

E-Book



.NET

Developer

Explorando a
linguagem C#





mensagem de recepção

Boas-vindas à nossa série de ebooks! Preparamos estes materiais pensando em como transmitir mais conhecimento diversificado para você explorar diferentes meios de desenvolver suas habilidades.

Aproveite este material e bons estudos!



SUMÁRIO

04

Propriedades, métodos e construtores



08

Manipulando valores



10

Exceções e Coleções



13

Tuplas, operador ternário e desconstrução de um objeto



15

Nuget, serializar e atributos do C#



18

Tipos especiais no C#



21

Stack, Heap e Garbage Collector



III Propriedades, métodos e construtores

Introdução ao Módulo

Explorar a linguagem C#, apresentando diferentes possibilidades e capacidades da linguagem, bem como nos aprofundarmos mais sobre suas características.

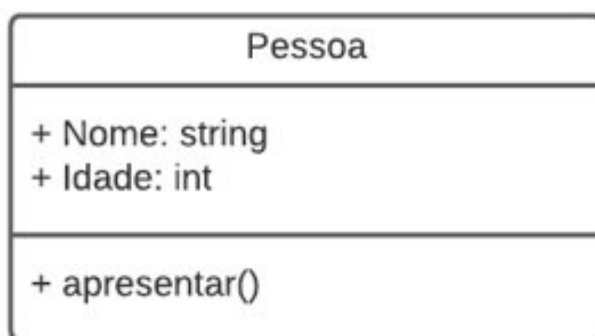
Requisitos :

- Módulo Fundamentos
- Algoritmos e Lógica de programação

Introdução Propriedades

Uma propriedade é um membro que oferece um mecanismo flexível para ler, gravar ou calcular o valor de um campo particular.

Por exemplo, no caso da classe Pessoa, podemos colocar propriedades com validadores, nome e idade. O validador do nome será que ele não pode ser vazio e que não pode passar de 20 caracteres. Já a idade não poderá receber valores negativos.



Propriedades na prática

Escrever **prop** na classe C# no VS Code serve como atalho para criação de novas propriedades:

```
10  |  public int MyProperty { get; set; }
```

Validações no GET e SET

Resumo da aula : Criação de validações para os métodos get e set. Propriedades com modificador de acesso private.

Um dos pilares da programação orientada a objetos é o encapsulamento, onde os atributos ficam protegidos de modificações externas. É exatamente esse o propósito de tornar um campo do tipo private.

Body Expressions

São maneiras mais resumidas de escrever os códigos do get e do set.

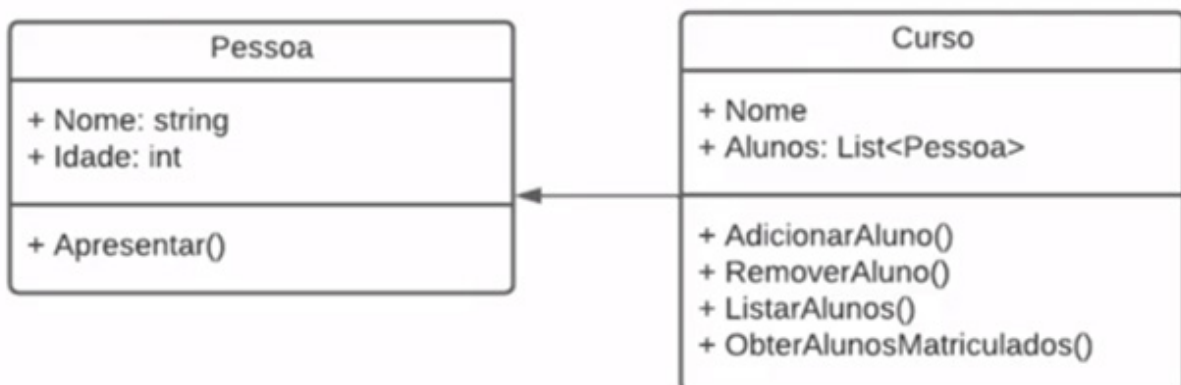
```
14  get => _nome.ToUpper();
```

Modificadores de acesso

- public
 - Qualquer um pode acessar
- private
 - Acesso restrito. Só é permitido acessar se for dentro da própria classe.

