

Técnico em Desenvolvimento de Sistemas



Docente: Rafael Sacramento

Sobre o Professor



- Formado em Bacharel em Sistemas de Informação, pós-Graduado em Análise de Sistemas com Ênfase em Governança e Docência da Tecnologia.
- Ex militar das Forças Armadas.
- Tenho 39 anos
- Atuo por mais de 14 anos como professor.



Histórico do Senac

O Senac – Foi criado em 10 de janeiro de 1946 através do decreto-lei 8.621. É uma entidade privada com fins públicos que recebe contribuição compulsória das empresas do comércio e de atividades assemelhadas. A nível nacional é administrado pela Confederação Nacional do Comércio.



Curso:

Carga Horária: 1.200 horas

Dividido em 12 Unidades Curriculares - UC;

Analisar requisitos e funcionalidades da aplicação – 108h;

Auxiliar na Gestão de Projetos de Tecnologia da Informação - 60h;

Desenvolver algoritmos - 108h;

Analisar programação estruturada e orientada a objetos- 48h;

Desenvolver aplicações desktop - 140h;

Criar e manter Banco de Dados - 108h;

Desenvolver aplicações web - 140h;

Desenvolver aplicações mobile - 140h;

Realizar operações de atualização e manutenção em aplicações desenvolvidas - 96h;

Realizar testes nas aplicações desenvolvidas - 108h;

Realizar operações de suporte junto ao usuário - 84h;

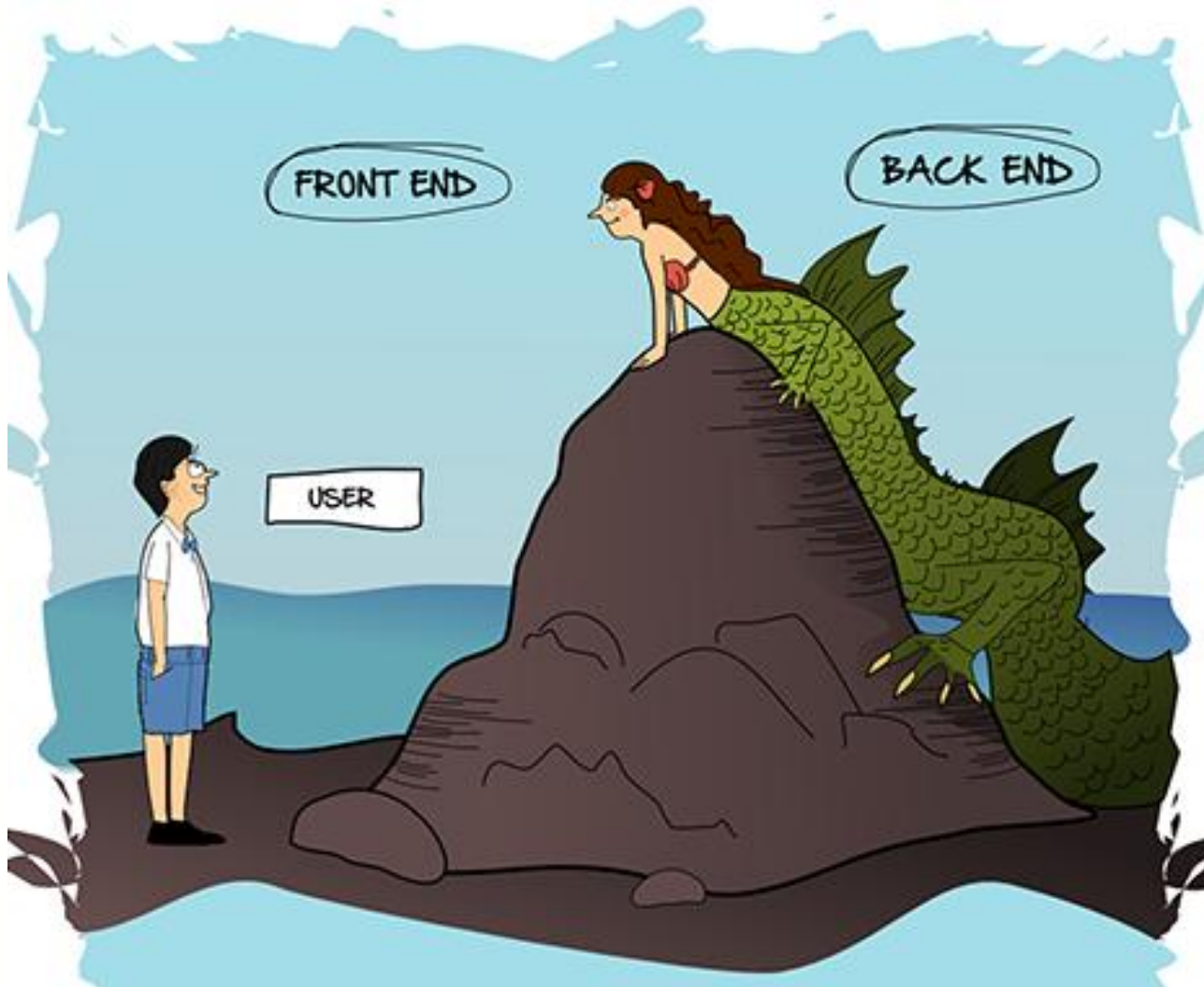
Projeto Integrador Desenvolvedor de aplicações - 60h;

