Curso Estruturas de Dados e Algoritmos Expert

Prof. Nelio Alves

Tópicos iniciais

https://devsuperior.com.br

Sobre este capítulo

Objetivos:

- Esclarecer questões sobre Estruturas de Dados e Algoritmos
- Oferecer um nivelamento sobre tópicos que vamos precisar nos capítulos seguintes

Conteúdo:

- Questões sobre Estruturas de Dados e Algoritmos
- O mínimo sobre objetos e funções que você precisa saber
  - Tipos estruturados (classes, atributos)
  - Chamada de função sem OO e com OO (métodos)
- Comportamento de memória
  - Tipos referência vs. tipos valor
  - Desalocação de memória: garbage collector e escopo local
- Leitura de arquivos JSON

Questões sobre Estruturas de Dados e Algoritmos

Eu já não aprendi “Algoritmos” na disciplina “Lógica de Programação e Algoritmos”?

“Tudo” é algoritmo. Porém:

Lógica de Programação e Algoritmos
- Estrutura sequencial (entrada, saída e processamento)
- Estruturas de controle (condicionais e loops)
- Arrays (vetores e matrizes)
- Funções

→ Algoritmos **simples**

Estruturas de Dados e Algoritmos
- Técnicas específicas/avançadas

→ Algoritmos **complexos**

Fazer a leitura dos dados de um conjunto de clientes de uma companhia telefônica, informando, para cada um, o seu nome, o número do seu telefone, o tipo de assinatura (0, 1 ou 2) e o número de minutos consumidos no mês (não é para fazer a leitura do valor da conta, pois este será calculado automaticamente mais tarde). Veja o exemplo abaixo:

Nome	Telefone	Tipo	Minutos	ValorDaConta
Maria José	3222-1234	1	120	
XY Informática	3223-6666	2	457	
Joaquim Silva	3222-3344	1	50	
Antônio Carlos	3212-6622	0	134	

Um dos problemas mais complexos do nosso curso de Lógica de Programação e Algoritmos

Cada coluna da tabela deve ser armazenada em um VETOR.

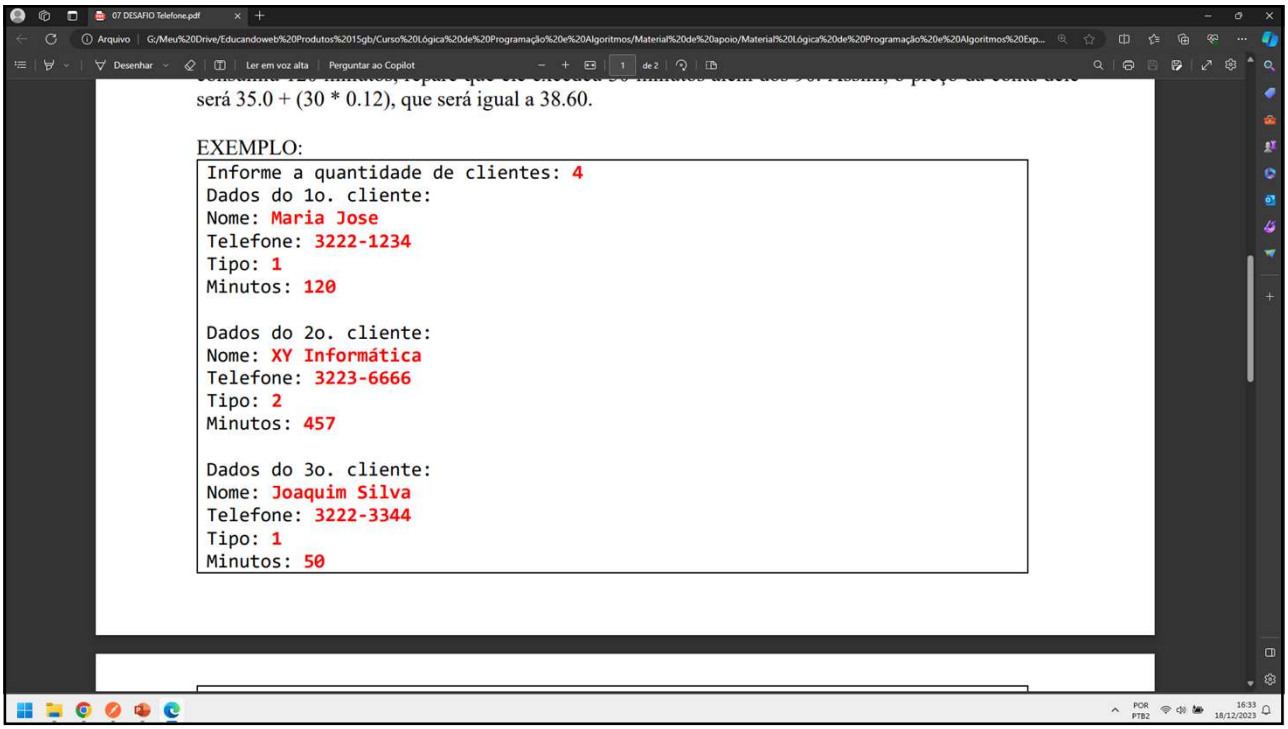
Fazer a leitura também dos três tipos de assinatura, sendo que cada tipo de assinatura possui um preço básico, e o preço do minuto excedente. Utilizar uma MATRIZ 3x2 para armazená-los, sendo que a primeira coluna corresponde ao preço básico da assinatura e a segunda coluna corresponde ao preço do minuto excedente, conforme mostrado no exemplo abaixo:

	0	1
0	25.5	0.10
1	35.0	0.12
2	60.0	0.15

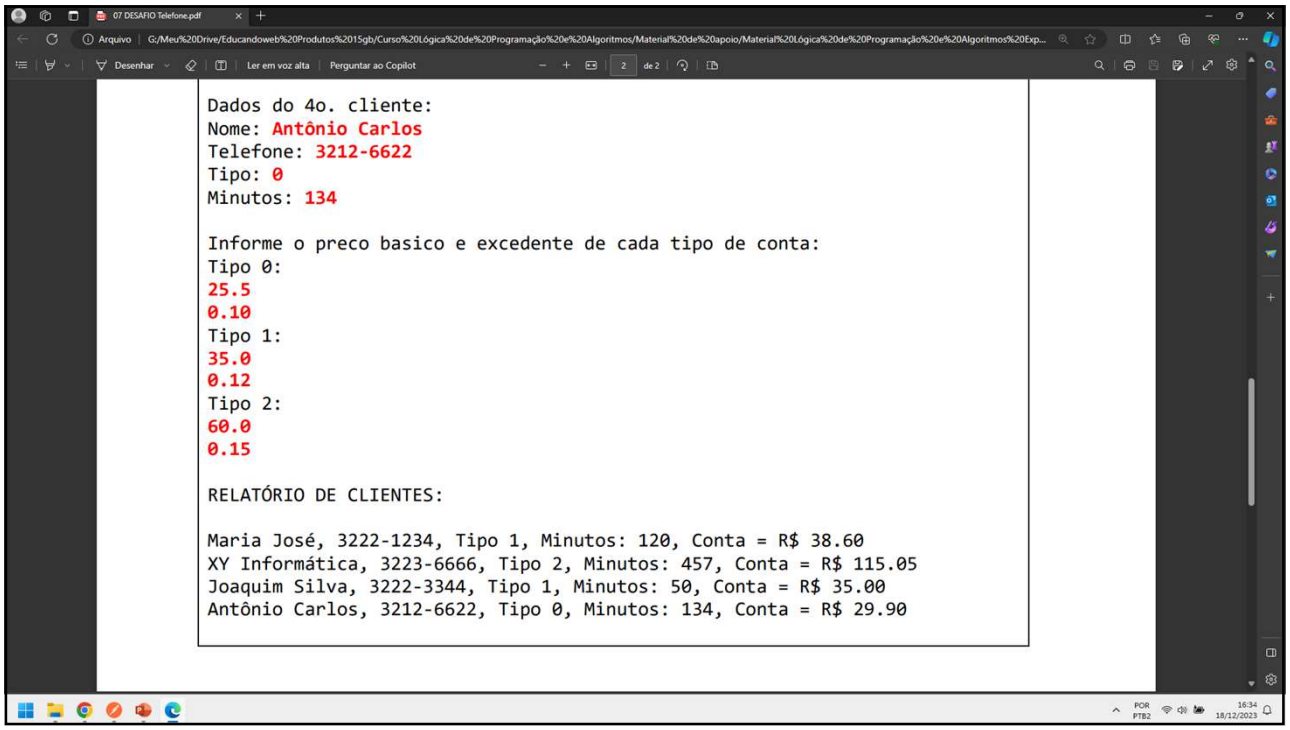
(certifique-se de que você entendeu a matriz ao lado. Por exemplo: a matriz diz, por exemplo, que o preço básico da assinatura tipo 1 é 35 reais e que cada minuto excedente deste tipo custa 12 centavos)

