vinculado a essa classe, ele não puxa as informações da outra classe, mesmo estando vinculadas. Para utilizar os atributos existentes na outra classe, você tem que colocar ela manual e depois dar um : e colocar a palavra BASE e em seguida, colocar dentro de parênteses os parâmetros de entrada para cada atributo dentro dos construtores, no caso, copiar e colar o construtor da classe principal, colocando apenas o atributo novo da subclasse. Isso faz com que o sistema entenda que você quer utilizar os mesmos atributos da classe principal. Depois disso, você não precisa colocar eles recebendo eles mesmo, apenas colocar o novo atributo recebendo ele mesmo.



Desse jeito.