Introdução métodos

Um método é um bloco de código que contém uma série de instruções. Vamos ver um exemplo :



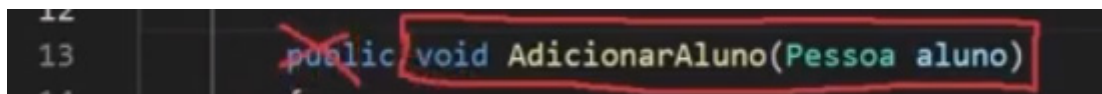
Os métodos são:

- Classe pessoa
 - Apresentar()
- Classe curso
 - AdicionarAluno()
 - RemoverAluno()
 - listarAlunos()
 - ObterAlunosMatriculados()

Implementando a classe curso

Resumo da aula : Criação da classe Curso. Explicação sobre assinatura de método.

Podemos identificar que um determinado código é um método pela assinatura de método :



```
12  
13 public void AdicionarAluno(Pessoa aluno)  
14
```

Nesse caso, o modificador de acesso `public` não faz parte da assinatura de um método. Detalhando as sessões da assinatura de método, temos:

- void
 - Tipo de retorno do método.
 - No nosso caso, não será retornado nada pois o método é apenas para adicionar um novo aluno. Por isso a utilização da palavra void que significa vazio, ou seja, retorno vazio.
- AdicionarAluno
 - Nome do metodo

- (Pessoa aluno)
 - Parâmetros do método
 - Pessoa
 - Tipo variável
 - No nosso caso, será do tipo da classe Pessoa
 - aluno
 - Nome da variável

Tipos de retorno

Resumo da aula : Criação de um método para retornar a quantidade de alunos.

Quando na assinatura de método colocamos um tipo de retorno diferente de `void`, significa que será obrigatório ao método retornar alguma coisa do mesmo tipo.

Trabalhando com construtores

Resumo da aula : Criação de dois construtores da classe Pessoa. Utilizar o construtor na classe principal.

Os construtores permitem que o programador defina valores padrão, limite a instanciação e grave códigos flexíveis e fáceis de ler.

Manipulando valores

Concatenação de Strings

Podemos formatar valores em diversas representações. Essa formatação de valores é conhecida por concatenação ou interpolação.

Interpolação de Strings

Uma outra forma de concatenar uma String é através da interpolação. Esse modo consiste em utilizarmos o sinal de cifrão (\$) antes da abertura das aspas, e depois utilizarmos as chaves ({ }) dentro das aspas.

Formatando valores monetários

```
C# Program.cs
1  using ExemploExplorando.Models;
2
3  decimal valorMonetario = 82.40M;
4
5  Console.WriteLine($"{valorMonetario:C}");
```

A letra “C” depois dos dois pontos significa currency, que é a moeda de acordo com a configuração de localização atual do sistema onde o código está sendo executado.

Alterando a localização do código

```
4  CultureInfo.DefaultThreadCurrentCulture = new CultureInfo("pt-BR");
```

Alterando a localização da cultura

Para podermos representar um valor em outra cultura caso o sistema esteja travado para uma cultura específica, podemos fazer o seguinte :

```
C# Program.cs
1  using ExemploExplorando.Models;
2  using System.Globalization;
3
4  CultureInfo.DefaultThreadCurrentCulture = new CultureInfo("pt-BR");
5
6  decimal valorMonetario = 1582.40M;
7
8  Console.WriteLine(valorMonetario.ToString("C", CultureInfo.CreateSpecificCulture("en-US")));
```


Representando Porcetagem

```
10 double porcentagem = .3421;  
11  
12 Console.WriteLine(porcentagem.ToString("P"));
```

Formatando o tipo DateTime

Formatando o tipo DateTime com o método `ToString`.