Depois de ler os dados dos clientes, bem como os dados dos três tipos de assinatura, calcular o valor da conta de cada cliente e mostrar uma tabela conforme exemplo abaixo. A regra de cálculo da conta de um cliente é a seguinte: se o número de minutos consumidos for menor ou igual a 90, então o valor da conta é simplesmente o preço básico da assinatura; caso contrário, o valor da conta é o preço básico mais o valor total dos minutos excedentes a 90. Por exemplo: se um cliente tem uma assinatura tipo 1 e consumiu 120 minutos, repare que ele excedeu 30 minutos além dos 90. Assim, o preço da conta dele será 35.0 + (30 * 0.12), que será igual a 38.60.

5



6



Exemplo de problema que requer técnica de programação dinâmica:

Dada um string S, retorne o maior substring palíndromo contido em S.

Exemplo 1:	Exemplo 2:	Exemplo 3:
Entrada: S = "geeks"	Entrada: S = "ceceg"	Entrada: S = "bnbggbnof"
Saída: "ee"	Saída: "cec"	Saída: "nbggbn"
	Obs: "ece" também é uma resposta válida.	

```javascript
const longestPalindrome = (str) => {
  const length = str.length;
  const dp = Array(length)
    .fill(false)
    .map(() => Array(length).fill(false));
  let maxLength = 1;
  let start = 0;

  // Single characters are palindromes
  for (let i = 0; i < length; i++) {
    dp[i][i] = true;
  }

  // Check for palindromic substrings of length 2
  for (let i = 0; i < length - 1; i++) {
    if (str[i] === str[i + 1]) {
      dp[i][i + 1] = true;
      maxLength = 2;
      start = i;
    }
  }

  // Check for palindromic substrings of length greater than 2
  for (let len = 3; len <= length; len++) {
    for (let i = 0; i < length - len + 1; i++) {
      const j = i + len - 1;
      if (str[i] === str[j] && dp[i + 1][j - 1]) {
        dp[i][j] = true;
        if (len > maxLength) {
          maxLength = len;
          start = i;
        }
      }
    }
  }
  return str.slice(start, start + maxLength);
};
```

https://medium.com/@stheodorejohn/finding-the-longest-palindromic-substring-in-javascript-199796db1bdf

9

Estruturas de Dados é sobre o quê?

Estruturas de Dados é sobre a organização e armazenamento de dados na memória de um computador.

Isso envolve o design e implementação de métodos para organizar e gerenciar dados de maneira que permita uma recuperação e modificação eficientes.

Estruturas de dados são um fundamento essencial para escrever software robusto e eficiente.

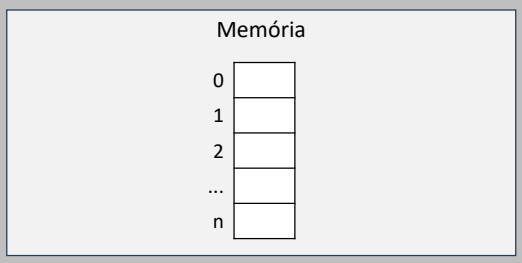
Objetivos:

- Organização e abstração dos dados
- Eficiência na recuperação e modificação de dados
- Bom gerenciamento de memória

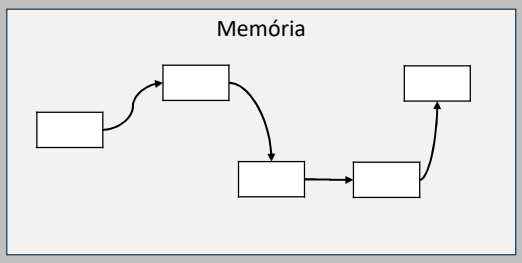
10

Principais tipos de estruturas de dados

- Arrays: coleção de elementos ocupando uma porção contígua de memória. Permite rápido acesso aos elementos pelo seu índice.

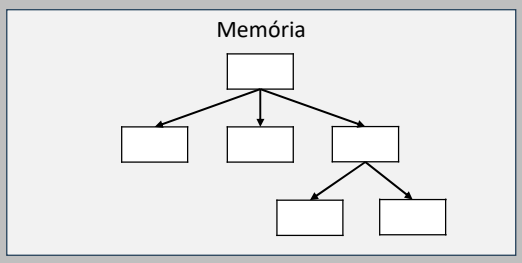


- Listas: coleção de elementos armazenados em nós **sequenciais**.

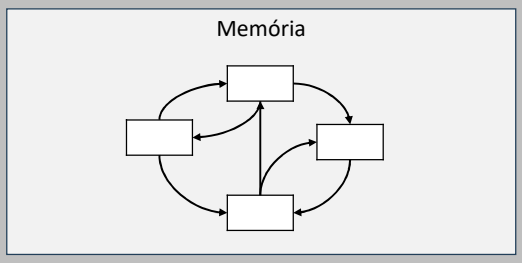


Principais tipos de estruturas de dados

- Árvores: coleção de elementos armazenados em nós de forma **hierárquica**.

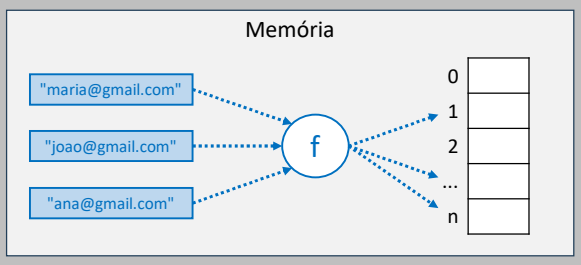


- Grafos: coleção de elementos armazenados em nós com **livre conexão** entre eles.



Principais tipos de estruturas de dados

- Tabelas hash: coleção de elementos que usa uma função de hash para mapear chaves aos elementos. Permite rápido acesso aos elementos pela sua chave.



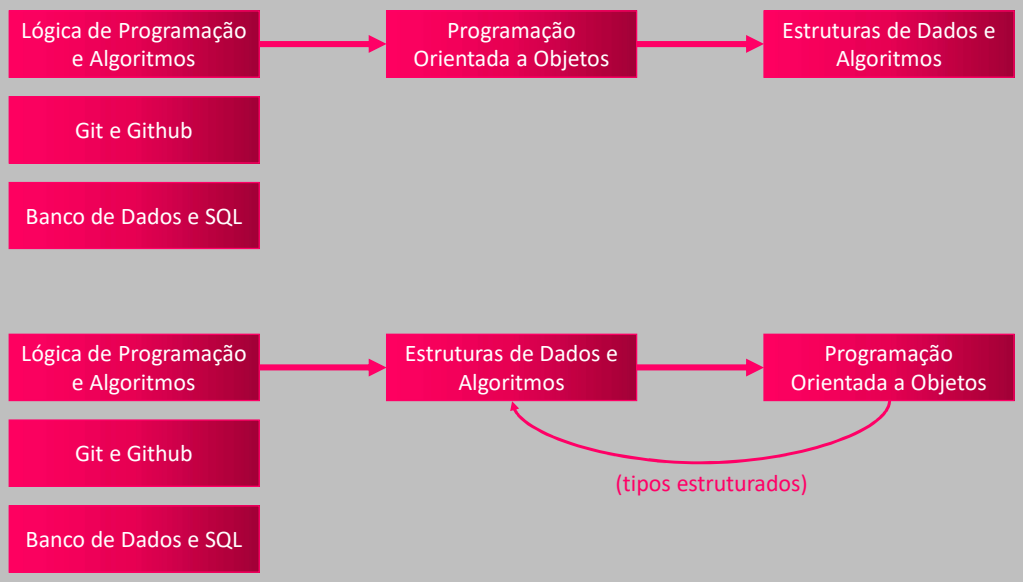
Precisa saber OO antes de ED?