Orientação Objetos (POO).



O que iremos abordar nessa UC4

- **Introdução Orientação Objetos;**
- **Classes e Objetos;**
- **Herança;**
- **Composição e Agregação;**
- **Exceções e Tratamento de Erros;**
- **Coleções e Estruturas de Dados;**
- **Entrada e Saída;**
- **Projeto Orientado a Objetos;**



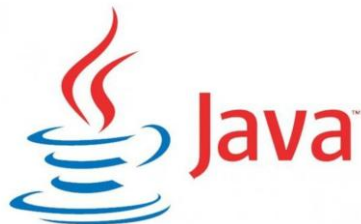
Os **desenvolvedores Back-End** geralmente ganham salários maiores do que os **desenvolvedores Front-end**, pelo fato das linguagens Back-End tenderem a ser mais técnicas. O salário médio de **um desenvolvedor Back-End é de R\$ 5.833** e do **desenvolvedor Front-End é de R\$ 5.281**.



Quais são as 10 principais linguagens de programação?

1. Python
2. C#
3. C++
4. JavaScript
5. PHP
6. Swift
7. Java
8. Go
9. SQL
10. Ruby

Linguagens de programação Back-End populares



Informações importantes:

- Uma das linguagens de programação mais popular do mundo;
- É uma linguagem mais difícil para iniciantes do que as outras linguagens Back-End; e
- Desenvolvedores ganham em média R\$ 6.995,00 por mês.



JS

Informações importantes:

- Pode ser utilizada tanto para front e Back-End.
- Muito popular com uma grande comunidade.
- Pode ser difícil de ser mantida e escalada.

Diferenças



Java é uma linguagem de programação OOP, ao passo que **Java Script** é uma linguagem de scripts OOP.

Java cria aplicações executadas em uma máquina virtual ou em um browser, ao passo que o código.

Introdução Orientação Objetos



É um conceito fundamental no mundo da programação. **Embora possa parecer intimidante para iniciantes.**

Vamos descomplicar a POO, **usando exemplos do mundo real para ilustrar cada parte dela.**




O que é Programação Orientada a Objeto?

Para começar, precisamos entender o que é a POO em Java. É um **paradigma** de **programação que se baseia na ideia de "objetos"**. Pense em objetos como coisas do mundo real. Cada objeto tem características (atributos) e pode realizar ações (métodos).



O que faz um desenvolvedor Back-End?

Então, como desenvolvedor Back-End, você talvez passe o seu dia fazendo coisas como:

- Mantendo aplicações legacy (programas antigos que foram construídos por outros programadores)
 - Apagando o fogo (por exemplo, bugs, aplicativos quebrados)
 - Indo em reuniões (para conseguir exigências para um projeto)
 - Colaborar com desenvolvedores Front-End em um projeto, etc.
 - De fato escrever códigos para novas aplicações/projetos.
- 



Então, enquanto você certamente estará programando com linguagens Back-End, não pense que isso será a única coisa que irá fazer como um desenvolvedor back end.

Tipos de linguagens back-End

A programação em Back-End pode tanto orientada a objetos/object-oriented (OOP) ou funcional.

- **OOP** é a técnica que **foca na criação de objetos**. Com a programação orientada a objetos, os códigos precisam ser executados de uma forma em particular. **Linguagens OOP populares são Java, .NET, Python e PHP.**
- **Programação Back-End funcional** é uma técnica mais focada na “ação”. A programação funcional utiliza a linguagem declarativa, isso significa que os statements podem ser executados de qualquer ordem. É comumente utilizado no data science e em linguagens populares como **F# e R.**
- **As linguagens Back-End podem ser escritas tanto de forma estática ou dinâmica.** O primeiro é mais **rígido**, mas melhor para encontrar erros, onde que o último é mais **flexível**, mas permite com que as variáveis mudem os tipos (no qual pode contar com erros inesperados).