Pontos importantes:

- letra M maiúscula representa o mês
- letra m minúscula representa minutos
- letra H maiúscula representa hora no formato 24h

Formatando data e hora

- `ToShortDateString`,
 - Mostrar apenas a data, sem as horas.
- `"yyyy-MM-dd HH:mm"`,
 - o formato que a data está nessa variável de entrada
- `CultureInfo.InvariantCulture`,
 - Como já definimos o formato da data, aqui sinalizamos que independe da localização.
- `DateTimeStyles.None`,
 - Da mesma forma, como já estamos utilizando um formato, aqui não precisaremos de estilo.
- `out DateTime data)`
 - variável de saída

Exceções e Coleções

Introdução Exceções

Os recursos de manipulação de exceção da linguagem C# ajudam você a lidar com quaisquer situações excepcionais ou inesperadas que ocorram quando um programa for executado.

Realizando a leitura de um arquivo

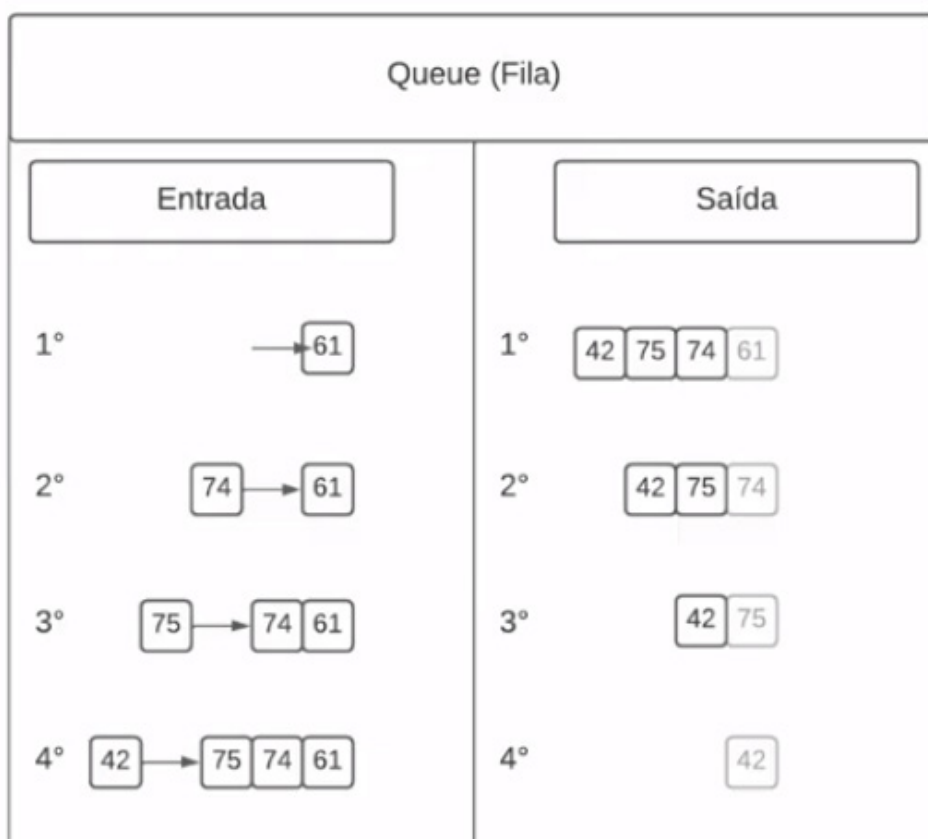
Resumo da aula : Criação de uma nova pasta e um arquivo de texto. Escrever um código na classe principal para ler o arquivo de texto.

File é uma classe própria para ler arquivos.

Entendendo o bloco finally

O bloco finally é utilizado em conjunto do bloco try...catch e sempre será executado no final do código, independente de ter acontecido uma exceção ou não.

Introdução a Filas



Seu funcionamento é exatamente como uma fila da vida real, o primeiro elemento que chegar tem que ser também o primeiro a sair, também conhecido como FIFO (First In First Out). Outra regra é que a coleção Fila também deve respeitar a instrução de possuir itens apenas do mesmo tipo específico.