Boa pergunta!

Toda base de OO não precisa.

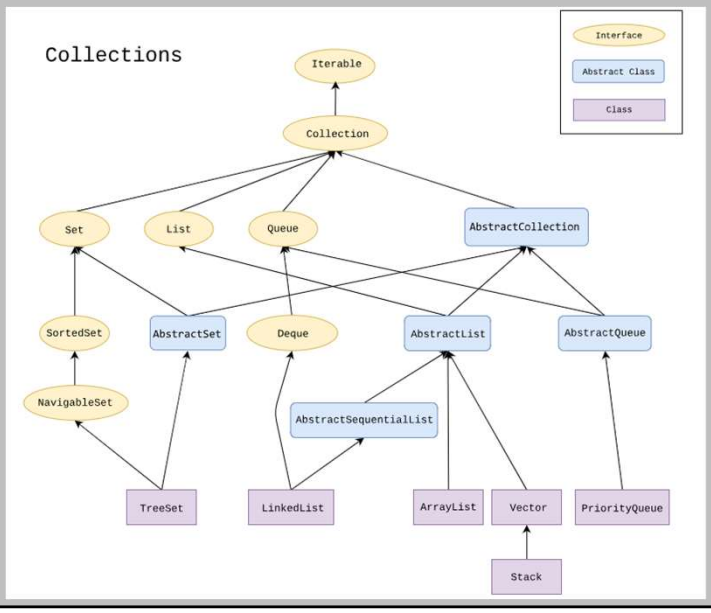
Alguns currículos colocam ED antes de OO. Neste caso, é necessário conhecimento de **tipos estruturados** (ou classes) antes de entrar de fato em ED.

Caminho de formação (Fundamentos de Programação)



15

Porém, bibliotecas de estruturas de dados que usam recursos modernos como herança, interfaces e generics, geralmente são soluções mais elegantes e flexíveis.



16

Quero seguir carreira da linguagem X (ou outra linguagem). Esse curso é para mim?

- SIM!
- Estruturas de Dados e Algoritmos é um tópico sobre **técnicas de elaboração de soluções** e não sobre uma linguagem específica.
- O importante é saber as estruturas e técnicas.
- Como programador(a) você deve ser capaz de:
  1. Entender a técnica e o raciocínio
  2. Saber implementar isso na sua linguagem escolhida

Desde que você saiba o básico da sua linguagem escolhida,

E desde que você esteja em um computador com Internet,

Você deve ser capaz de “traduzir” o código de outra linguagem para sua linguagem escolhida.

“Ái, mas eu só aceito se for na linguagem X"

- Já está na hora de mudar esse pensamento. Linguagem não é time de futebol.
- Em problemas complexos, a linguagem é o menor dos problemas.
- Problema difícil vai ser difícil independentemente da linguagem.
- Programador deve saber usar Google, Stackoverflow, ChatGPT, etc.
  - Exemplo de palavras chave: hashmap python equivalent

O mínimo sobre objetos e funções que você precisa saber

- Tipos estruturados (classes, atributos)
- Chamada de função sem OO e com OO (métodos)

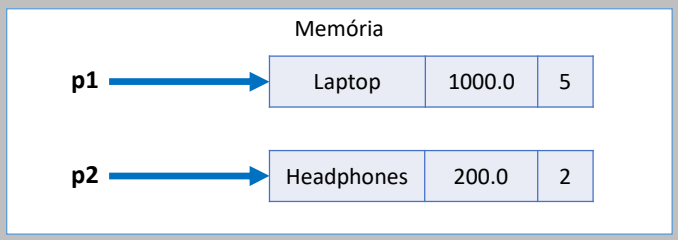
(todas linguagens)

Tipos estruturados (classes, atributos) (Javascript)

Exercício:

Fazer um programa para armazenar em memória os dados de dois produtos (nome, preço e quantidade), depois mostrar esses dados na tela.

```
Exemplo:
Laptop, $1000.00, 5
Headphones, $200.00, 2
```



```
class Product {
    constructor(name, price, quantity) {
        this.name = name;
        this.price = price;
        this.quantity = quantity;
    }

    toString() {
        return `${this.name}, $${this.price.toFixed(2)}, ${this.quantity}`;
    }
}
```

```
const p1 = new Product("Laptop", 1000.00, 5);
const p2 = new Product("Headphones", 200.00, 2);

console.log(p1.toString());
console.log(p2.toString());
console.log(p1.name);
console.log(p2.name);
```

23

```
function Product(name, price, quantity) {
    this.name = name;
    this.price = price;
    this.quantity = quantity;

    this.toString = function () {
        return `${this.name}, $${this.price.toFixed(2)}, ${this.quantity}`;
    };
}
```

```
const p1 = new Product("Laptop", 1000.00, 5);
const p2 = new Product("Headphones", 200.00, 2);

console.log(p1.toString());
console.log(p2.toString());
console.log(p1.name);
console.log(p2.name);
```

24

Funções sem OO e com OO (Javascript)

Exercício:

Queremos acrescentar duas funções para lidar com produtos:

```
function total(product: Product): number
Parâmetros:
  - product: um objeto do tipo Product
Efeito: retorna o valor total do produto (preço multiplicado pela quantidade)
```

```
function updatePrice(product: Product, percentage: number): void
Parâmetros:
  - product: um objeto do tipo Product
  - percentage: um valor de porcentagem
Efeito: atualiza o preço do produto, aumentando-o conforme a porcentagem dada
```

```
class Product {
    constructor(name, price, quantity) {
        this.name = name;
        this.price = price;
        this.quantity = quantity;
    }

    toString() {
        return `${this.name}, $${this.price.toFixed(2)}, ${this.quantity}`;
    }
}
```

```
function total(product) {
    return product.price * product.quantity;
}

function updatePrice(product, percentage) {
    product.price = product.price * (1 + percentage / 100);
}

const p1 = new Product("Laptop", 1000.0, 5);
const p2 = new Product("Headphones", 200.0, 2);

const total1 = total(p1);
const total2 = total(p2);
console.log(total1);
console.log(total2);

updatePrice(p1, 10);
console.log(p1.price);
```

27

```
class Product {
  constructor(name, price, quantity) {
    this.name = name;
    this.price = price;
    this.quantity = quantity;
  }

  total() {
    return this.price * this.quantity;
  }

  updatePrice(percentage) {
    this.price = this.price * (1 + percentage / 100);
  }

  toString() {
    return `${this.name}, $${this.price.toFixed(2)}, ${this.quantity}`;
  }
}
```

```
const p1 = new Product("Laptop", 1000.0, 5);
const p2 = new Product("Headphones", 200.0, 2);

const total1 = p1.total();
const total2 = p2.total();
console.log(total1);
console.log(total2);

p1.updatePrice(10);
console.log(p1.price);
```

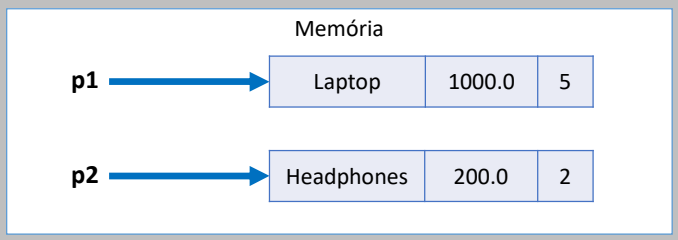
28

Tipos estruturados (classes, atributos) (Java)

Exercício:

Fazer um programa para armazenar em memória os dados de dois produtos (nome, preço e quantidade), depois mostrar esses dados na tela.

```
Exemplo:
Laptop, $1000.00, 5
Headphones, $200.00, 2
```



```java
public class Product {

    public String name;
    public double price;
    public int quantity;

    public Product(String name, double price, int quantity) {
        this.name = name;
        this.price = price;
        this.quantity = quantity;
    }

    @Override
    public String toString() {
        return name + ", $" + String.format("%.2f", price) + ", " + quantity;
    }
}
```

```java
public class Program {
    public static void main(String[] args) {

        var p1 = new Product("Laptop", 1000.00, 5);
        var p2 = new Product("Headphones", 200.00, 2);
        System.out.println(p1);
        System.out.println(p2);
        System.out.println(p1.name);
        System.out.println(p2.name);
    }
}
```



```java
package app;

public class Product {

    private String name;
    private double price;
    private int quantity;

    public Product(String name, double price, int quantity) {
        this.name = name;
        this.price = price;
        this.quantity = quantity;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public double getPrice() {
        return price;
    }

    public void setPrice(double price) {
        this.price = price;
    }

    public int getQuantity() {
        return quantity;
    }

    public void setQuantity(int quantity) {
        this.quantity = quantity;
    }

    @Override
    public String toString() {
        return name + ", $" + String.format("%.2f", price) + ", " + quantity;
    }
}
```

Versão com getters e setters (encapsulamento)

```java
package app;

public class Program {

    public static void main(String[] args) {

        var p1 = new Product("Laptop", 1000.00, 5);
        var p2 = new Product("Headphones", 200.00, 2);
        System.out.println(p1);
        System.out.println(p2);
        System.out.println(p1.getName());
        System.out.println(p2.getName());
    }
}
```


Funções sem OO e com OO (Java)

Exercício:

Queremos acrescentar duas funções para lidar com produtos:

```
function total(product: Product): number
Parâmetros:
  - product: um objeto do tipo Product
Efeito: retorna o valor total do produto (preço multiplicado pela quantidade)
```

```
function updatePrice(product: Product, percentage: number): void
Parâmetros:
  - product: um objeto do tipo Product
  - percentage: um valor de porcentagem
Efeito: atualiza o preço do produto, aumentando-o conforme a porcentagem dada
```



```java
public class Product {

    private String name;
    private double price;
    private int quantity;

    public Product(String name, double price, int quantity) {
        this.name = name;
        this.price = price;
        this.quantity = quantity;
    }

    // getters e setters...

    @Override
    public String toString() {
        return name + ", $" + String.format("%.2f", price) + ", " + quantity;
    }
}
```

```java
public class Program {

    public static double total(Product product) {
        return product.getPrice() * product.getQuantity();
    }

    public static void updatePrice(Product product, double percentage) {
        double newPrice = product.getPrice() * (1.0 + percentage / 100.0);
        product.setPrice(newPrice);
    }

    public static void main(String[] args) {

        var p1 = new Product("Laptop", 1000.00, 5);
        var p2 = new Product("Headphones", 200.00, 2);

        var total1 = total(p1);
        var total2 = total(p2);
        System.out.println(total1);
        System.out.println(total2);

        updatePrice(p1, 10);
        System.out.println(p1.getPrice());
    }
}
```

37

```java
public class Product {

    private String name;
    private double price;
    private int quantity;

    public Product(String name, double price, int quantity) {
        this.name = name;
        this.price = price;
        this.quantity = quantity;
    }

    // getters e setters...

    public double total() {
        return price * quantity;
    }

    public void updatePrice(double percentage) {
        price = price * (1.0 + percentage / 100.0);
    }

    @Override
    public String toString() {
        return name + ", $" + String.format("%.2f", price) + ", " + quantity;
    }
}
```

38

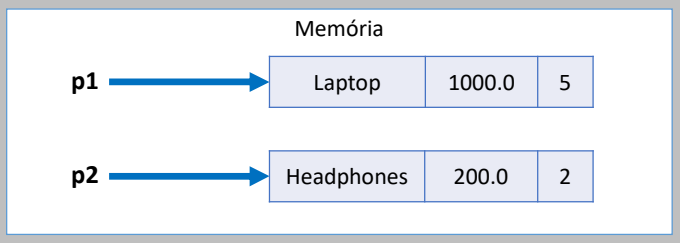
```java
public class Program {

    public static void main(String[] args) {

        var p1 = new Product("Laptop", 1000.00, 5);
        var p2 = new Product("Headphones", 200.00, 2);

        var total1 = p1.total();
        var total2 = p2.total();
        System.out.println(total1);
        System.out.println(total2);

        p1.updatePrice(10);
        System.out.println(p1.getPrice());
    }
}
```


Tipos estruturados (classes, atributos) (C#)

Exercício:

Fazer um programa para armazenar em memória os dados de dois produtos (nome, preço e quantidade), depois mostrar esses dados na tela.

Exemplo:
Laptop, $1000.00, 5
Headphones, $200.00, 2




```
namespace Course
{
    class Product
    {

        public string Name { get; set; }
        public double Price { get; set; }
        public int Quantity { get; set; }

        public Product(string name, double price, int quantity)
        {
            Name = name;
            Price = price;
            Quantity = quantity;
        }

        public override string ToString()
        {
            return Name + ", $" + Price.ToString("F2") + ", " + Quantity;
        }
    }
}
```

```
namespace Course
{
    class Program
    {
        static void Main(string[] args)
        {
            var p1 = new Product("Laptop", 1000.00, 5);
            var p2 = new Product("Headphones", 200.00, 2);
            Console.WriteLine(p1);
            Console.WriteLine(p2);
            Console.WriteLine(p1.Name);
            Console.WriteLine(p2.Name);
        }
    }
}
```


Funções sem OO e com OO (C#)

Exercício:

Queremos acrescentar duas funções para lidar com produtos:

```
function total(product: Product): number
Parâmetros:
  - product: um objeto do tipo Product
Efeito: retorna o valor total do produto (preço multiplicado pela quantidade)
```

```
function updatePrice(product: Product, percentage: number): void
Parâmetros:
  - product: um objeto do tipo Product
  - percentage: um valor de porcentagem
Efeito: atualiza o preço do produto, aumentando-o conforme a porcentagem dada
```



```csharp
namespace Course
{
    class Product
    {

        public string Name { get; set; }
        public double Price { get; set; }
        public int Quantity { get; set; }

        public Product(string name, double price, int quantity)
        {
            Name = name;
            Price = price;
            Quantity = quantity;
        }

        public override string ToString()
        {
            return Name + ", $" + Price.ToString("F2") + ", " + Quantity;
        }
    }
}
```

```
namespace Course
{
    class Program
    {
        static double total(Product product)
        {
            return product.Price * product.Quantity;
        }

        static void updatePrice(Product product, double percentage)
        {
            product.Price = product.Price * (1.0 + percentage / 100.0);
        }

        static void Main(string[] args)
        {
            var p1 = new Product("Laptop", 1000.00, 5);
            var p2 = new Product("Headphones", 200.00, 2);

            var total1 = total(p1);
            var total2 = total(p2);
            Console.WriteLine(total1);
            Console.WriteLine(total2);

            updatePrice(p1, 10);
            Console.WriteLine(p1.Price);
        }
    }
}
```

47

```
using System.Globalization;

namespace Course
{
    class Product
    {

        public string Name { get; set; }
        public double Price { get; set; }
        public int Quantity { get; set; }

        public Product(string name, double price, int quantity)
        {
            Name = name;
            Price = price;
            Quantity = quantity;
        }

        public double total()
        {
            return Price * Quantity;
        }

        public void updatePrice(double percentage)
        {
            Price = Price * (1.0 + percentage / 100.0);
        }

        public override string ToString()
        {
            return Name + ", $" + Price.ToString("F2", CultureInfo.InvariantCulture) + ", " + Quantity;
        }
    }
}
```

48

```
namespace Course
{
    class Program
    {
        static void Main(string[] args)
        {
            var p1 = new Product("Laptop", 1000.00, 5);
            var p2 = new Product("Headphones", 200.00, 2);

            var total1 = p1.total();
            var total2 = p2.total();
            Console.WriteLine(total1);
            Console.WriteLine(total2);

            p1.updatePrice(10);
            Console.WriteLine(p1.Price);
        }
    }
}
```


Tipos estruturados (classes, atributos) (Python)

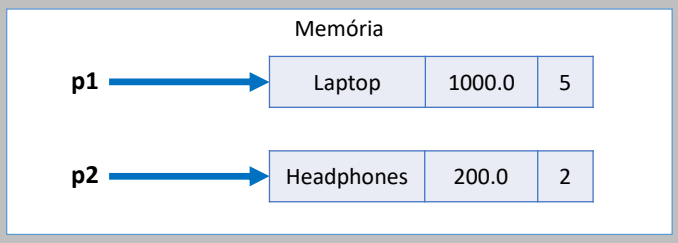
Exercício:

Fazer um programa para armazenar em memória os dados de dois produtos (nome, preço e quantidade), depois mostrar esses dados na tela.