Programação Funcional vs Orientada a Objetos

Programação Funcional:

- Funções como Base: Centraliza tudo em funções.
- Imutabilidade: Dados não mudam após criados.
- Sem Efeitos Colaterais: Funções não alteram nada fora delas.
- Recursão: Usa funções que se chamam repetidamente.

Programação Orientada a Objetos (OOP):

- Objetos e Classes: Código organizado em objetos.
- Encapsulamento: Agrupa dados e métodos, escondendo detalhes internos.
- Herança: Classes podem herdar características de outras classes.
- Polimorfismo: Objetos tratados como de classes diferentes.

Programação Funcional vs Orientada a Objetos

Resumo

- **Funcional:** Foco em funções e dados imutáveis.
- **OOP:** Foco em objetos que podem mudar de estado.

Visualização

- **Funcional:** Função -> Dados
- **OOP: Objeto** -> Ações e Estados.

Exemplos:

Programação Funcional

- **Funções como Base:** Tudo é baseado em funções.
- **Imutabilidade:** Dados não mudam após criados.
- **Sem Efeitos Colaterais:** Funções não alteram nada fora delas.
- **Recursão:** Usa funções que se chamam repetidamente.

Programação Orientada a Objetos (OOP)

- **Objetos e Classes:** Código organizado em objetos.
- **Encapsulamento:** Agrupa dados e métodos.
- **Herança:** Classes podem herdar características de outras.
- **Polimorfismo:** Objetos tratados como de classes diferentes.

Visual Studio Code - VSCODE

É um editor de código-fonte desenvolvido pela Microsoft para **Windows, Linux e macOS**. Ele inclui suporte para depuração, controle de versionamento Git incorporado, realce de sintaxe, complementação inteligente de código, snippets e refatoração de código.



Execute o VSCODE;



Extensões VSCODE

Extension Pack for Java



VSCODE Icon



Enhance Me



GPT CoPilot



Primeiro código em Java

```
public class App {  
    public static void main(String[] args) throws  
    Exception {  
        System.out.println("Hello, World!");  
    }  
}
```

Vamos tentar entender linha por linha:

1 - `public class App {`

public: Este é um **modificador de acesso** que significa que a classe **pode ser acessada de qualquer lugar**.

class: Esta palavra-chave é usada para **declarar uma classe em Java**.

App: Este é o **nome da classe**. Em Java, os **nomes de classe geralmente começam com uma letra maiúscula**.

2 - `public static void main(String[] args) throws Exception {`

public: Este é um **modificador de acesso** que significa que a classe **pode ser acessada de qualquer lugar**.

static: Este **modificador** indica que o **método** **pertence à classe** e não a uma **instância da classe**. Isso **significa** que você **pode chamar este método** **sem criar um objeto da classe**.

void: Significa que o **método não retorna nenhum valor**.

main: Este é o **nome do método**. Em Java, **main é o ponto de entrada para qualquer aplicação que deseja ser executada**. É um método especial que a JVM (Java Virtual Machine) **procura para iniciar a execução do programa**.

String[] args: Este é o **parâmetro do método main**. Ele aceita um **array de strings**, que são os **argumentos da linha de comando** **passados para o programa quando ele é executado**.

throws Exception: Esta parte da declaração do método indica **que o método pode lançar uma exceção do tipo Exception**. Isso é usado para indicar que o método pode ter um comportamento que resulta em uma exceção que **precisa ser tratada**.

3 - System.out.println("Hello, World!");

System: Esta é uma classe predefinida em Java que contém vários métodos e variáveis de utilidade, **incluindo out**.

out: Este é um membro estático da classe System que representa o console padrão de saída (**geralmente a tela do computador**).

println: Este é um método de out que imprime uma linha de texto no console. **A linha de texto é seguida por uma nova linha**.

"Hello, World!": Esta é a **string** que será impressa no console.

Programação Orientada a Objeto

Você lembra que eu falei que é um **paradigma** de **programação que se baseia na ideia de "objetos"**. Pare e pense em objetos como coisas do mundo real.



Cada objeto tem **características (atributos)** e pode realizar **ações (métodos)**.

Vamos entender o que é **atributos e métodos**:



Atributos:

Imagine um carro. Ele tem **atributos como cor, modelo e marca**. Em POO em Java, chamamos esses **atributos de "variáveis de instância"**.