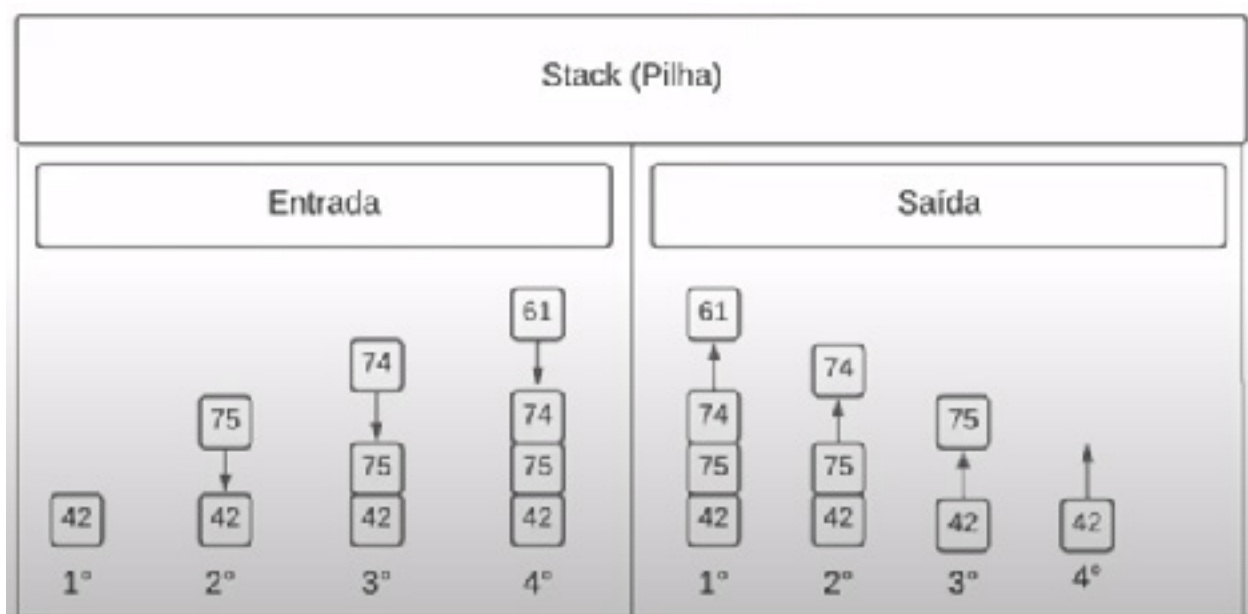
Fila na prática

O método

- **Enqueue**
 - Adicionar um novo elemento ao final da fila.
- **Dequeue**
 - Remove e retorna sempre o primeiro elemento.

Introdução a Pilhas

A coleção do tipo Pilha ou Stack tem o funcionamento parecido com as Filas, porém obedecem a ordem de LIFO (Last in First Out) ou seja, os que entram por último serão os primeiros a saírem.



Pilha na prática

O método

- **Enqueue**
 - Adicionar um novo elemento ao final da fila.
- **Dequeue**
 - Remove e retorna sempre o primeiro elemento.

Introdução Dictionary

Um Dictionary é uma coleção de chave-valor para armazenar valores únicos sem uma ordem específica. O dictionary deve garantir que os elementos sejam únicos com base na chave.

Para entendermos melhor o código :

- **Dictionary<string, string>**
 - o primeiro **string** é o tipo da chave
 - o segundo **string** é o tipo do valor que essa chave terá
- **estado.Add**
 - Método para adicionar novos elementos
 - O primeiro parâmetro é a chave
 - Os egundo parâmetro é o valor

Removendo e alterando elementos

Para removermos um elemento no dictionary, vamos utilizar o método **Remove(x)** onde x é a chave que queremos remover.

Não é possível alterar uma chave , para isso precisamos remover e adicionar a chave novamente.

Tuplas, operador ternário e desconstrução de um objeto

Introdução

Tuplas fornecem sintaxe para agrupar vários elementos de dados em uma estrutura de dados leve. Uma tupla não é uma coleção mas pode conter uma coleção.

Outra sintaxe da tupla

Outras duas maneiras de representar a mesma tupla, porém com a desvantagem de não nomear as chaves :

```
ValueTuple<int,string,decimal> outroExemplotupla=(1,"Leonardo","buta",1,80M);
```

```
Var outroExemplotupla = Tuple.Create(1, "leonardo", "Buta", 1,80M) ;
```

Tupla em métodos

- **File**
 - Classe para realizar a leitura de arquivos
- **ReadAllLines**
 - Método para ler as linhas de um arquivo de texto
- **count**
 - Método que retorna a quantidade total de itens de uma lista

Testando nosso método

- `var`
 - utilizada para identificar automaticamente os tipos de variáveis que recebem os valores do retorno do método.

