Introdução à Programação Orientada a Objetos em Java

I. Introdução à Linguagem de Programação Java

Breve Histórico

- 1995 – Um dos criadores: James Gosling (Sun);
- Origens: Oak - Linguagem para controle de dispositivos;
- Com o nascimento dos Browsers foi identificado a necessidade de se ter uma linguagem multiplataforma para estender as capacidades dos Navegadores;
- Oak foi rebatizada Java.

Evolução

- Sucesso como linguagem de programação;
- Sucesso para aplicações em Servidores Web;
- Sucesso limitado para aplicações em desktops e web browsers;
- Disputa com Python a liderança da linguagem de programação mais utilizada.

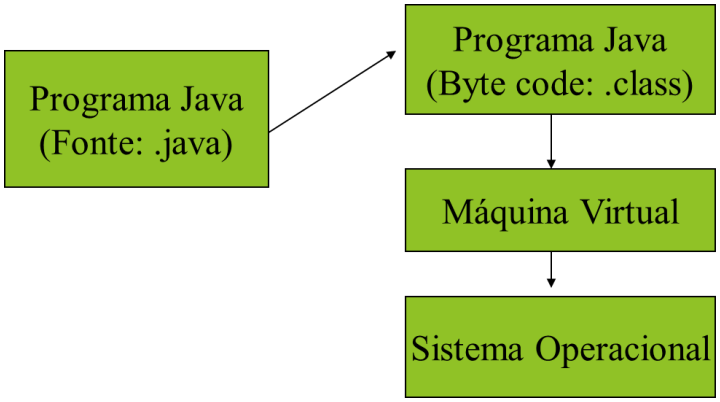
Características

Java é uma linguagem de uso geral:

- Multiplataforma;
- 100% orientada a objetos;
- Segurança de uma linguagem compilada;
- Conjunto de classes rico em funcionalidades: GUI, Networking, Acesso a Banco de Dados, Imagens, Multimídia, Aplicações em servidores WWW, aplicações de pequeno, médio e grande porte;
- Possui coleta automática de lixo de memória;
- 100% gratuita;
- Parecida com C e C++;
- Possui características herdadas de muitas outras linguagens de programação: Objective-C, Smalltalk, Eiffel, Modula-3, etc;
- É uma feliz união de tecnologias testadas por vários centros de pesquisa e desenvolvimento de software;
- Compilada: Um programa em Java é compilado para o chamado *bytecode*, que é um código para ser executado por uma máquina virtual, idealizada pelos criadores da linguagem. Por isso Java pode ser mais rápida do que se fosse simplesmente interpretada.

- Portável: O *bytecode* gerado pelo compilador pode ser transportado entre plataformas distintas que suportam Java (Solaris, Windows, Linux, Mac/Os etc.). Não é necessário recompilar um programa para que ele rode numa máquina e/ou sistema diferente.
- Orientada a Objetos: A portabilidade é uma das características de uma linguagem orientada a objetos. Java suporta herança e interfaces, onde uma classe herda o comportamento de sua superclasse, além de oferecer uma implementação para uma ou mais interfaces. Java permite ainda a criação de classes abstratas.
- Linguagem Segura:
  - A coleta automática de lixo, evita erros comuns que os programadores cometem quando são obrigados a gerenciar diretamente a memória (C,C++, Pascal);
  - A eliminação do uso de ponteiros, em favor do uso de vetores, objetos e outras estruturas traz benefícios em termos de segurança, pois o programador é proibido de obter acesso a memória que não pertença ao seu programa;
  - Ser *strongly typed* (fortemente tipada) também é uma vantagem em termos de segurança, que está aliada a eliminação de conversões implícitas de tipos como em C++;
  - A presença de mecanismos de tratamento de exceções torna as aplicações mais robustas, não permitindo que elas abortem, mesmo quando rodando sob condições anormais.
- Suporta concorrência:
  - A linguagem permite a criação de maneira fácil, de vários "threads" de execução. Este tópico será útil quando você estudar animações, e é particularmente poderoso nos ambientes em que aplicações Java são suportadas, ambientes estes que geralmente podem mapear os *threads* da linguagem em processamento paralelo real;
  - Suporte para programação de sistemas distribuídos: fornece facilidades para programação com *sockets*, *remote method call*, TCP-IP etc.
- Eficiente:
  - Como Java foi criada para ser usada em computadores pequenos, ela exige pouco espaço e pouca memória. Java é muito mais eficiente que grande parte das linguagens de *scripting* existentes, embora seja cerca de 20 vezes mais lenta que C, o que não é um marco definitivo.
  - Além disso, hoje em dia, Java já permite gerar código executável de uma particular arquitetura *on the fly*, tudo a partir do *bytecode*.