Exemplo de Código:

```
public class Carro {  
    String cor;  
    String modelo;  
    String marca;  
}
```

- **String**: Este é o tipo dos atributos, que é uma sequência de caracteres (texto).
- **cor, modelo, marca**: Estes são os nomes dos atributos da classe Carro. Eles armazenam a cor, o modelo e a marca do carro, respectivamente.

Métodos:

Agora, pense nas ações que um carro pode executar: **acelerar, frear e virar**. Essas ações correspondem aos "métodos" na POO em Java.

Exemplo:

- O carro está acelerando.
- O carro está freando.
- O carro está virando para **direção**.

Exemplo de Código:

```
public class Carro {  
    String cor;  
    String modelo;  
    String marca;  
  
    void acelerar() {  
        System.out.println("O carro está acelerando.");  
    }  
    void frear() {  
        System.out.println("O carro está freando.");  
    }  
    void virar(String direcao) {  
        System.out.println("O carro está virando para " + direcao + ".");  
    }  
}
```

Entenda linha por linha:

1 - **void** **acelerar()** {

void: Indica que o método não retorna nenhum valor.

acelerar(): Este é o nome do método. Em Java, os métodos são normalmente nomeados com verbos para indicar que realizam uma ação.

2 - **void** **virar(String direcao)** {

virar(String direcao): Método que recebe um parâmetro direção do tipo String e imprime a mensagem "O carro está virando para " seguido da direção fornecida.

Classes e Objetos: Os Blocos de Construção

Para tornar isso mais claro, pense em uma "classe" como um modelo para criar objetos. **Uma classe pode ser vista como um projeto que descreve como os objetos devem ser construídos.**



Exemplo do Mundo Real: Carro

Vamos usar o exemplo do carro para ilustrar esses conceitos:

- **Classe:** A classe é como o plano de construção para um carro. Ela define quais atributos (**cor, modelo, marca**) e **métodos** (**acelerar, frear, virar**) um carro terá.
- **Objeto:** Um objeto é uma instância da classe. Se você tem um **carro vermelho da marca "Toyota"**, você tem um objeto que segue as **especificações da classe de carros**.

Exemplo de Código:

// Criando um objeto a partir da classe Carro

```
public static void main(String[] args) {  
    carro meuCarro = new carro();  
    meuCarro.cor = "Vermelho";  
    meuCarro.modelo = "Sedan";  
    meuCarro.marca = "Toyota";
```

```
System.out.println("Meu carro é um " + meuCarro.marca + " " +  
    meuCarro.modelo + " de cor " + meuCarro.cor + ".");
```


// Usando métodos do objeto

meuCarro.acelerar();

meuCarro.frear();

meuCarro.virar("esquerda");

}

}

Construtor de Classe:

Construtor de classe em Java é um **método especial** utilizado para **inicializar objetos**. Ele é **chamado automaticamente** quando um **novo objeto de uma classe é instanciado**. Os construtores têm o mesmo nome que a classe e não têm um tipo de retorno, nem mesmo “**void**”.

Características dos Construtores:

- 1 - Nome Igual ao da Classe:** O construtor deve ter o mesmo nome que a classe.
- 2 - Nome Igual ao da Classe:** O construtor deve ter o mesmo nome que a classe.
- 3 - Sem Tipo de Retorno:** Construtores não têm tipo de retorno, nem mesmo **void**.
- 4 - Inicialização de Objetos:** Eles são usados para inicializar os atributos de um objeto ao criar uma instância da classe.

Qual a função do construtor de classe?

É responsável pela criação do objeto daquela classe, iniciando com valores seus atributos ou realizando outras funções que possam vir a ser necessárias

Entenda linha por linha:

A criação da classe “**main**” em um programa Java serve como o ponto de entrada para a aplicação. É dentro desse método que a execução do programa começa.

Carro meuCarro = new Carro(); Esta linha cria uma nova instância da classe Carro e a armazena na variável meuCarro.

meuCarro.cor = "Vermelho"; Atribui a cor "Vermelho" ao atributo cor do objeto meuCarro.

meuCarro.modelo = "Sedan"; Atribui o modelo "Sedan" ao atributo modelo do objeto meuCarro.

meuCarro.marca = "Toyota"; Atribui a marca "Toyota" ao atributo marca do objeto meuCarro.

System.out.println("Meu carro é um " + meuCarro.marca + " " + meuCarro.modelo + " de cor " + meuCarro.cor + "."); Imprime uma mensagem que descreve o carro no console.

meuCarro.acelerar(); Chama o método acelerar do objeto meuCarro.

meuCarro.frear(); Chama o método frear do objeto meuCarro.

meuCarro.virar("esquerda"); Chama o método virar do objeto meuCarro, passando a string "esquerda" como argumento.

