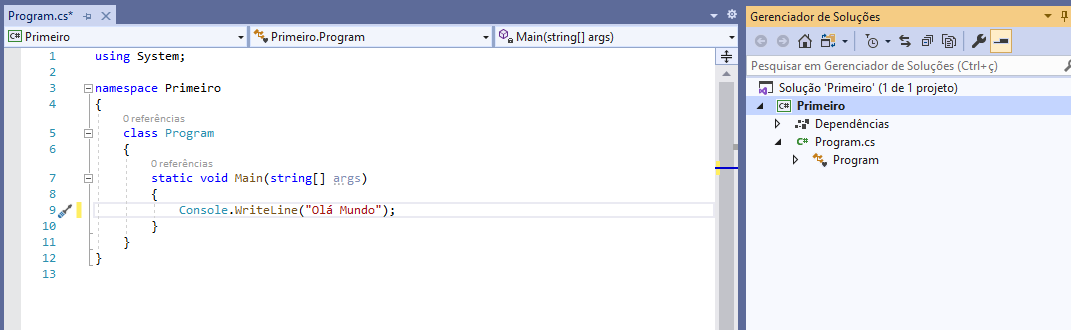
**Aula 10**

Código Fonte



\_\_\_\_

HotKeys:

Identação Automática: CTRL + K + D

Rodar o projeto: CTRL + F5

Fonte: CTRL + Mouse Scroll

\_\_\_\_

**Aula 12**

**Identação diferente:**

Tools>Options  
Buscar New Lines  
Desmarcar todos os itens contidos em “New line option for braces”.

\_\_\_\_

**Aula 13**

OverFlow = Quando um cálculo estrapola o limte da sua variável.

\_\_\_\_

**Aula 18**

**Simplificar casas decimais:**

Console.WriteLine(x.ToString("F2")) -> Duas casas decimais.

Ou para mostar com PONTO

Importar a biblioteca:

using System.Globalization;

Escrever o código:  
  
Console.WriteLine(x.ToString("F2", CultureInfo.InvariantCulture));

**Placeholders, concatenação e interpolação:**

//Concatenação

Console.WriteLine("A paciente " + z + " tem " + y + " anos e seu sexo é: " + w);

//PlaceHolder

Console.WriteLine("A paciente {0} tem {1} anos e seu sexo é {2} e seu salário atual é {3:F2}", z, y, w, x);

//Interpolação

Console.WriteLine($"A paciente {z} tem {y} anos e seu sexo é {w} e seu salário atual é {x:F2}");

\_\_\_\_

**Aula 19**

**Operadores de atribuição**

int a = 10;

int b = a++;

Console.WriteLine(a);

Console.WriteLine(b);

Valor de a será 11 e b 10;

B recebe 10 antes do a ser incrementado.

\_\_\_\_

**Aula 20**

//Conversão implícita

float x = 4.5f;

double y = x;

Console.WriteLine(y);

//Conversão casting

double a;

float b;

a = 5.1;

b = a; //Não funciona: a tem 8bytes e b tem 4bytes;

//Funciona assim:

b = (float)a;

**Aula 21**

**Operadores Aritméticos**

//Delta

double a = 1.0, b = -3.0, c = -4.0;

double delta = Math.Pow(b, 2.0) - 4.0 \* a \* c;

double x1 = (-b + Math.Sqrt(delta)) / (2.0 \* a);

double x2 = (b + Math.Sqrt(delta)) / (2.0 \* a);

**Aula 23**

**Entrada de Dados**

int n1 = int.Parse(Console.ReadLine());

char ch = char.Parse(Console.ReadLine());

double n2 = double.Parse(Console.ReadLine(), CultureInfo.InvariantCulture);

**Converter dados utilizando o Split para recortar**

Console.WriteLine("Digite seu nome, sexo (F ou M), idade e altura ");

string[] vet2 = Console.ReadLine().Split(' ');

string nome = vet2[0];