Exemplo:

Laptop, $1000.00, 5

Headphones, $200.00, 2




```python
class Product:
    def __init__(self, name, price, quantity):
        self.name = name
        self.price = price
        self.quantity = quantity

    def __str__(self):
        return f"{self.name}, ${self.price:.2f}, {self.quantity}"
```

```python
p1 = Product("Laptop", 1000.00, 5)
p2 = Product("Headphones", 200.00, 2)

print(p1)
print(p2)
print(p1.name)
print(p2.name)
```

Funções sem OO e com OO (Python)


```python
class Product:
    def __init__(self, name, price, quantity):
        self.name = name
        self.price = price
        self.quantity = quantity

    def __str__(self):
        return f"{self.name}, ${self.price:.2f}, {self.quantity}"
```

```python
def total(product):
    return product.price * product.quantity


def update_price(product, percentage):
    product.price = product.price * (1 + percentage / 100)


p1 = Product("Laptop", 1000.00, 5)
p2 = Product("Headphones", 200.00, 2)

total1 = total(p1)
total2 = total(p2)
print(total1)
print(total2)

update_price(p1, 10)
print(p1.price)
```

```python
class Product:
    def __init__(self, name, price, quantity):
        self.name = name
        self.price = price
        self.quantity = quantity

    def total(self):
        return self.price * self.quantity

    def update_price(self, percentage):
        self.price = self.price * (1 + percentage / 100)

    def __str__(self):
        return f"{self.name}, ${self.price:.2f}, {self.quantity}"
```

```python
p1 = Product("Laptop", 1000.00, 5)
p2 = Product("Headphones", 200.00, 2)
totalPrice = p1.total() + p2.total()

p1 = Product("Laptop", 1000.00, 5)
p2 = Product("Headphones", 200.00, 2)

total1 = p1.total()
total2 = p2.total()
print(total1)
print(total2)

p1.update_price(10)
print(p1.price)
```

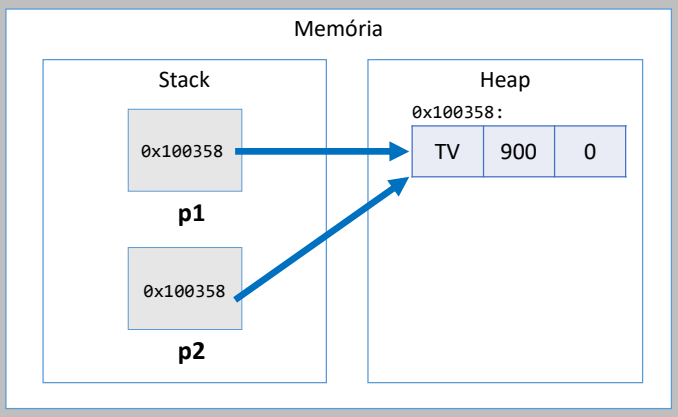

Tipos referência vs. tipos valor (Java)

Classes são tipos referência

Variáveis cujo tipo são classes não devem ser entendidas como caixas, mas sim "tentáculos" (ponteiros) para caixas

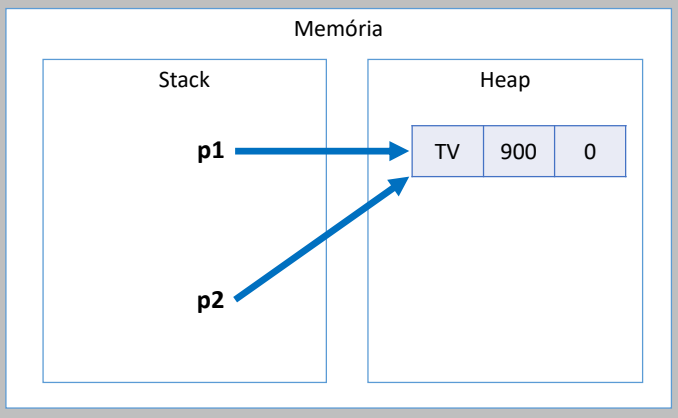
```
Product p1, p2;

p1 = new Product("TV", 900.00, 0);

p2 = p1;
```

p2 = p1;
"p2 passa a apontar para onde p1 aponta"



Desenho simplificado

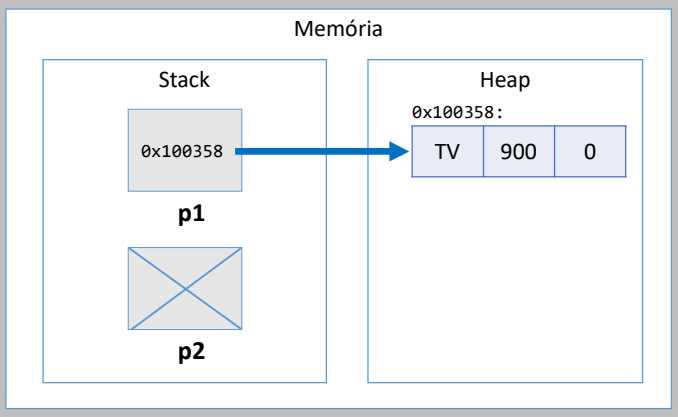
```
Product p1, p2;

p1 = new Product("TV", 900.00, 0);

p2 = p1;
```



Valor "null"

Tipos referência aceitam o valor "null", que indica que a variável aponta pra ninguém.

```
Product p1, p2;

p1 = new Product("TV", 900.00, 0);

p2 = null;
```

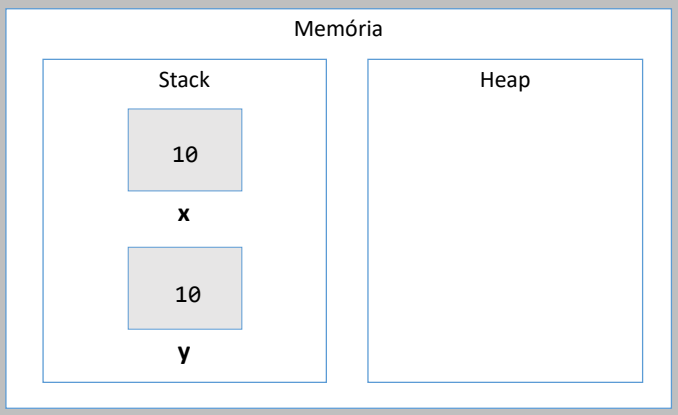


Tipos primitivos são tipos valor

Em Java, tipos primitivos são tipos valor. Tipos valor são CAIXAS e não ponteiros.

```
double x, y;

x = 10;

y = x;
```

y = x;
"y recebe uma CÓPIA de x"



Type	Contains	Default	Size	Range
`boolean`	`true` or `false`	`false`	1 bit	NA
`char`	Unicode character	`\u0000`	16 bits	`\u0000` to `\uFFFF`
`byte`	Signed integer	0	8 bits	-128 to 127
`short`	Signed integer	0	16 bits	-32768 to 32767
`int`	Signed integer	0	32 bits	-2147483648 to 2147483647
`long`	Signed integer	0	64 bits	-9223372036854775808 to 9223372036854775807
`float`	IEEE 754 floating point	0.0	32 bits	±1.4E-45 to ±3.4028235E+38
`double`	IEEE 754 floating point	0.0	64 bits	±4.9E-324 to ±1.7976931348623157E+308