Exercícios 04 OOP - Crie uma **classe Nadador** com os atributos **nome** e **idade**. Em seguida, crie **um objeto** dessa classe e defina os valores dos atributos.

Que tal fazemos o código assim:

1. Vamos começar com a definição da Classe Nadador

```
public class Nadador {
```

Definimos uma classe pública chamada Nadador. A palavra-chave public significa que a classe pode ser acessada de qualquer outro pacote.

2. Atributos da Classe

```
String nome;
```

```
int idade;
```

Declaramos um **atributo nome do tipo String e idade do tipo int**. Este atributo é de acesso **padrão** (default), o que significa que é acessível apenas dentro do mesmo pacote.

3. Construtor da Classe

```
public Nadador(String nome, int idade) {  
    this.nome = nome;  
    this.idade = idade;  
}
```

public Nadador(String nome, int idade): Define um construtor público que aceita dois parâmetros: nome e idade.

this.nome = nome: Atribui o valor do parâmetro nome ao atributo da instância nome.

this.idade = idade: Atribui o valor do parâmetro idade ao atributo da instância idade.

4. Métodos Getters

```
public String getNome() {  
    return nome;  
}  
  
public int getIdade() {  
    return idade;  
}
```

public String getNome(): Método getter público que retorna o valor do atributo **nome**.

public int getIdade(): Método getter público que retorna o valor do atributo **idade**

4. Método main

```
public static void main(String[] args) {  
    Nadador nadador = new Nadador("Rafael", 22);
```

public static void main(String[] args): Define o **método main**, ponto de entrada da aplicação.

public permite que o método seja chamado pela JVM

static significa que o método pertence à classe e não a uma instância da classe
void indica que o método não retorna nenhum valor

String[] args é um array de strings que pode receber argumentos de linha de comando.

Nadador nadador = new Nadador("Rafael", 22): Criamos uma nova instância da classe **Nadador**, passando "Rafael" e 22 como argumentos para o construtor, e atribui esta instância à variável **nadador**.

5. Exibindo os valores dos atributos do objeto

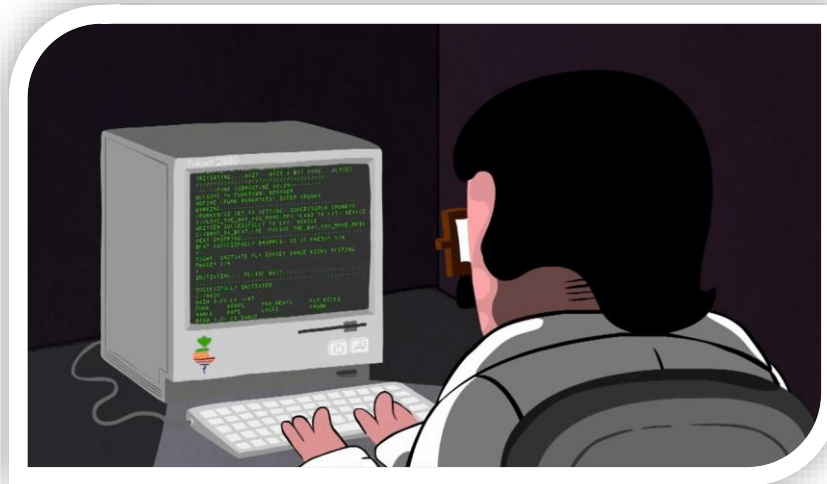
```
System.out.println("Nome: " + nadador.getNome());  
  
System.out.println("Idade: " + nadador.getIdade());  
  
}  
  
}
```

System.out.println("Nome: " + nadador.getNome()): Imprime no console a string "Nome: " concatenada com o valor retornado pelo método getNome().

System.out.println("Idade: " + nadador.getIdade()): Imprime no console a string "Idade: " concatenada com o valor retornado pelo método getIdade()

métodos getter

Usado para **obter** o valor de um atributo privado. Eles **permitem que outras classes acessem os valores dos atributos de forma controlada**. Um método getter geralmente começa com a palavra "**get**" seguida pelo nome do atributo com a primeira letra em maiúscula. **Exemplo do código do exercício.**



Exercícios 05 OOP - Jav

métodos Setter

Usado para **definir ou modificar** o valor de um atributo privado. Eles permitem que outras classes **alterem** os valores dos atributos de forma controlada. Um método setter geralmente começa com a palavra "**set**" seguida pelo nome do **atributo com a primeira letra em maiúscula**.



O que são Getters e Setters?