Arquitetura



O código fonte de um programa em Java (.java) ao ser compilado, gera um arquivo em *bytecode* (.class) que pode ser executado em qualquer máquina virtual Java e em qualquer equipamento que possua um sistema operacional que rode essa VM. A própria VM é quem se encarrega de interpretar o *bytecode* e se comunicar com o SO.

Máquina Virtual (JVM)

- A *Java Virtual Machine* (JVM) é uma parte fundamental do ambiente de execução, sendo responsável por interpretar e executar o *bytecode* gerado a partir do código-fonte;
- Atua como uma camada intermediária entre o código Java e o sistema operacional, convertendo-o em *bytecode* compreensível pela própria máquina virtual;
- Oferece interpretação em tempo real e gerenciamento automático de memória;
- Possui independência do Sistema Operacional;
- Gerencia a alocação e liberação de memória, otimizando o uso dos recursos disponíveis;
- Realiza *just-in-time compilation* (JIT), convertendo o *bytecode* em código de máquina conforme necessário, para melhorar o desempenho;
- Possui: portabilidade, gerenciamento de memória e segurança;
- Continua a evoluir com novas versões, melhorias de desempenho e suporte a recursos modernos;
- Versões recentes incluem recursos como: módulos, *record types* e var.

II. Fundamentos da Programação Orientada a Objetos

O que é um paradigma de programação?

Um paradigma é a abordagem utilizada para resolver problemas por meio da programação de computadores. Existem vários paradigmas, incluindo o imperativo, declarativo, estruturado e orientado a objetos.

O que é o paradigma de POO?

A POO é um modelo de análise, projeto e programação que se baseia na aproximação entre o mundo real e o mundo virtual. Ela envolve a criação e interação entre objetos, atributos, códigos e métodos.

Quais são os 4 Pilares básicos da POO?

Encapsulamento: Oculta detalhes internos de um objeto, permitindo que ele seja usado sem se conhecer sua implementação.

Herança: Permite criar classes a partir de classes existentes, reutilizando código e estabelecendo hierarquias.

Polimorfismo: Permite que objetos de diferentes classes sejam tratados de forma uniforme, facilitando a flexibilidade e extensibilidade do código.

Abstração: Representa objetos do mundo real de forma simplificada, focando nos aspectos relevantes.

Quais são as vantagens de usar a POO?

Reusabilidade: Código pode ser reutilizado em diferentes partes do programa.

