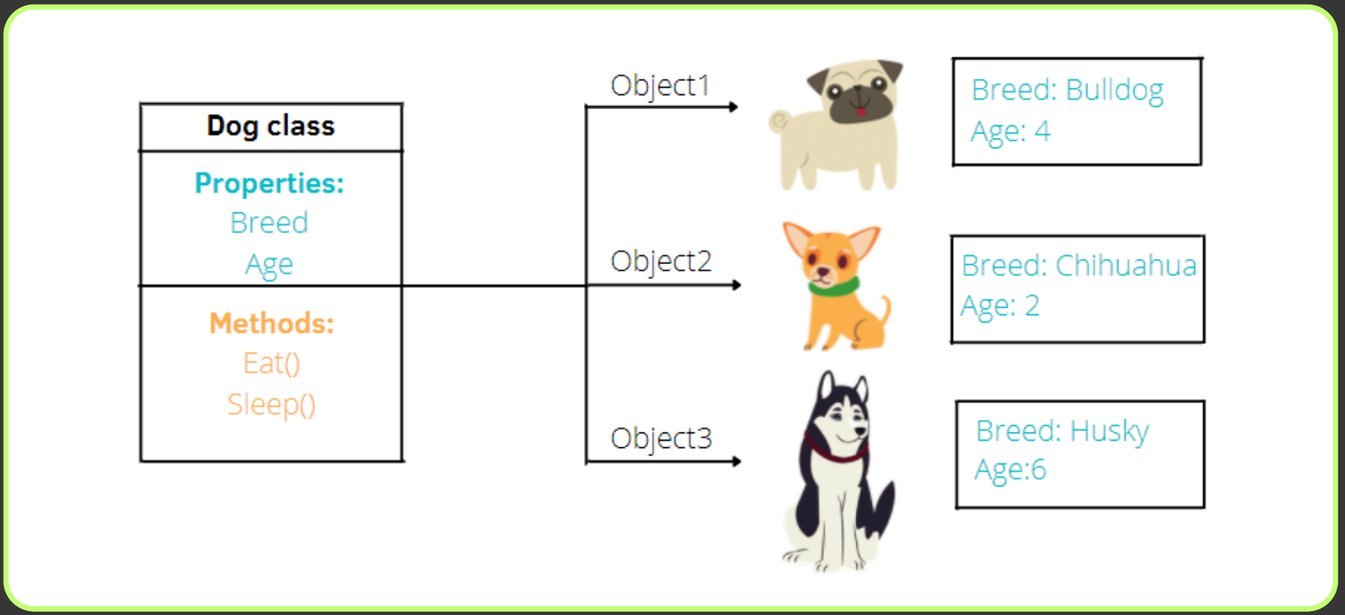

USE LINUX
ANDERSON SIMPLICIO
DEV

## Introdução à Criação de Classes em Java

No Java, classes são a espinha dorsal da **programação orientada a objetos (POO)**.

Uma classe pode ser entendida como um modelo ou um molde que define as propriedades (atributos) e comportamentos (métodos) de um objeto.

Em outras palavras, uma **classe** é como uma **planta arquitetônica**, enquanto um **objeto** é a construção real baseada nessa planta.



## Estrutura Básica de uma Classe

Uma **classe** em Java é definida utilizando a palavra-chave **class**, seguida pelo **nome da classe**, que geralmente começa com uma letra **maiúscula**. Aqui está um exemplo básico de uma classe em Java:

```java
public class Carro {
    // Atributos (propriedades)
    String cor;
    String modelo;
    int ano;

    // Método (comportamento)
    public void acelerar() {
        System.out.println("O carro está acelerando.");
    }
}
```

# Estrutura Básica de uma Classe

```java
public class Carro {
    // Atributos (propriedades)
    String cor;
    String modelo;
    int ano;

    // Método (comportamento)
    public void acelerar() {
        System.out.println("O carro está acelerando.");
    }
}
```

No exemplo acima:

- **Carro**: é o nome da classe. Ele descreve um carro genérico.
- **Atributos**: cor, modelo e ano são os dados ou propriedades que a classe Carro pode ter.
- **Método**: **acelerar()** é uma função que define um comportamento da classe, ou seja, o que o carro pode fazer.

## Criando Objetos a Partir de Classes

Depois de definir uma **classe**, podemos criar objetos com base nela. Para isso, utilizamos o operador new. Veja um exemplo de como instanciar (criar) um objeto da classe Carro:

```java
public class Main {
    public static void main(String[] args) {
        // Criando um objeto da classe Carro
        Carro meuCarro = new Carro();

        // Atribuindo valores aos atributos do objeto
        meuCarro.cor = "Vermelho";
        meuCarro.modelo = "Sedan";
        meuCarro.ano = 2021;

        // Chamando o método do objeto
        meuCarro.acelerar();
    }
}
```

# Criando Objetos a Partir de Classes

```java
public class Main {
    public static void main(String[] args) {
        // Criando um objeto da classe Carro
        Carro meuCarro = new Carro();

        // Atribuindo valores aos atributos do objeto
        meuCarro.cor = "Vermelho";
        meuCarro.modelo = "Sedan";
        meuCarro.ano = 2021;

        // Chamando o método do objeto
        meuCarro.acelerar();
    }
}
```

Explicando o Código:

1. **Instanciação:** Carro meuCarro = new Carro(); cria um novo objeto da classe Carro chamado meuCarro.
2. **Acessando Atributos:** A linha meuCarro.cor = "Vermelho"; define o valor da cor do carro como "Vermelho".
3. **Chamando Métodos:** O método acelerar() é chamado através do objeto meuCarro, resultando na saída "O carro está acelerando.".