Imagine que você tem um cofre (que é a sua classe) e dentro dele tem um segredo (que são os atributos privados da classe). Para proteger o segredo, você não deixa as pessoas mexerem diretamente. Em vez disso, você dá a elas um **controle remoto (os métodos getter e setter)** que permite ver ou mudar o segredo de maneira segura.



Métodos Getter

O método getter é como um botão no controle remoto que você usa para **ver** o segredo. Quando você aperta esse botão, o segredo é mostrado para você.

Métodos Setter

O método setter é como um botão no controle remoto que você usa para **mudar** o segredo. Quando você aperta esse botão e passa um novo valor, o segredo é atualizado com esse valor.

Resumo:

Getter: Usado para ver o valor de um atributo.

Setter: Usado para mudar o valor de um atributo.

Exemplo com setter:

```
public class Pessoa {  
    private String nome;  
  
    public void setNome(String nome) {  
        this.nome = nome;    }  
  
    public String getNome() {  
        return nome;    }  
  
    public static void main(String[] args) {  
        Pessoa pessoa = new Pessoa();  
  
        pessoa.setNome("Maria");  
  
        System.out.println("Nome da pessoa: " +  
        pessoa.getNome());  
    }  
}
```


Veja os detalhes no Código

Classe Pessoa:

Atributo nome: Um atributo privado que armazena o nome da pessoa.

Método setNome: Define o valor do atributo nome.

Método getNome: Retorna o valor atual do atributo nome.

Método main:

Criação da Instância: Cria uma instância da classe Pessoa.

Uso do Setter: Define o nome da pessoa como "**Maria**" usando o **método setNome**.

Uso do Getter: Obtém o nome usando o método getNome e exibe-o no console.

Demonstrando que o método setNome foi usado para definir o valor do nome da pessoa e que o método getNome foi usado para obter e exibir esse valor.

Os pilares da P00

EHP:

- **Encapsulamento;**
- **Herança; e**
- **Polimorfismo;**

Encapsulamento

É um conceito importante na POO em Java. **Ele envolve o agrupamento de dados (atributos) e os métodos que operam esses dados em uma única unidade, uma classe.**

Essa técnica utilizada para esconder uma ideia, ou seja, não expor detalhes internos para o usuário, tornando partes do sistema mais independentes possível.



Exemplo com setter com validação:

```
public class Pessoa {  
    // Atributos privados  
    private String nome;  
    private int idade;  
  
    // Construtor  
    public Pessoa(String nome, int idade) { this.nome = nome;  
    this.idade = idade;  
    }  
  
    // Métodos públicos para acessar e modificar os atributos  
  
    public String getNome() {  
        return nome;  
    }  
  
    public void setNome(String nome) {  
        this.nome = nome;  
    }  
}
```

```
public int getIdade() {  
    return idade;  
}
```

```
public void setIdade(int idade) {
```

```
    // Podemos adicionar lógica para validação
```

```
    // Verifica se a quantidade é não negativa
```

```
    if (idade >= 0) {  
        this.idade = idade; // 'this.idade' é o atributo da classe, 'idade' é o parâmetro do construtor  
    } else { System.out.println("Mensagem"); // Mensagem de erro  
    }  
}
```

```
    // Método para exibir as informações da pessoa
```

```
public void exibirInformacoes() {  
    System.out.println("Nome: " + this.nome); System.out.println("Idade: " + this.idade);  
}
```

```
// Método principal para testar a classe  
public static void main(String[] args) {
```

```
// Criação de um objeto Pessoa  
Pessoa pessoa = new Pessoa("Alice", 30);
```

```
// Exibir as informações da pessoa  
pessoa.exibirInformacoes();
```

```
// Alterar e exibir as novas informações  
pessoa.setNome("Maria"); pessoa.setIdade(28);  
pessoa.exibirInformacoes();  
}  
}
```

Criação de Objeto: `Pessoa pessoa = new Pessoa("Alice", 30);`

Estamos criando um novo objeto Pessoa com o nome "Alice" e idade 30.

Chamada de Métodos: `pessoa.exibirInformacoes();`

Exibe as informações iniciais. Após isso, `pessoa.setNome("Maria");` e `pessoa.setIdade(28);` modificam os atributos do objeto, que são então exibidos novamente.

Olhe na exibição depois do Debug

O primeiro par de linhas exibe as informações iniciais, e o **segundo par exibe as informações após a modificação dos atributos nome e idade**. O **encapsulamento** neste exemplo é importante porque permite controlar como os dados são acessados e modificados, além de permitir a validação de entrada (como a verificação de idade não negativa).