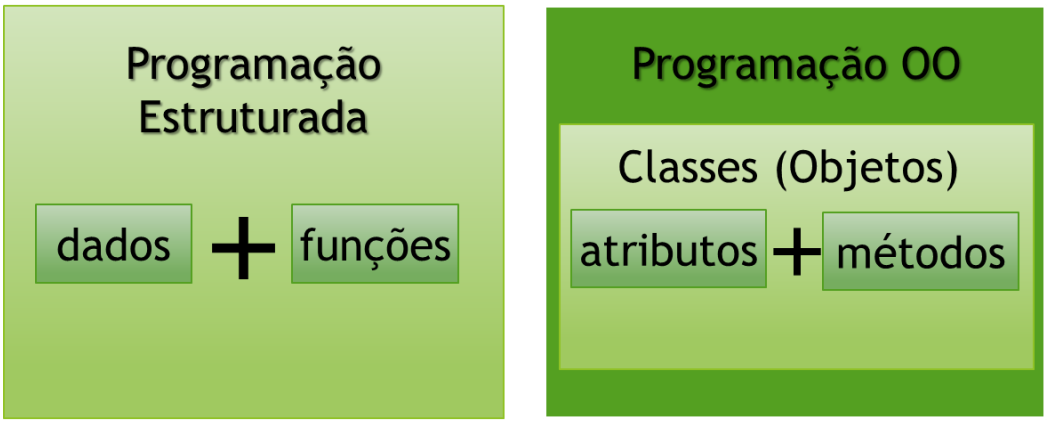
Manutenção: Mudanças em uma classe não afetam outras partes do código.

Organização: Classes e objetos tornam o código mais organizado e modular.

Que linguagens de programação utilizam POO?

A maioria das linguagens modernas, como Java, Python, C#, Ruby e C++, são baseadas em POO.

Programação: Estruturada X OO



O que é um objeto?

Um objeto é um elemento computacional que representa, no domínio da solução, alguma entidade (abstrata ou concreta) do domínio de interesse do problema sob análise.

Objetos similares são agrupados em classes.

No paradigma da orientação a objetos, tudo pode ser potencialmente representado como um objeto.

Sob o ponto de vista da programação orientada a objetos, um objeto não é muito diferente de uma variável no paradigma de programação convencional.

Variáveis X Objetos

Para fins didáticos, podemos fazer uma analogia entre uma variável de memória e um objeto:

Por exemplo, quando se define uma variável do tipo *int* em Java, essa variável possui:

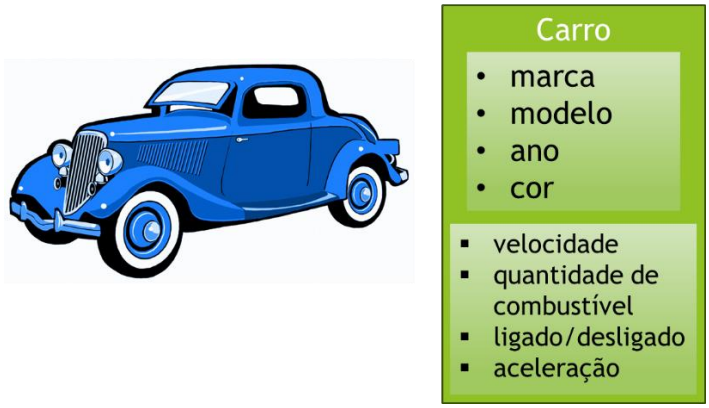
- um espaço em memória para registrar o seu estado atual (um valor);
- um nome (identificador) que a define e serve para "localizá-la" na memória;
- um conjunto de operações associadas que podem ser aplicadas a ela, através dos operadores definidos na linguagem que podem ser aplicados a valores inteiros (soma, subtração, inversão de sinal, multiplicação, divisão inteira, resto da divisão inteira, incremento, decremento etc.).

Objeto

Agrupamento de dados e procedimentos que atuam sobre os dados.

Atributos: descrevem as propriedades do objeto.

Métodos: operações que descrevem ou modificam o comportamento do objeto.



Toda linguagem de programação orientada a objetos oferece mecanismos para definir os tipos de objetos para cada aplicação, através do conceito de classes.

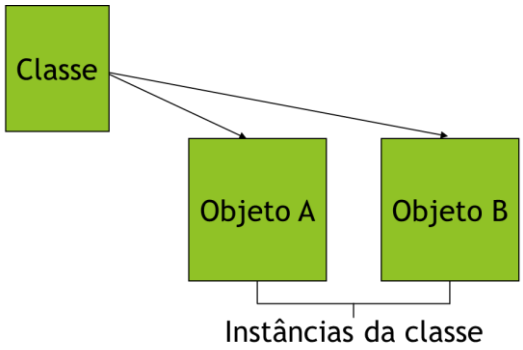
Objetos são instâncias de classes. Essas instâncias precisam ser criadas para que, através da sua manipulação, possam realizar seu trabalho.