## Como o Encapsulamento Funciona em Java

No **Java**, o encapsulamento é alcançado utilizando modificadores de acesso, como **private, protected, public, e default**. Esses modificadores controlam o nível de visibilidade e o acesso aos atributos e métodos de uma classe.

- **private:** Somente a própria classe pode acessar os **atributos (variáveis da classe)**ou métodos declarados como private. É o nível de visibilidade mais restrito e muito usado para atributos que não devem ser acessados diretamente.
- **public:** Atributos e métodos públicos podem ser acessados por qualquer classe. Normalmente, métodos que representam a "interface" pública da classe são declarados como public.
- **protected:** Acessível dentro do mesmo pacote e por subclasses.
- **default (sem modificador explícito):** Acessível apenas dentro do mesmo pacote.

# Como o Encapsulamento Funciona em Java

```java
package aula;

public class ContaBancaria {
    private double saldo;
    private String titular;

    public ContaBancaria(String titular, double saldoInicial) {
        this.titular = titular;
        this.saldo = saldoInicial;
    }

    // Método para acessar o saldo da conta
    public double getSaldo() {
        return saldo;
    }

    // Método para atualizar o titular da conta
    public void setTitular(String titular) {
        this.titular = titular;
    }
```

# Herança

**Herança** é um conceito fundamental na programação orientada a objetos, e em Java, permite que uma classe derive de outra, herdando seus atributos e métodos. Isso ajuda a evitar código repetido e promove a reutilização, além de facilitar a manutenção e expansão do sistema.

## Como a Herança Funciona em Java

Em Java, a herança é implementada com a palavra-chave extends. A classe que herda é chamada de subclasse (ou classe filha), enquanto a classe da qual herda é chamada de superclasse (ou classe pai).

# Herança

```java
public class Funcionario {
    protected String nome;
    protected double salario;

    // Construtor
    public Funcionario(String nome, double salario) {
        this.nome = nome;
        this.salario = salario;
    }

    // Método para exibir informações do funcionário
    public void exibirInformacoes() {
        System.out.println("Nome: " + nome);
        System.out.println("Salário: " + salario);
    }
}
```

# Herança

```java
public class Desenvolvedor extends Funcionario {
    private String linguagemPrincipal;

    // Construtor da subclasse Desenvolvedor
    public Desenvolvedor(String nome, double salario, String linguagemPrincipal) {
        super(nome, salario); // Chama o construtor da superclasse
        this.linguagemPrincipal = linguagemPrincipal;
    }

    // Método para exibir informações específicas do desenvolvedor
    @Override
    public void exibirInformacoes() {
        super.exibirInformacoes();
        System.out.println("Linguagem Principal: " + linguagemPrincipal);
    }
}
```

# Herança

```java
public class Gerente extends Funcionario {
    private double bonus;

    // Construtor da subclasse Gerente
    public Gerente(String nome, double salario, double bonus) {
        super(nome, salario); // Chama o construtor da superclasse
        this.bonus = bonus;
    }

    // Método para exibir informações específicas do gerente
    @Override
    public void exibirInformacoes() {
        super.exibirInformacoes();
        System.out.println("Bônus: " + bonus);
    }
}
```

# Herança

**javac -d out** src/heranca/Funcionario.java
src/heranca/Desenvolvedor.java
src/heranca/Gerente.java
src/heranca/Main.java

- **javac:** É o compilador Java, responsável por compilar os arquivos **.java** e gerar arquivos **.class** (bytecode Java).
- **-d out**: Especifica o diretório de destino para os arquivos compilados.
- Aqui, **out** é o diretório onde serão salvos os arquivos .class após a compilação. Esse diretório será criado caso ainda não exista.

```
project/
├── out/
│   └── heranca/
│       ├── Funcionario.class
│       ├── Desenvolvedor.class
│       ├── Gerente.class
│       └── Main.class
└── src/
    └── heranca/
        ├── Funcionario.java
        ├── Desenvolvedor.java
        ├── Gerente.java
        └── Main.java
```

# Herança

## Executando o Programa Compilado

Depois de compilar, você pode executar a classe **Main** especificando o **classpath -cp** como **out**, onde os **arquivos .class** foram gerados. No terminal, execute:

```
java -cp out heranca.Main
```