Herança

Um dos pilares fundamentais da programação orientada a objetos (POO) e é uma **forma de estabelecer uma hierarquia entre classes**. Em Java, a herança permite que uma classe (denominada classe filha ou subclasse) **herde atributos e métodos de outra classe** (denominada classe mãe ou superclasse).

É como a genética na programação. **Você pode criar uma nova classe baseada em uma classe existente**, herdando seus atributos e métodos. Isso economiza tempo e promove a reutilização de código.



Exemplo

Class Cachorro

Atributo String Nome

Metodos:

void comer ()

void beber ()

void latir ()

void lamber ()

Class Gato

Atributo String Nome

Metodos:

void comer ()

void beber ()

void miar ()

Exemplo Cachorro sem Herança

```
public class Cachorro {  
    private String nome;  
  
    public void comer(){  
        System.out.println("O cachorro  
comeu");  
    }  
    public void beber(){  
        System.out.println("O cachorro bebeu");  
    }  
    public void latir(){  
        System.out.println("O cachorro latiu");  
    }  
    public void lamber(){  
        System.out.println("O cachorro  
lambeu");  
    }  
}
```

```
    public String getNome() {  
        return nome;  
    }  
    public void setNome(String nome) {  
        this.nome = nome;  
    }  
    public static void main(String[] args) {  
        Cachorro cachorro = new Cachorro();  
        cachorro.setNome("Rex");  
        cachorro.comer();  
        cachorro.beber();  
        cachorro.latir();  
        cachorro.lamber();  
    }  
}
```

Exemplo Gato sem Herança

```
public class Gato {  
    private String nome;  
  
    public void comer(){  
        System.out.println("O gato comeu");  
    }  
    public void beber(){  
        System.out.println("O gato bebeu");  
    }  
    public void miar(){  
        System.out.println("O gato miou");  
    }  
    public String getNome() {  
        return nome;  
    }  
    public void setNome(String nome) {  
        this.nome = nome;  
    }  
}
```

```
public static void main(String[] args) {  
    Gato gato = new Gato();  
    gato.setNome("Felix");  
    gato.comer();  
    gato.beber();  
    gato.miar();  
}
```

Class com Heranças

Exemplo Class Super Pai

//package animais:: Define que esta classe pertence ao pacote animais.

//Pacotes são usados para organizar classes relacionadas em grupos.

package animais;

// Definição da classe Animal

public class Animal {

// Atributo privado para armazenar o nome do animal

private String nome;

// Construtor da classe Animal que inicializa o nome

**public Animal(String nome) {
 this.nome = nome;
}**

// Método para simular o ato de comer

**public void comer() {
 System.out.printf("%s comeu\n", nome);
}**

// Método para simular o ato de beber

**public void beber() {
 System.out.printf("%s bebeu\n", nome);
}**

// Método getter para obter o nome do animal

**public String getNome() {
 return nome;
}**

// Método setter para definir o nome do animal

**public void setNome(String nome) {
 this.nome = nome;
}
}**

println() - imprime e pula uma linha

printf() - um recurso muito poderoso onde você pode definir uma formatação de saída que pode consistir em texto fixo e **especificadores de formato**.

Os **especificadores** de formato são como marcadores de lugares para um valor, especificando o tipo da saída dos dados que iniciam com um sinal de porcentagem (%) seguido por um caractere representando seu tipo de dado.

%d	representa números inteiros
%f	representa números floats
%2f	representa números doubles
%b	representa valores booleanos
%c	representa valores char

Exemplo Class Filho Cachorro

package animais;; Define que esta classe pertence ao pacote animais

package animais;

// Definição da classe Cachorro, que herda de Animal

`public class Cachorro extends Animal {`

// Construtor da classe Cachorro que chama o construtor da classe pai (Animal)

`public Cachorro(String nome) {`

`super(nome);` *// Chama o construtor da classe Animal para inicializar o nome*

`}`

// Método para simular o ato de latir
`public void latir() {`

// Imprime uma mensagem indicando que o cachorro latiu

`System.out.printf("%s latiu\n",
getNome());
}`

// Método para simular o ato de lambear
`public void lambear() {`

// Imprime uma mensagem indicando que o cachorro lambeu

`System.out.printf("%s lambeu\n",
getNome());
}
}`

package animais; indica que a classe Cachorro faz parte do pacote animais.

A palavra-chave **extends** indica que Cachorro é uma subclasse de Animal, o que significa que ela herda todos os atributos e métodos da classe Animal.

O **super()** é um método especial que chama o construtor do método de onde você herdou a classe.