Tipos primitivos e inicialização

- Demo:

```
int p;
System.out.println(p); // erro: variável não iniciada

p = 10;
System.out.println(p);
```

Valores padrão

- Quando alocamos (new) qualquer tipo estruturado (classe ou array), são atribuídos valores padrão aos seus elementos
  - números: 0
  - boolean: false
  - char: caractere código 0
  - objeto: null

```
Product p = new Product();
```

Tipos referência vs. tipos valor

CLASSE	**TIPO PRIMITIVO**
Vantagem: usufrui de todos recursos OO	Vantagem: é mais simples e mais performático
Variáveis são ponteiros	Variáveis são caixas
Objetos precisam ser instanciados usando new, ou apontar para um objeto já existente.	Não instancia. Uma vez declarados, estão prontos para uso.
Aceita valor null	Não aceita valor null
Y = X; "Y passa a apontar para onde X aponta"	Y = X; "Y recebe uma cópia de X"
Objetos instanciados no heap	"Objetos" instanciados no stack
Objetos não utilizados são desalocados em um momento próximo pelo garbage collector	"Objetos" são desalocados imediatamente quando seu escopo de execução é finalizado

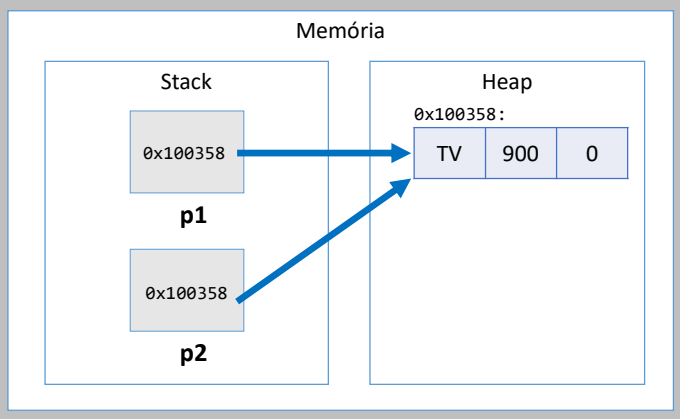
Tipos referência vs. tipos valor (C#)

Classes são tipos referência

Variáveis cujo tipo são classes não devem ser entendidas como caixas, mas sim "tentáculos" (ponteiros) para caixas

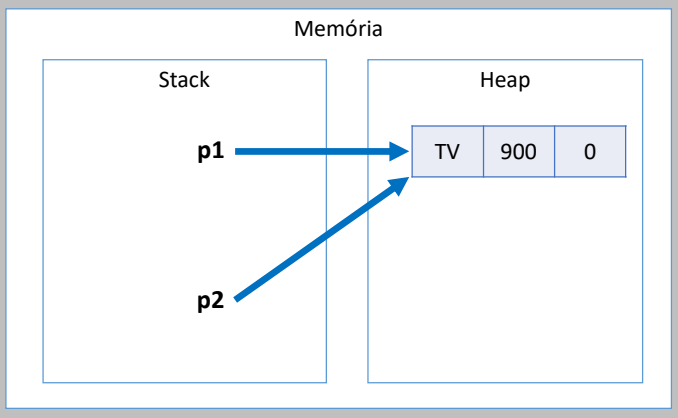
```
Product p1, p2;

p1 = new Product("TV", 900.00, 0);

p2 = p1;
```

p2 = p1;
"p2 passa a apontar para onde p1 aponta"



Desenho simplificado

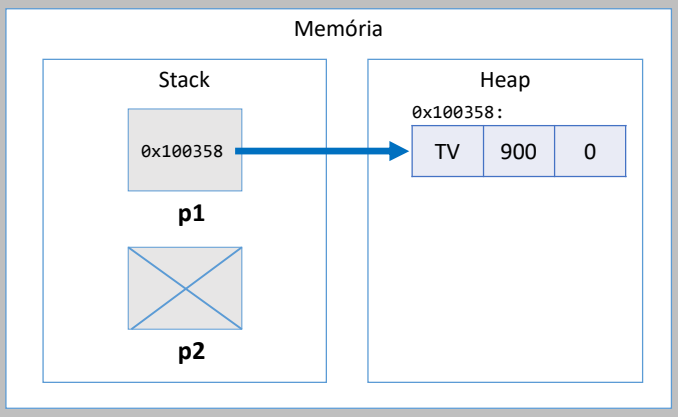
```
Product p1, p2;

p1 = new Product("TV", 900.00, 0);

p2 = p1;
```



Valor "null"

Tipos referência aceitam o valor "null", que indica que a variável aponta pra ninguém.

```
Product p1, p2;

p1 = new Product("TV", 900.00, 0);

p2 = null;
```

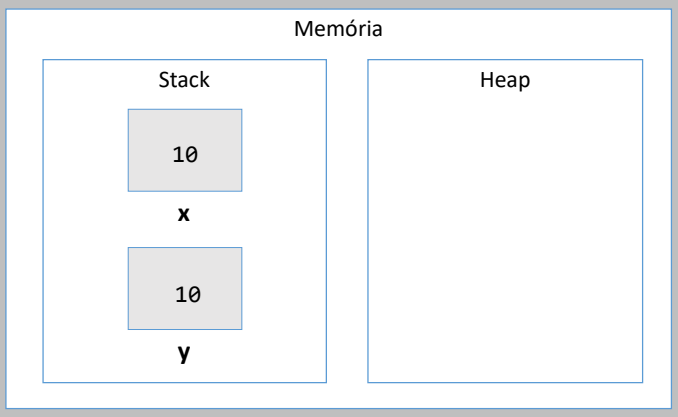


Structs são tipos valor

A linguagem C# possui também tipos valor, que são os "structs". Structs são CAIXAS e não ponteiros.

```
double x, y;

x = 10;

y = x;
```

y = x;
"y recebe uma CÓPIA de x"



69

C# Type	.Net Framework Type	Signed	Bytes	Possible Values
sbyte	System.Sbyte	Yes	1	-128 to 127
short	System.Int16	Yes	2	-32768 to 32767
int	System.Int32	Yes	4	-2^{31} to $2^{31}-1$
long	System.Int64	Yes	8	-2^{63} to $2^{63}-1$
byte	System.Byte	No	1	0 to 255
ushort	System.Uint16	No	2	0 to 65535
uint	System.Uint32	No	4	0 to $2^{32}-1$
ulong	System.Uint64	No	8	0 to $2^{64}-1$
float	System.Single	Yes	4	$\pm 1.5 \times 10^{-45}$ to $\pm 3.4 \times 10^{38}$ with 7 significant figures
double	System.Double	Yes	8	$\pm 5.0 \times 10^{-324}$ to $\pm 1.7 \times 10^{308}$ with 15 or 16 significant figures
decimal	System.Decimal	Yes	12	$\pm 1.0 \times 10^{-28}$ to $\pm 7.9 \times 10^{28}$ with 28 or 29 significant figures
char	System.Char	N/A	2	Any Unicode character
bool	System.Boolean	N/A	1/2	true or false

Outros structs importantes: DateTime, TimeSpan

70

É possível criar seus próprios structs

```
namespace Course {
    struct Point {

        public double X, Y;

        public override string ToString() {
            return "(" + X + "," + Y + ")";
        }
    }
}
```


Structs e inicialização

- Demo:

```
Point p;
Console.WriteLine(p); // erro: variável não atribuída
p.X = 10;
p.Y = 20;
Console.WriteLine(p);
p = new Point();
Console.WriteLine(p);
```

Valores padrão

- Quando alocamos (new) qualquer tipo estruturado (classe, struct, array), são atribuídos valores padrão aos seus elementos
  - números: 0
  - bool: False
  - char: caractere código 0
  - objeto: null
- Lembrando: uma variável apenas declarada, mas não instanciada, inicia em estado "não atribuída", e o próprio compilador não permite que ela seja acessada.