Após a conclusão das suas atividades, objetos podem ser removidos, caso necessário.

Classe

As linguagens de programação orientada a objetos (POO) permitem ao programador definir seus tipos de dados através da composição de outros tipos de dados e operações (métodos). O Java suporta a definição de classes, herança, interfaces, polimorfismo, encapsulamento e outros conceitos fundamentais da POO.

Uma Classe em POO representa um gabarito para objetos.



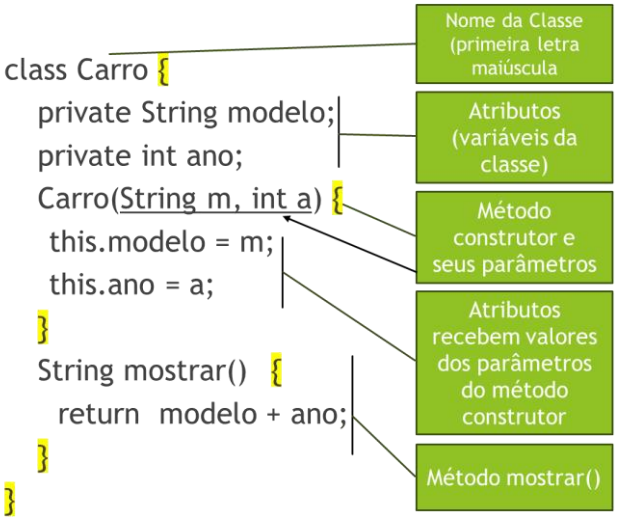
Uma Classe é basicamente a definição de um modelo conceitual para o domínio da aplicação:

- Categoria geral (abstração) de objeto que descreve um conjunto de objetos específicos similares;
- Uma estrutura de dados que contém métodos e atributos.
- Um gabarito para a definição de objetos;
- Através da definição de uma classe, descreve-se que propriedades ou atributos o objeto terá;
- Além da especificação de atributos, descreve também qual o comportamento dos seus objetos, ou seja, que funcionalidades podem ser aplicadas a eles:
  - Essas funcionalidades são descritas através de métodos. Um método nada mais é que o equivalente a um procedimento ou função em outros paradigmas de programação, com a restrição que ele manipula apenas suas variáveis locais e os atributos da classe em que está inserido.
- Dentro do corpo de uma classe:
  - Uma variável recebe o nome de propriedade ou atributo;
  - Uma função recebe o nome de método.

Classes X Objetos

Objetos são instâncias de classes, que determinam qual informação conterá e como poderá manipulá-la. Um programa desenvolvido com uma linguagem de programação orientada a objetos manipula estruturas de dados através dos objetos, da mesma forma que um programa em linguagem tradicional utiliza variáveis.

Assim, classe é uma estrutura que define um tipo de dado composto. No Java uma classe é definida usando a palavra-chave class seguida pelo nome da classe e um par de chaves, veja no exemplo a seguir:



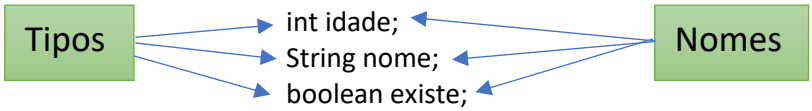
Dentro das chaves fica o corpo da classe e pode conter propriedades, métodos e construtores:

- Propriedade: é uma variável definida no corpo da classe.

III. Tipos de Dados e Instâncias

Declarando variáveis

As declarações de variáveis consistem em um tipo e um nome de variável, como no exemplo a seguir:



Os nomes de variáveis **podem** começar com uma letra, um sublinhado (_), ou um cifrão ($). Elas **não podem** começar com um número. Depois do primeiro caractere pode-se colocar qualquer letra ou número.