Desconstrutor

- `out`
 - Palavra utilizada antes do tipo nos parâmetros.
 - Para dizer quais serão as variáveis de saída.

IF Ternário

Foi mostrado na prática o uso do if ternário, reduzindo o número de linhas do código.

III Nuget, serializar e atributos do C#

Introdução gerenciador de pacotes

Um pacote é um conjunto de códigos úteis, que possibilita o compartilhamento e reuso do código. Um exemplo de um pacote é o Google Maps.

Nuget é um gerenciador de pacotes, que permite desenvolvedores compartilharem códigos e bibliotecas. É o gerenciador de pacotes oficial da plataforma .NET.

Instalando um pacote pelo VS Code

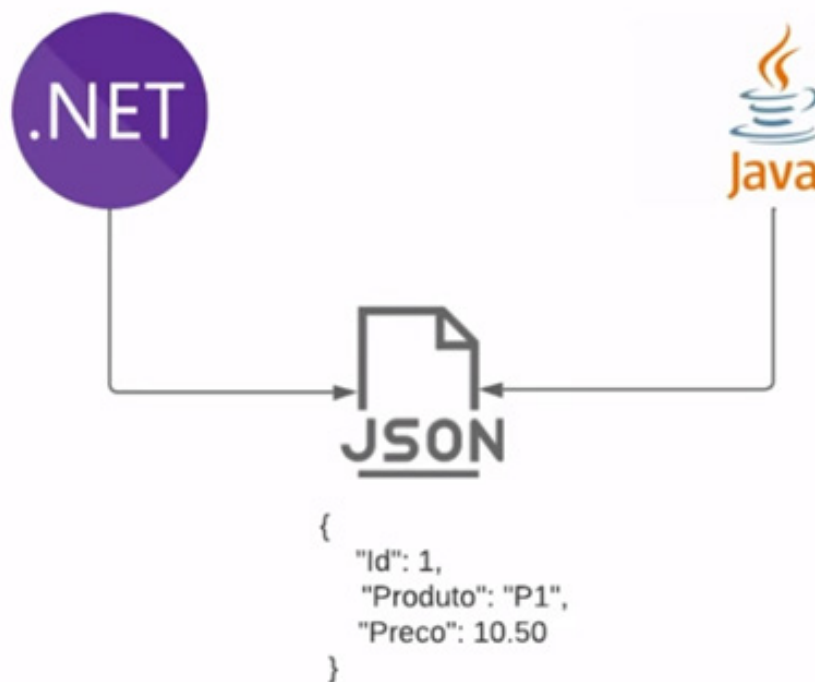
Objetivo da aula : instalar o pacote Newtonsoft.Json no VS Code utilizando CLI do Nuget.

- ***dotnet add package Newtonsoft.Json***
 - Comando para instalar o pacote

Introdução serialização

O processo de serializar consiste em transformar objetos em um fluxo de bytes para seu armazenamento ou transmissão.

JSON ou Javascript Object Notation é um formato de texto que segue a sintaxe do Javascript, muito usada para transmitir dados entre aplicações. É muito utilizado como padrão de comunicação em APIs.



Serialização na prática

- **JsonConvert**
 - Classe vinda do pacote Newtonsoft.Json
- **SerializeObject**
 - Método que retorna uma string no formato JSON.
- **Formatting.Indented**
 - Utilizado para formatar a string de saída com indentação.

Serialização na prática

- **File.WriteAllText**
 - Utilizado para gerar um arquivo, geralmente no formato json
 - Recebe dois parâmetros :
 - O caminho com o nome do arquivo de saída
 - O objeto serializado cuja informação será escrita no arquivo de saída

JSON Viewer

Resumo da aula : Acessar o site Code Beautify e na opção JSON Viewer explorar as funções:

- Validador de JSON
- Beautify
- Minify

DateTime em JSON

Resumo da aula : Representar o tipo DateTime em JSON e conhecer sobre o padrão ISO 8601.