Tipos referência vs. tipos valor

CLASSE	**STRUCT**
Vantagem: usufrui de todos recursos OO	Vantagem: é mais simples e mais performático
Variáveis são ponteiros	Variáveis são caixas
Objetos precisam ser instanciadas usando new, ou apontar para um objeto já existente.	Não é preciso instanciar usando new, mas é possível
Aceita valor null	Não aceita valor null
Suporte a herança	Não tem suporte a herança (mas pode implementar interfaces)
Y = X; "Y passa a apontar para onde X aponta"	Y = X; "Y recebe uma cópia de X"
Objetos instanciados no heap	Objetos instanciados no stack
Objetos não utilizados são desalocados em um momento próximo pelo garbage collector	"Objetos" são desalocados imediatamente quando seu escopo de execução é finalizado

Tipos referência vs. tipos valor (Javascript)

Tipos primitivos em Javascript

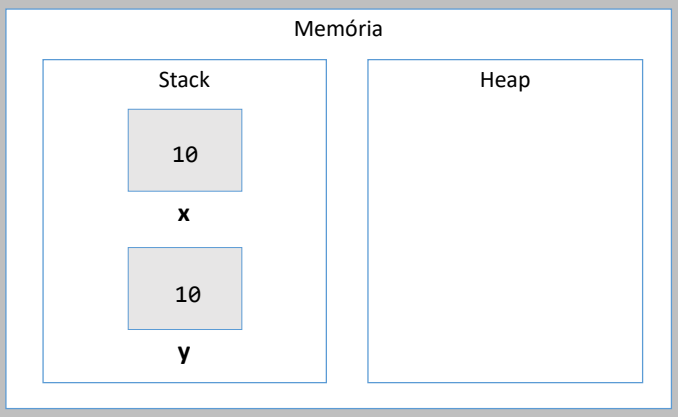
- **String**: sequência de caracteres.
- **Number**: números, inteiros ou de ponto flutuante.
- **Boolean**: verdadeiro ou falso.
- **BigInt**: números inteiros de “precisão arbitrária”.
- **undefined**: tipo/valor indefinido ou não atribuído.
- **null**: valor que representa uma referência que não aponta para ninguém.
- **Symbol**: Introduzido no ECMAScript 6, representa um identificador único e é frequentemente usado para propriedades de objetos que devem ser únicas.

```
let name = "Maria";
let age = 42;
let married = true;
let property = Symbol("prop");
```

Tipos primitivos (tipos valor)

```
let x = 10;

let y = x;
```

y = x;
"y recebe uma CÓPIA de x"



Quase tudo em Javascript é objeto!

Cuidado com a pegadinha!
Em Javascript, todos os tipos primitivos - com exceção de null e undefined - **são tratados como** objetos.

```
10.567.toFixed(2)

"Maria".length
```

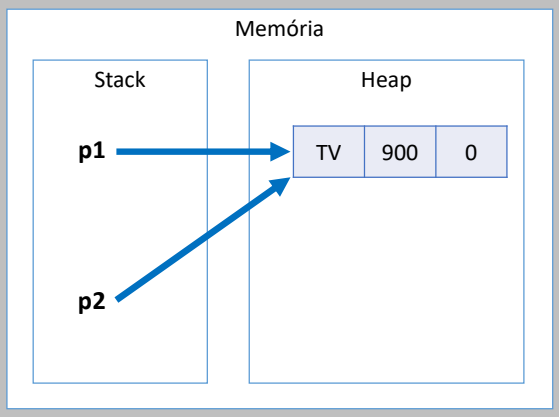
As operações acima fazem um “auto-boxing” do valor, criando temporariamente um objeto no heap, que faz a chamada do método ou propriedade.

Tipos referência

Em Javascript, tipos referência são **objetos**, **arrays** e **funções**.

```
const Product = function(name, price, quantity) {
    this.name = name;
    this.price = price;
    this.quantity = quantity;
}

let p1 = new Product("TV", 900.0, 0);

let p2 = p1;
```

p2 = p1;
"p2 passa a apontar para onde p1 aponta"



Links

https://www.javascripttutorial.net/javascript-primitive-vs-reference-values

https://developer.mozilla.org/en-US/docs/Glossary/Primitive

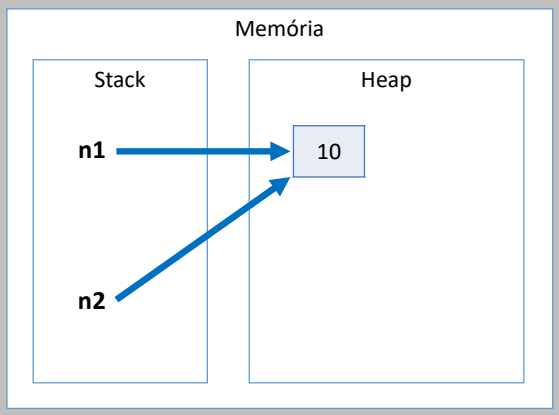
Tipos referência vs. tipos valor (Python)

Tipos referência

Todos tipos em Python são tipos referência. Qualquer que seja o tipo da variável, será instanciado um objeto no Heap.

```
n1 = 10;

n2 = n1;
```

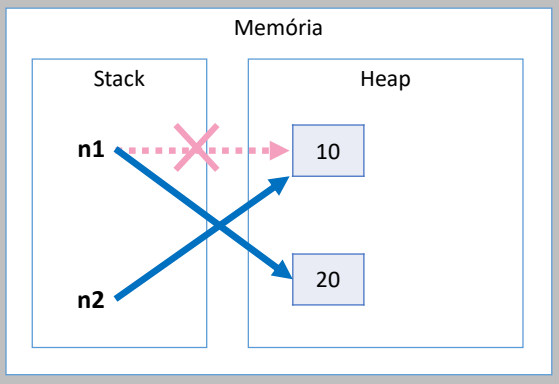
n2 = n1;
"n2 passa a apontar para onde n1 aponta"



Tipos imutáveis

Cuidado com a pegadinha! Os tipos numéricos (int, float, complex), str, bool, tuple e frozenset em Python são imutáveis. Isso significa que uma nova atribuição a variáveis desse tipo vai na verdade instanciar um novo objeto.

```
n1 = 10;

n2 = n1;

n1 = 20;
```



Links

https://stackoverflow.com/questions/6158907/what-does-python-treat-as-reference-types

https://docs.python.org/3/library/stdtypes.html

Desalocação de memória - garbage collector e escopo local

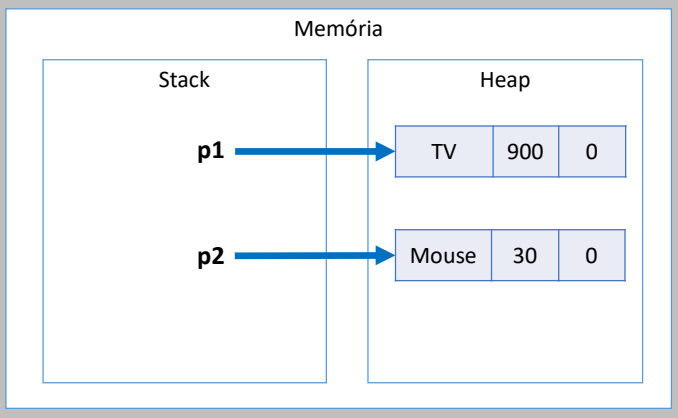
(este fundamento vale para todas linguagens)

Garbage collector

- É um processo que automatiza o gerenciamento de memória de um programa em execução
- O garbage collector monitora os objetos alocados dinamicamente pelo programa (no heap), desalocando aqueles que não estão mais sendo utilizados.