O método **lamber** **imprime uma mensagem indicando que o cachorro lambeu.** Assim como no método latir, **ele utiliza o método getNome da classe Animal.**

Exemplo Class Filho Gato

package animais;; Define que esta classe pertence ao pacote animais

package animais;

// Declaração da classe Gato, que herda da classe Animal

public class Gato extends Animal {

// Construtor da classe Gato que chama o construtor da classe pai (Animal)

public Gato(String nome) {

super(nome); *// Chama o construtor da classe Animal para inicializar o nome*

}

// Método para simular o ato de miar
public void miar() {

// Imprime uma mensagem indicando que o gato miou

System.out.printf("%s miou\n",
getNome());

}

}

Exemplo Importa a classe para exibição

```
import animais.Cachorro; // Importa a classe  
Cachorro do pacote 'animais'
```

```
import animais.Gato; // Importa a classe Gato  
do pacote 'animais'
```

```
// Definição da classe principal App
```

```
public class App {  
    // Método principal que será executado ao iniciar  
    o programa  
    public static void main(String[] args) throws  
    Exception {  
        // Criação de um objeto da classe Cachorro e  
        passando o nome "Cachorro"  
        Cachorro cachorro = new  
        Cachorro("Cachorro");
```

```
// Chamando os métodos da classe Cachorro  
cachorro.comer(); // Simula o cachorro  
comendo
```

```
cachorro.latir(); // Simula o cachorro latindo  
cachorro.beber(); // Simula o cachorro  
bebendo
```

```
cachorro.lamber(); // Simula o cachorro  
lambendo
```

```
// Criação de um objeto da classe Gato e  
passando o nome "Gato"
```

```
Gato gato = new Gato("Gato");
```

```
// Chamando os métodos da classe Gato  
gato.comer(); // Simula o gato comendo  
gato.beber(); // Simula o gato bebendo  
gato.miar(); // Simula o gato miando
```

```
}  
}
```

As importações trazem as classes Cachorro e Gato do pacote animais para **que** possam ser usadas na classe App.

Cachorro cachorro = new Cachorro("Cachorro"); - Criação de um objeto da classe Cachorro e passando o nome "Cachorro"

Chamadas de Métodos do Objeto Cachorro:

`cachorro.comer();` // *Simula o cachorro comendo*

`cachorro.latir();` // *Simula o cachorro latindo*

`cachorro.beber();` // *Simula o cachorro bebendo*

`cachorro.lamber();` // *Simula o cachorro lambendo*

Chamando os métodos comer, latir, beber e lambar do objeto cachorro, simulando as ações correspondentes.

Polimorfismo

Significa "muitas formas", é o termo definido em linguagens orientadas a objeto, como por exemplo Java, C# e C++, que permite ao desenvolvedor usar o mesmo elemento de formas diferentes. Polimorfismo denota uma situação na qual um objeto pode se comportar de maneiras diferentes ao receber uma mensagem.

É como a genética na programação. Você pode criar uma nova classe baseada em uma classe existente, herdando seus atributos e métodos. Isso economiza tempo e promove a reutilização de código.

Vamos aprofundar o conceito de polimorfismo orientado a objetos em Java com exemplos práticos. Polimorfismo permite que objetos de diferentes classes sejam tratados como objetos da mesma classe base, permitindo que você chame métodos sem se preocupar com a classe específica do objeto.

Utilização de Polimorfismo na Classe App

```
import animais.Cachorro; // Importa a classe  
Cachorro do pacote 'animais'
```

```
import animais.Gato; // Importa a classe Gato  
do pacote 'animais'
```

```
// Definição da classe principal App  
public class App {
```

```
    // Método principal que será executado ao iniciar  
    o programa
```

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

```
        // Criar um array de animais
```

```
        Animal[] animais = new Animal[3];
```

```
// Adicionar diferentes tipos de animais ao array
```

```
    animais[0] = new Cachorro("Rex");
```

```
    animais[1] = new Gato("Mimi");
```

```
    animais[2] = new Cachorro("Max");
```

```
    // Iterar sobre o array e chamar o método  
    fazerSom em cada animal
```

```
        for (Animal animal : animais) {  
            animal.fazerSom();
```

```
        }
```

```
    }
```

```
}
```

Agradeço atenção.

- Rafael Sacramento – rferfa@gmail.com
- Linkledin - <https://www.linkedin.com/in/rafael-do-sacramento-bomfim-9150784b/>
- Instagram - <https://www.instagram.com/rafaelrfe/>

“90% DO SUCESSO SE BASEIA EM INSISTIR”