Atribuição

Para se atribuir valores às variáveis previamente criadas, utiliza-se o atribuidor “=”, como pode ser visto nos exemplos a seguir:

```
idade = 18;
nome = “Fulano”;
existe = true;
```

Tipos primitivos de dados

Tipo	Ocupa (Bytes)	Faixa de valores
byte	1	-128 a 127
short	2	-32.768 a 32.767
int	4	-2.147.483.648 a 2.147.483.647
long	8	-9x10e18 a 9x10e18
float	4	$-3x10^{38}$ a $3x10^{38}$ (com 6 dígitos significativos)
double	8	$-1,7x10^{308}$ a $1,7x10^{308}$ (com 15 dígitos significativos)
char	2	Utiliza padrão unicode, tem como subconjunto o ASCII
boolean	1	True ou False
String	-	Coleção de caracteres. Não é considerado um tipo primitivo e sim por referência

Tipagem estática: O Java é uma linguagem de programação de tipagem estática, o que significa que as variáveis têm um tipo definido em tempo de compilação (mais precisamente ao criar a variável) e não podem ser alteradas posteriormente. Tome como exemplo os códigos a seguir. A variável "nro" tem o tipo de dado "int" definido na declaração e não poderá receber conteúdos de outros tipos de dados:

```
// int é o tipo de dado da variável
int number;
nro = 12;
// a atribuição a seguir dará o seguinte erro:
// o tipo 'String' não é atribuível ao tipo 'int'
nro = "12";
```

O Java adiciona a capacidade de definir tipos para variáveis, parâmetros de função, retornos de função e outros elementos do código, o que pode ajudar a detectar erros em tempo de desenvolvimento. No exemplo a seguir a função "dif" só aceita parâmetros do tipo "int". O editor Java mostrará a mensagem de erro ao digitar, ou seja, não será necessário executar para obter o erro ao fazer a atribuição usando o tipo String:

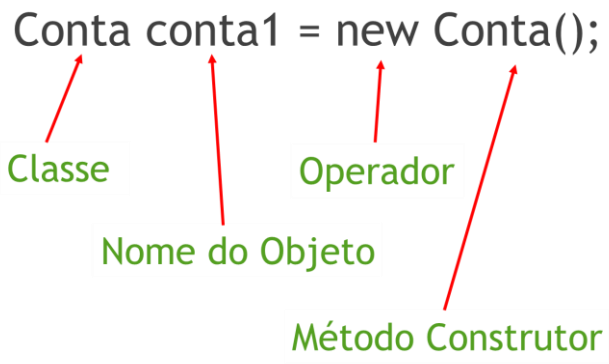
```
// as variáveis a e b podem receber somente valores do tipo int
function dif(int a, int b){
      return a != b;
}
boolean number = dif(2,3);
// os valores passados a seguir como parâmetros
// dará erro por serem do tipo String
boolean number = dif("1","2");
```

Revisando alguns conceitos de POO

- **Classe** – Um modelo para um objeto, que contém variáveis (atributos), para descrever o objeto e métodos para descrever como o objeto se comporta. As classes podem herdar atributos e métodos de outras classes.
- **Objeto** – Uma instância de uma classe. Vários objetos que são instância da mesma classe têm acesso aos mesmos métodos, mas normalmente possuem valores diferentes para seus atributos de instância.

- **Instância** – O mesmo que um objeto. Cada objeto é uma nova ocorrência ou instância de alguma classe.
- **Método** – Um grupo de instruções em uma classe, que define como seus objetos se comportarão. Os métodos precisam estar sempre localizados dentro de uma classe. Ou seja, método é uma função definida no corpo da classe. Os métodos são chamados de operações no contexto da POO.
- **Variável de classe** – Uma variável que descreve um atributo de uma classe, em vez das instâncias específicas da classe.
- **Variável de instância** – Uma variável que descreve um atributo de uma instância de uma classe, em vez da própria classe.
- **Interface** – Uma especificação de comportamento abstrato, que as classes individuais podem então implementar.
- **Pacote** – Uma coleção de classes e interfaces.
- **Subclasse** – Uma classe mais abaixo na hierarquia de classes do que outra classe, sua superclasse. Uma classe pode ter tantas subclasses quantas forem necessárias.
- **Superclasse** – Uma classe mais acima na hierarquia de classes do que outra classe, sua subclasse. Uma subclasse só pode ter uma superclasse imediatamente acima dela.