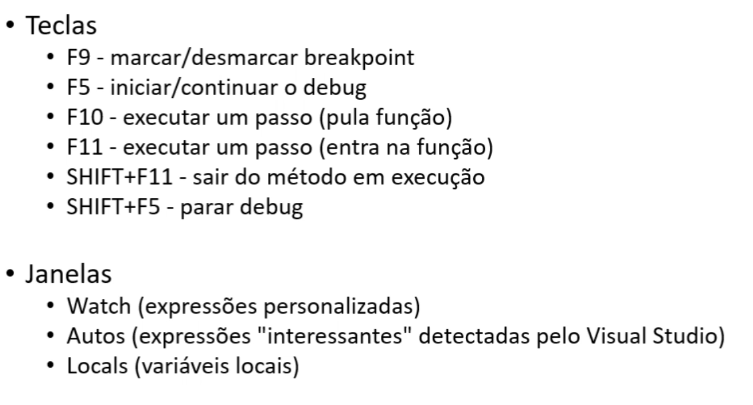
char sexo = char.Parse(vet2[1]);

int idade = int.Parse(vet2[2]);

double altura = double.Parse(vet2[3], CultureInfo.InvariantCulture);

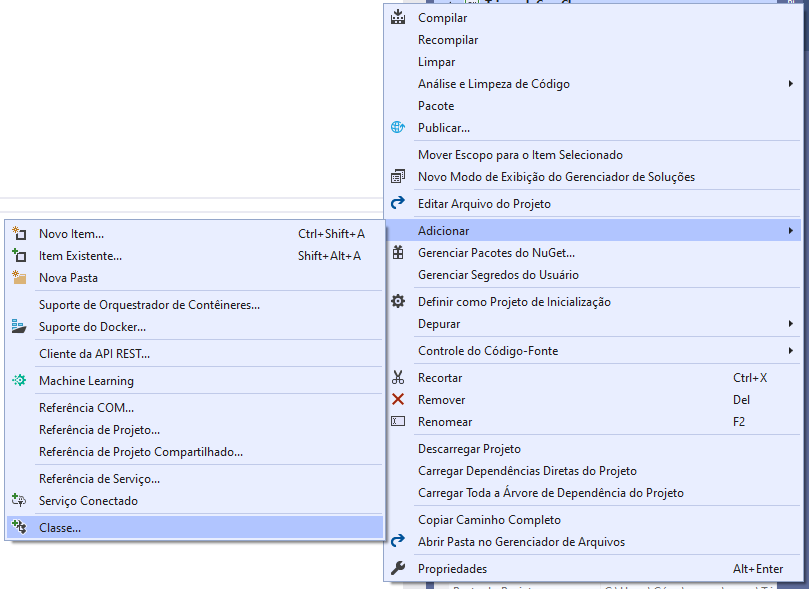
**Aula 31**

**Debug**

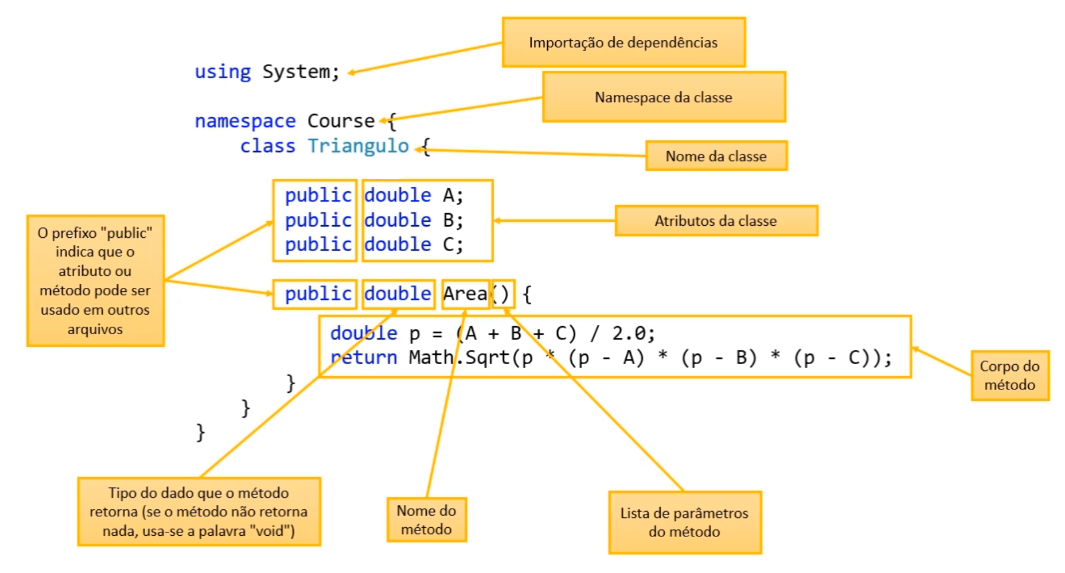
****

**Aula 39**

**Criar classe:**

****

Obs.: Método PUBLIC permite que seja acessado em outros lugares.