Desalocação por garbage collector

```
Product p1, p2;

p1 = new Product("TV", 900.00, 0);

p2 = new Product("Mouse", 30.00, 0);
```

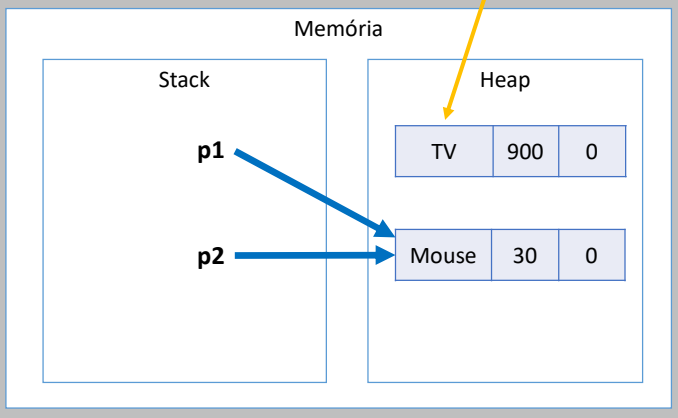


Desalocação por garbage collector

Será desalocado pelo garbage collector

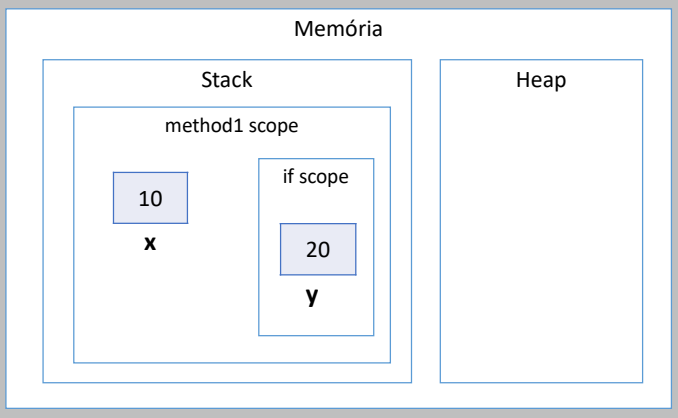
```
Product p1, p2;

p1 = new Product("TV", 900.00, 0);

p2 = new Product("Mouse", 30.00, 0);
```

```
p1 = p2;
```



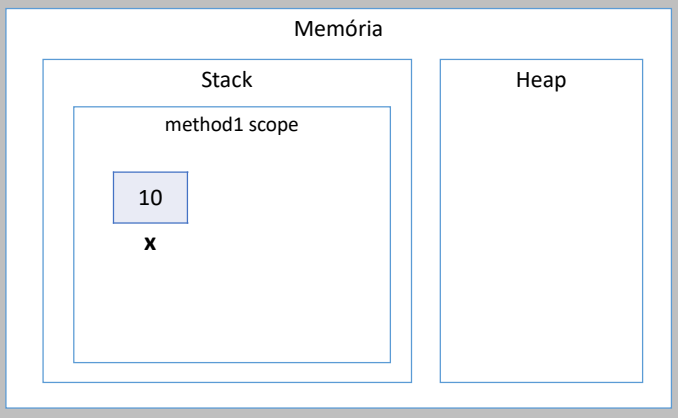
Desalocação por escopo

```
void method1() {
    int x = 10;
    if (x > 0) {
        int y = 20;
    }
    System.out.println(x);
}
```



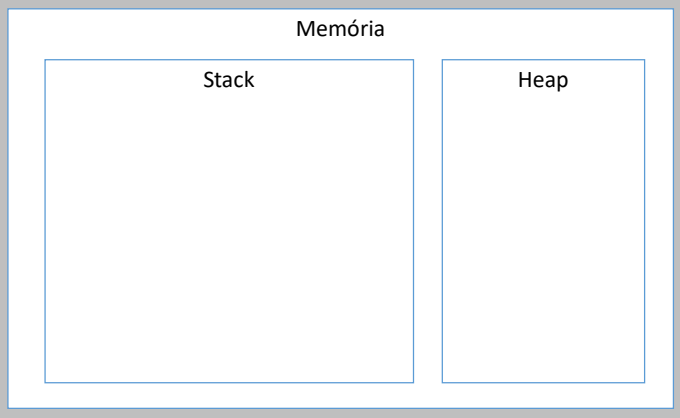
Desalocação por escopo

```
void method1() {
    int x = 10;
    if (x > 0) {
        int y = 20;
    }
    System.out.println(x);
}
```



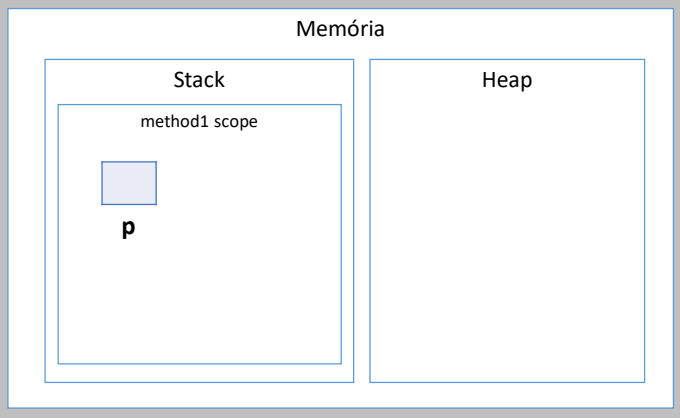
Desalocação por escopo

```
void method1() {
    int x = 10;
    if (x > 0) {
        int y = 20;
    }
    System.out.println(x);
}
```



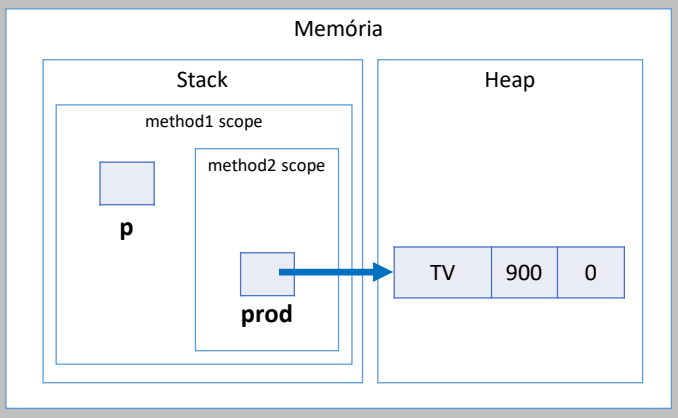
Outro exemplo

```
void method1() {
  Product p = method2();
  System.out.println(p.Name);
}

Product method2() {
  Product prod = new Product("TV", 900.0, 0);
  return prod;
}
```



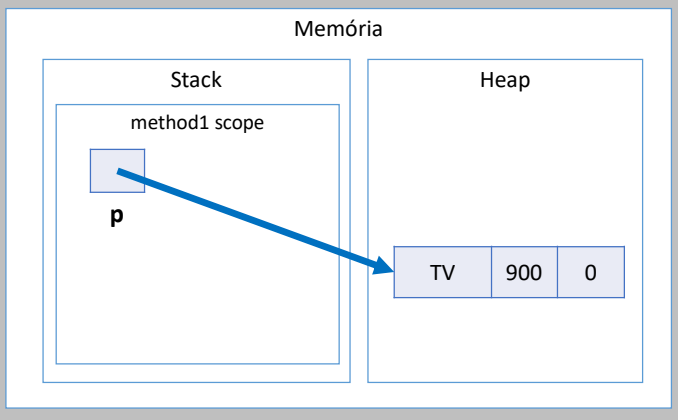
Outro exemplo

```
void method1() {
  Product p = method2();
  System.out.println(p.Name);
}

Product method2() {
  Product prod = new Product("TV", 900.0, 0);
  return prod;
}
```



Outro exemplo

```
void method1() {
  Product p = method2();
  System.out.println(p.Name);
}

Product method2() {
  Product prod = new Product("TV", 900.0, 0);
  return prod;
}
```



Resumo

- Objetos alocados dinamicamente, quando não possuem mais referência para eles, serão desalocados pelo garbage collector
- Variáveis locais são desalocadas imediatamente assim que seu escopo local sai de execução