Instâncias



Quando criamos um objeto (conta1) de uma determinada classe (Conta) dizemos que estamos instanciando um objeto daquela classe. Indicamos a classe que será usada, damos um nome ao objeto, usamos o sinal de atribuição (=), o operador *new* e indicamos o método construtor que pretendemos usar (Conta()).

Instância da classe e objeto da classe são termos sinônimos no contexto da POO.

A classe é um *template* (modelo) para a construção de objetos/instâncias da classe. Um objeto/instância é uma "cópia da classe".

Tipos de Referência

Os tipos de referência facilitam a manipulação dos objetos em memória, agilizando a construção do código, a gerência de memória e a navegação entre os objetos do sistema.

Uma variável de referência permite o acesso a instâncias de objetos em memória, pois a ela é atribuído o endereço de memória de um objeto. Essa associação é que recebe o nome de referência. Veja o exemplo:

```
public class TesteConta {
	public static void main(String [] args) {
		Conta conta1 = new Conta();
		conta1.numero = 10;
		conta1.saldo = 500;
		System.out.println(conta1);
		System.out.println("Numero: "+conta1.numero);
		System.out.println("Saldo: "+conta1.saldo);
		Conta conta2 = new Conta();
		conta2.numero = 11;
		conta2.saldo = 5330;
		System.out.println(conta2);
		System.out.println("Saldo: "+conta2.numero);
		System.out.println("Saldo: "+conta2.numero);
	}
 }
```

Nesse exemplo é possível perceber que os objetos conta1 e conta2 são do tipo referência, por duas razões:

1- Não são de tipo primitivo;
2- Estão associados ao operador new.

Método Construtor

- É um "método especial" dentro de uma classe que é responsável por criar e inicializar objetos da classe. Ele é executado automaticamente quando um objeto é criado a partir da classe;
- Ele pode ser reconhecido por utilizar o mesmo nome da classe e por não ter nenhum tipo de retorno;
- Pode-se definir quantos métodos construtores forem necessários. A isso dá-se o nome de **sobrecarga de métodos**;
- Cada construtor permite que sejam atribuídos valores específicos para os argumentos da classe, utilizando tipos primitivos e de referência;
- O compilador Java diferencia os construtores baseado na quantidade e nos tipos dos argumentos passados (**assinatura do método**).

Assim, o construtor é um bloco de código que só será chamado durante a construção (instanciação) do objeto. Na linguagem Java o construtor é definido pelo bloco de nome igual ao da sua classe.

Todas as classes possuem um construtor padrão, no exemplo a seguir a instrução new Pessoa() invocará o construtor padrão da classe Pessoa:

```
class Pessoa { }

Pessoa p = new Pessoa();
```

Porém, o construtor padrão deixa de existir se definirmos um construtor no corpo da classe, assim como podemos ver no exemplo a seguir:

```
class Pessoa {
	Pessoa() {
		System.out.println("Construiu");
	}
}
```

Geralmente, usamos o construtor para inicializar os atributos. No exemplo a seguir o construtor recebe uma *String* e um *int* como parâmetros e esses valores são usados para inicializar, respectivamente, as propriedades nome e idade.