**Aula 46**

**Membro Estáticos**

São membros que fazem sentido independentemente de objetos. Não precisam de objetos para serem chamados. Exemplo: Math.sqtr. é um método que não precisa de objeto instanciado para chamar o método.

**Aula 52**

**Sobrecarga**

É um recurso que uma classe possui de oferecer mais de uma operação com o mesmo nome, porém com diferentes listas de parâmetros:

//Construtor normal

public Produto(string nome, double preco, int quantidade) {

Nome = nome;

Preco = preco;

Quantidade = quantidade;

}

//Sobrecarregar

public Produto(string nome, double preco) {

Nome = nome;

Preco = preco;

}

public Produto() {

}

**Aula 53**  
  
Uma classe pode funcionar sem construtores! Só colocar chaves e atribuir as variáveis criadas às variáveis do objeto dentro de chaves.

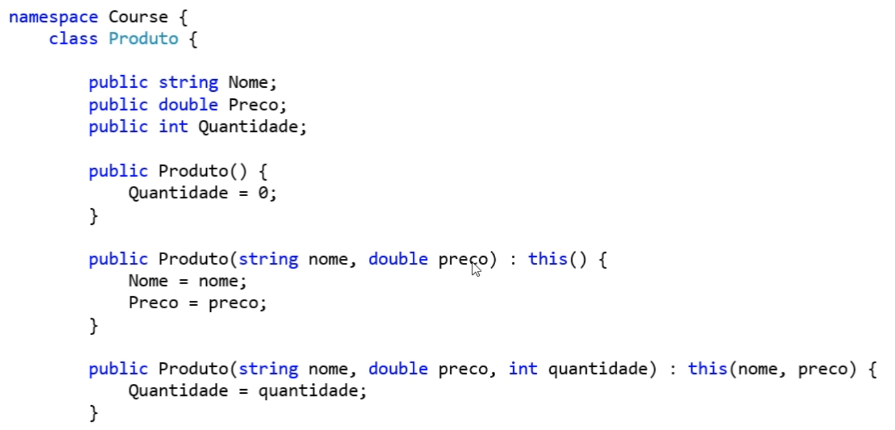
string Nome = “”;

double Preco = 0 ;

int Quantidade = 0;

Produto p = new Produto { Nome = Nome, Preco = Preco, Quantidade = Quantidade };

**Aula 54  
  
Entendendo o This nesta situação:**



O this nesta situação, serve para APROVEITAR o primeiro construtor chamado PRODUtO.

Da mesma forma, o this nesta situação, aproveita os parâmetros do outro construtor.

Obs.: a principal função deste this após os parâmetros, é aproveitar o código sem ter que escrever mais.

**Aula 55**

**Encapsulamento**

Encapsulamento serve para **controlar** o acesso aos atributos e métodos de uma classe. Exemplo: Quero permitir que alguém altere o nome contido em um atributo (posso criar o método get e set), porém, não desejo que alguém altere a quantidade de algum atributo, só crio o método get.

\*Colocar métodos Getters and Setter  
\*Colocar atributos privativos para não serem acessíveis  
\*Iniciar os atributos com \_letraMinúscula

**Aula 56**

Properties (Propriedades)

No C# não é usual fazer get e set separado:

Jeito menos certo:  
  
 public string Nome {

get {

return \_nome;

}

set {

if (nome != null && nome.Length > 1) {

\_nome = nome;

}

else {

Console.WriteLine("Valor digitado é nulo.");

}

}

}

Jeito Certo: incluir o nome VALUE no set

public string Nome {

get {

return \_nome;

}

set {

if (value != null && value.Length > 1) {

\_nome = nome;

}

else {

Console.WriteLine("Valor digitado é nulo.");

}

}

}

**Aula 73**

**Modificadores Ref e Out**

Ref serve para modificar o valor inicial de uma variável quando se utiliza, por exemplo, o membro estático.

Out, não precisa de ter uma variável inicializada.

class Modificador {

public static int valorTriplo (ref int valor) {

valor \*= 3;

return valor;

}

public static void valorTripeOut(int origin, out int result) {

result = origin \* 3;

}

}

static void Main(string[] args) {

//REF

int a = 10;

Modificador.valorTriplo(ref a);

Console.WriteLine(a);

//OUT

int b = 10;

Modificador.valorTripeOut(b, out b);

Console.WriteLine(b);

Console.ReadKey();

}

**Aula 57**

**Desalocação por escopo:**

void method1() {

int = 10;

if (x > 0) {

int y = 20;

}

Console.WriteLine(x);

Quando a execução sai desta linha onde está y, a desalocação ocorre.