- ISO 8601
 - padroniza a representação de data e hora entre sistemas

Deserializando um objeto

Resumo da aula : Criação da classe Venda para ser usada como modelo. Fazer o caminho contrário, onde a partir de um arquivo JSON será gerada uma lista de objetos do tipo Venda .

- `JsonConvert.DeserializeObject`
 - Método que recebe como parâmetro o caminho do arquivo JSON

Atributos

Resumo da aula : Entender melhor como funciona o método de deserialização. Utilizar o atributo JsonProperty na classe Venda como solução para que o nome da propriedade respeite a convenção do C# caso a informação no arquivo JSON não respeite.

Exemplo:

```
[JsonProperty("Nome_Produto")]  
public string Produto { get; set; }
```

Tipos especiais no C#

Introdução e tipos de valores nulos

Um tipo de valor nulo permite representar o tipo da variável e adicionalmente, o valor nulo.

Um exemplo é o tipo de dado bool que pode receber um valor nulo além de true ou false, caso seja utilizado o sinal de interrogação logo após o tipo da variável :

```
bool = true, false  
bool? = true, false, null
```

Tipos nulos na prática

Resumo da aula: Ver na prática o uso do valor null numa variável do tipo bool. Utilizar as propriedades `HasValue` e `Value` que aparecem como opção no cenário onde a variável pode receber null como valor.

Tipos nulos em propriedades

Resumo da aula : Adicionar uma nova informação ao arquivo JSON chamada Desconto, que poderá ou não receber null. Criar uma propriedade para receber essa informação na classe Venda e torná-la do tipo decimal nullable, mostrando no código através de um if ternário.

Tipos anônimos

Os tipos anônimos fornecem um modo conveniente de encapsular propriedades somente leitura.

Criação de uma variável do tipo anônimo que encapsula 3 propriedades : Nome, Sobrenome e Altura.

```
var tipoAnonimo = new {Nome = "Leonardo", Sobrenome = "Buta", Altura = 1.80};
```

Tipos anônimos em coleção

Resumo da aula : Criação de uma variável do tipo Lista onde apenas as informações Produto e Preço serão selecionadas através do método Select de uma outra variável também do tipo Lista que contém todas as informações deserializadas do arquivo JSON.

```
var listaAnonimo = listaVenda.Select ( x => new {x.Produto, x.Preço}
```

Introdução classes genéricas

As classes genéricas encapsulam operações que não são específicas de um determinado tipo de dados.

Criando uma classe genérica

Resumo da aula : Criação da classe genérica MeuArray que “imita” o funcionamento de uma Lista, com um método para adicionar elementos genéricos.

- `public class MeuArray<T>`
 - A letra T significa que se trata de um tipo genérico
 - poderia ser qualquer outra letra.
 - o tipo será atribuído ao instanciar a classe
- `public void AdicionarElementoArray(T elemento)`
 - Como não se sabe o tipo do elemento que será adicionado, aqui repete-se o uso da letra T para indicar o tipo do elemento.

Testando nossa classe genérica

Resumo da aula : Criação de uma variável do tipo Lista onde apenas as informações Produto e Preço serão selecionadas através do método Select de uma outra variável também do tipo Lista que contém todas as informações deserializadas do arquivo JSON.

```
8     MeuArray<int> arrayInteiro = new MeuArray<int>();  
15    MeuArray<string> arrayString = new MeuArray<string>();
```

Métodos de extensão

Nada mais é do que estender um determinado tipo com algum comportamento, tornando esse comportamento “nativo” àquele tipo.

Criando uma classe genérica

Resumo da aula : Criação da classe genérica MeuArray que “imita” o funcionamento de uma Lista, com um método para adicionar elementos genéricos.

```
8     public static class IntExtensions  
9     {  
10        public static bool EhPar(this int numero)  
11        {  
12            return numero % 2 == 0;  
13        }  
14    }
```

- **static**
 - A classe não será instanciada
- **this int**
 - refere-se a todos os objetos do tipo inteiro
 - toda variável do tipo int terá acesso a esse método

Resumo da aula : Criação da classe IntExtensions.Criação do método EhPar que estende o tipo int.