O termo *this* refere-se à instância atual da classe. É uma palavra-reservada que permite acessar as propriedades e métodos da instância da classe dentro dos métodos e construtor da classe. No exemplo, a seguir a instrução *this*.nome acessará a propriedade nome do objeto que está sendo criado:

```
class Pessoa {
    String nome;
    int idade;

    Pessoa(String a, int b) {
       this.nome = a;
       this.idade = b;
    }

    imprimir(){
       System.out.println(this.nome+" possui "+this.idade+" anos");
    }
}

Pessoa p1 = new Pessoa("Ana",18);
Pessoa p2 = new Pessoa("Pedro",20);
p1.imprimir();
p2.imprimir();
```

Constitui erro chamar o construtor:

- Passando os parâmetros fora de ordem:

```
Pessoa p3 = new Pessoa(21,"Maria");
```

- Passando a quantidade incorreta de parâmetros:

```
Pessoa p4 = new Pessoa();
Pessoa p5 = new Pessoa("João");
Pessoa p6 = new Pessoa("Lucas",25,"Souza");
```

No Java, as propriedades não precisam ser inicializadas ao construir o objeto. No exemplo a seguir, a classe Cliente não possui construtor explícito, então as propriedades foram inicializadas na definição da classe e, desta forma, os objetos que estão nas variáveis um e dois serão inicializados com "Ana" e 18. Já na classe Individuo as propriedades são inicializadas no construtor, desta forma, os objetos que estão nas variáveis tres e quatro serão inicializados com os valores recebidos como parâmetro pelo construtor.

```
class Cliente {
    String nome = "Ana";
    int idade = 18;
}

Cliente um = new Cliente();
Cliente dois = new Cliente();

class Individuo {
    String nome;
    int idade;

    Individuo(String a, int b) {
       this.nome = a;
       this.idade = b;
    }
}

Individuo tres = new Individuo("Ana",18);
Individuo quatro = new Individuo("Pedro",20);
```

Alguns erros cometidos em construtores:

- Errado: o construtor **não** pode ter anotação de tipo de retorno. Neste exemplo a anotação em amarelo está errada.

```
class Individuo {

    String nome;
    int idade;

    void Individuo (String a, int b) {
       this.nome = a;
        this.idade = b;
    }
}
```

- Errado: o construtor não pode ter a instrução *return* com um valor diferente de this. Um construtor retorna por padrão o objeto criado e neste exemplo está retornando uma String.

```
class Individuo {

    String nome;
    int idade;

    Individuo (String a, int b) {
       this.nome = a;
       this.idade = b;
       return "Oi";
    }
}
```

IV. Utilizando e Criando um Projeto Java no NetBeans

Nesta capacitação usaremos o Apache NetBeans IDE 21 como editor. Ele facilita a criação e execução de projetos e suas classes em Java.

Instalar a versão mais recente da JVM para poder executar o NetBeans, caso ainda não tenha instalada, em: https://www.java.com/pt-BR/

Você poderá baixar o instalador do NetBeans nesse link: https://netbeans.apache.org/front/main/download/.

Veja na aba Videoaulas como instalar e utilizar o NetBeans para criar um projeto Java e suas classes.

V. Referências

Partes desta apostila, foram baseadas no material didático do prof. Dr. Arley Ferreira de Souza coordenador do curso de DSM da Fatec de Jacareí.

Java: How to program, 4th. Ed., Harvey M. Deitel e Paul J. Deitel, Prentice-Hall, 2001, ISBN 0-13-034151-7.

Design patterns: Elements of reusable object-oriented software, Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, Addison-Wesley, 1995, ISBN 0-201-63361-2.

Object-oriented design in Java, Stephen Gilbert, Bill McCarty, The Waite Group Press, 1998, ISBN 1-57169-134-0.

The Unified Modeling Language User Guide, Grady Booch, James Rumbaugh, Ivar Jacobson, Addison-Wesley, 1999, ISBN 0-201-57168-4.

The Java Tutorial: A practical guide for programmers, Mary Campione e Kathy Walrath, acesso em 30/04/2024.

Apache NetBeans Community, Fundação Apache, em https://netbeans.apache.org/front/main/community/, acesso em 30/04/2024.

Java (linguagem de programação), Wikipédia, acesso em 30/04/2024.