\_\_\_\_

**Desalocação por Garbage Collector:**

Product p1, p2;

p1 = new Product(“TV”, 900, 0);

p2 = new Product(“Mouse”, 30, 0)

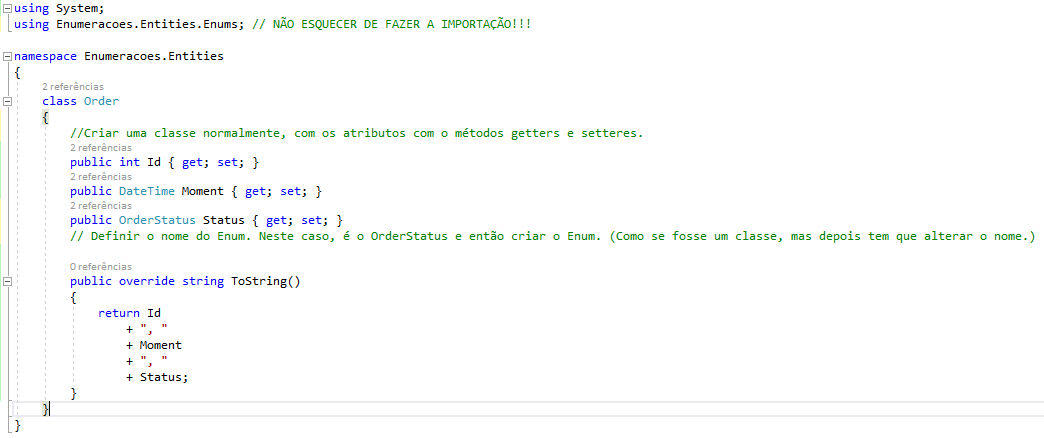
p1 = p2;

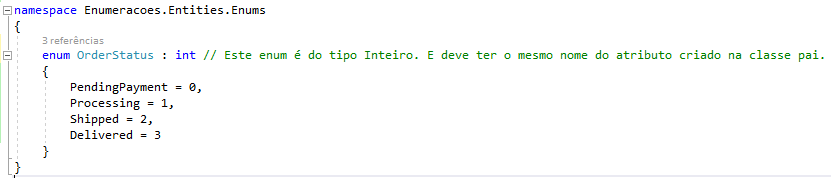
O ponteiro do p1 aponta para o local de memória de p2. Após um certo tempo, o garbage collector detecta que ele não tem mais usabilidade e é excluído.

**Aula 115**

**Enumerações**

É um tipo especial que serve para especificar de forma literal um conjunto de constantes relacionadas.  
Ex.: Dia de semana, ao invés de colocar os números, coloca-se os nomes dos dias da semana.

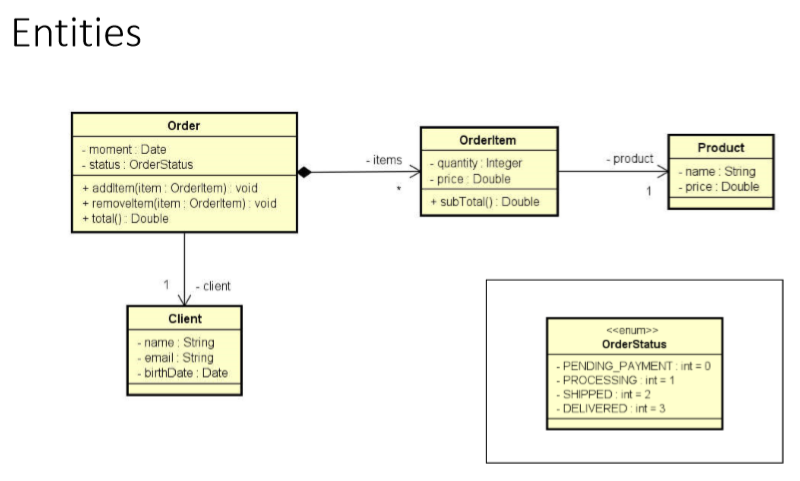


****

**Aula 117**

**Composição**

É um tipo de associação que permite que um objeto contenha outro.

Relação “tem-um” ou “tem-vérios”

Relação de Composição

Uma ordem, contém vários pedidos. O “todo” é a classe Order e a “parte” é a classe OrdemItem.