Stack, Heap e Garbage Collector

Entendendo a Stack e Heap

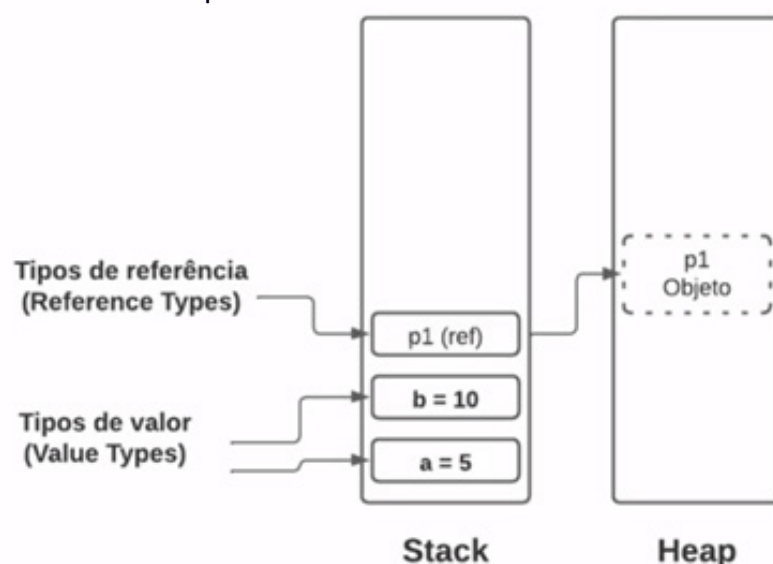
Dependendo do tipo da variável, ela será armazenada na memória Stack ou Heap. Variáveis simples como do tipo `int` são armazenadas na memória Stack. Variáveis de tipo mais complexa como `Object` ficam na memória Heap e apenas uma referência fica na memória Stack.

Limpeza de memória

Finalizada a execução do programa, a memória do tipo Stack irá apagar automaticamente na ordem LIFO (O último que entrar é o primeiro a sair) tudo aquilo que estava depositado nela. Já para limpar a memória HEAP, o Garbage Collector entra em ação e elimina da memória HEAP tudo aquilo que perdeu a referência com a memória Stack.

Tipos de valor e referência

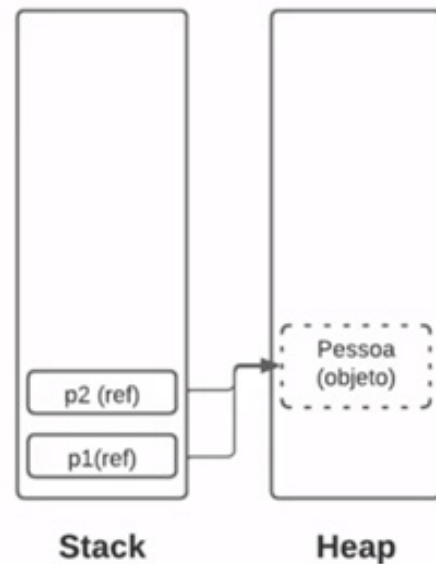
- Tipo de valor
 - Uma variável de um tipo de valor contém uma instância do tipo.
- Tipo de referência
 - Uma variável de um tipo de referência contém uma referência a uma instância do tipo.



Atribuições de tipos

Ao atribuir uma variável do tipo objeto a uma outra variável, será criada uma nova referência na memória STACK do mesmo objeto que está na memória HEAP.

```
void Metodo() {  
    // Linha 1  
    Pessoa p1 = new Pessoa("Leonardo", "Buta");  
  
    // Linha 2  
    Pessoa p2 = p1;  
    p2.Nome = "Eduardo";  
}
```



Atribuindo tipo de referência

Resumo da aula : Criação de 2 objetos do tipo Pessoa, p1 e p2, onde para p2 é atribuído p1. Alteração da propriedade Nome de p2, mostrando que a mesma propriedade de p1 também será alterada.

Obrigado pela leitura!

Você não chegou aqui pulando páginas, né? 😊

Brincadeiras à parte, realmente nós da DIO
esperamos que esteja curtindo sua jornada de
aprendizado aqui conosco e desejamos seu
sucesso sempre!

Vamos em frente!