Não é uma relação de composição, pois, o cliente não é parte do pedido. Mas uma composição de objetos. Quando se implementa o pedido, tem um atributo do tipo cliente.

Quando se tem uma relação todo-parte, há uma lista.

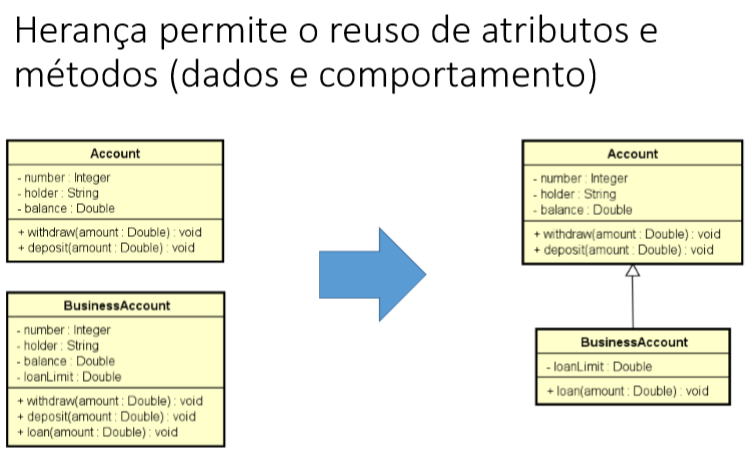
**Aula 126**

**Herança e Polimorfismo**

É um tipo de associação que permite que uma classe herde dados e comportamentos de outro.

Sintaxe em C# => :

“Extend”



A BusinessAccount tem tudo que a classe Account tem, por isto, ela herda a classe Account.

Significa herdar, entender. A classe que herda algo da outra, vai herdar os atributos da outra classe, e executa os atributos/métodos dela mesma.

Account: SuperClasse

BusinessAccount: Classe derivada

**Aula 128**

**Sobreposição ou sobrescrita**

É a implementação de um método de uma superclasse numa subclasse. Utiliza-se OVERRIDE. Em outras palavras, é quando se reimplementa na subclasse aquela operação QUE JÁ EXISTIA NA SUPER CLASSE.

**Aula 129**

**Classes e métodos selados: Sealed**

Evita que classes sejam herdadas e métodos sobreposto possa ser sobreposto novamente.

**Aula 130**

**Polimorfismo**

Recurso que permite que variáveis de um mesmo tipo mais genérico possam apontar para objetos de tipos espe3cíficos diferentes.

Objetos diferentes, porém os tipos das variáveis é o mesmo:

Account acc1 = new Account(1001, “Alex”, 500);

Account acc2 = new SavingsAccount(1002, “Anna”, 500, 0.01);

Acc1.Withdraw(10.0)

Acc2.Withdraw(10.0)

**Aula 134**

**Classes Abstratas**

São classes que não podem ser instanciadas.

É uma forma de garantir herança total: somente subclasses não abstratas podem ser instanciadas, mas nunca a superclasse abstrata.

abstract class Shape // Classe abstrata! Existem várias formas geométricas (shape)

{

public Color Color { get; set; }

public Shape(Color color)

{

Color = color;

}

public abstract double area(); //método abstrato para ser chamado em outras classes.

}

**Aula 141**

**Estrutura Try-Catch**

Bloco Try: contém o código que representa a exceção normal do recho de código que pode acarretar em uma exceção.

Bloco catch (capturar): contém o código a ser executado caso uma exceção ocorra.

Dever ser especificado o tipo da exceção a ser tratada. (Upcasting é permitido)

try // Tentar executar

{

int n1 = int.Parse(Console.ReadLine());

int n2 = int.Parse(Console.ReadLine());

int result = n1 / n2;

Console.WriteLine(result);

}

catch (DivideByZeroException) // se der um tipo de erro, será exibida a msg abaixo

{

Console.WriteLine("Division by zero is not allowed!");

}

catch (FormatException) // se der outro tipo de erro, outra msg será exibida.

{

Console.WriteLine("Não use string!");

}

Bloco Finally: é um bloco que contém código a ser executado independente de ter ocorrido ou não uma exceção.

**Aula 186**

**Trabalhar com arquivos: File, FileInfo e IOException**

File: membros estáticos, não precisa instanciar objetos pra chamar os métodos. É mais simples que o FileInfo.

FileInfo: precisa instanciar um objeto. É mais pesado.

IOException: exceções para trabalhar com arquivos.