

## Orientação a Objetos Na aula passada...

- Funções em Java: definição, sintaxe e usos práticos;
- Método construtor: definição, tipos e usos práticos;
- Revisão da aula 1 de Orientação a Objetos.



## Orientação a Objetos

### Na aula de hoje

- Encapsulamento: revisão do conceito e aplicação dos modificadores de acesso;
- Herança: classe abstrata, superclasse, classe derivada;
- Getters e Setters;
- Interface.

**Coding In Java Be Like**



```
header1_initialDistance  
if ($(window).scrollTop() > header1_initialDistance) {  
    if (parseInt(header1.css('padding-top'), 10) > header1_initialPadding) {  
        header1.css('padding-top', '' + $(window).scrollTop() - header1_initialDistance + header1_initialPadding);  
    }  
} else {  
    header1.css('padding-top', '' + header1_initialPadding);  
}
```

```
header2_initialDistance  
if ($(window).scrollTop() > header2_initialDistance) {  
    if (parseInt(header2.css('padding-top'), 10) > header2_initialPadding) {  
        header2.css('padding-top', '' + $(window).scrollTop() - header2_initialDistance + header2_initialPadding);  
    }  
} else {  
    header2.css('padding-top', '' + header2_initialPadding);  
}
```

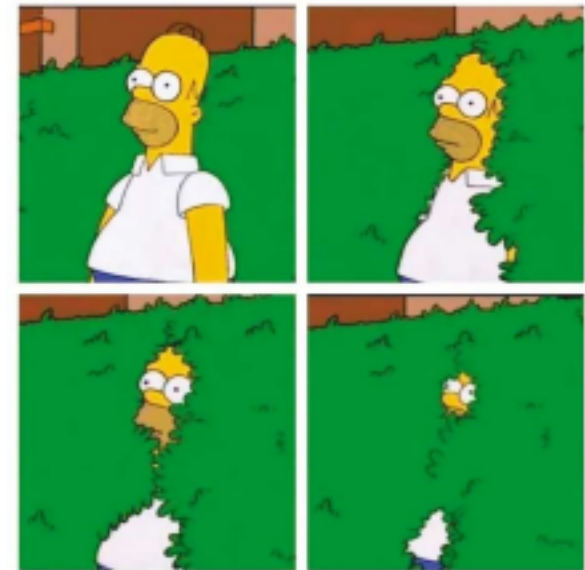
# Orientação a Objetos Encapsulamento

## Conceito

Encapsulamento é um dos princípios fundamentais da programação orientada a objetos (POO). Ele se refere à prática de **esconder os detalhes internos de um objeto e expor apenas o que é necessário através de métodos públicos**.

## Benefícios

- Segurança: Impede alterações não autorizadas nos dados.
- Manutenção: Facilita a modificação do código sem impactar outras partes do programa.
- Reutilização: Permite reutilizar a classe sem se preocupar com a implementação interna.



# Orientação a Objetos Encapsulamento

## Modificadores de acesso

## Tipos de modificadores

Modificadores de acesso controlam a visibilidade dos membros de uma classe (atributos e métodos).

São palavras-chave que definem o nível de visibilidade dos membros de uma classe.

public

acessível por qualquer classe

private

acessível apenas dentro da própria classe

protected

acessível por classes no mesmo pacote ou por subclasses

Sem modificador (package-private)

acessível apenas dentro do mesmo pacote.

## Orientação a Objetos Modificadores de Acesso

Exemplo de código

```
public class Pessoa {  
    private String nome; public int idade;  
    Explicação  
    public void ExibirNome()  
    {  
        System.out.println("Nome: " + nome);  
    }  
}
```

O atributo nome é private, acessível apenas dentro da classe Pessoa.

O atributo idade é public, acessível por qualquer parte do código.

Os métodos ExibirNome e DefinirNomesão public,

permitindo o acesso externo.

```
}  
  
public void DefinirNome(String novoNome) {  
    nome = novoNome;  
}  
}
```

Exercício 1 - Carro

## Orientação a Objetos

### Modificadores de Acesso

- Crie uma classe Carro com os atributos marca (público), modelo (protegido) e preço (privado). Adicione um método para exibir as informações do carro.
- Modifique a classe Carro para permitir que a variável preço possa ser acessada através de um método público.
- Crie uma classe Veiculo no mesmo pacote que Carro e tente acessar os atributos.

```
header1_initialDistance  
if ($(window).scrollTop() > header1_initialDistance) {  
    if (parseInt(header1.css('padding-top'), 10) > header1_initialPadding) {  
        header1.css('padding-top', '' + $(window).scrollTop() - header1_initialDistance + header1_initialPadding);  
    }  
} else {  
    header1.css('padding-top', '' + header1_initialPadding);  
}
```

```
header2_initialDistance  
if ($(window).scrollTop() > header2_initialDistance) {  
    if (parseInt(header2.css('padding-top'), 10) > header2_initialPadding) {  
        header2.css('padding-top', '' + $(window).scrollTop() - header2_initialDistance + header2_initialPadding);  
    }  
} else {  
    header2.css('padding-top', '' + header2_initialPadding);  
}
```



# Orientação a Objetos

## Getters e Setters

### Getters

Métodos que permitem a leitura (acesso) de um atributo privado.

### Síntaxe

```
public tipo_do_atributo getAtributo() { return atributo;
}
```

### Setters

Métodos que permitem a escrita (modificação) de um atributo privado.

### Sintaxe

```
public void setAtributo(tipo_do_atributo atributo) { this.atributo = atributo; }
```

## Orientação a Objetos Getters e Setters

### Exemplo de código

```
public class Pessoa { private String nome; private int
idade;
public void setIdade(int idade) {
    if (idade >= 0) { // Validação
        this.idade = idade;
    }
}
```

```

    } else {
public String getNome() {

    return nome; }

}

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

public int getIdade() {
    return idade;
}

```

```

System.out.println("Idade
                        inválida!");
}

```

Arquivo 'Pessoa.java'

## Orientação a Objetos

### Getters e Setters

Exemplo de código

```

class Main {
    public static void main(String[] args) { Pessoa pessoa = new Pessoa();
        pessoa.setNome("João");
        pessoa.setIdade(30);

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

```

Arquivo 'Main.java'

Exercício 2 - Carro

## Orientação a Objetos

### Modificadores de Acesso

- Refatore o exercício anterior inserindo um getter e um setter para o atributo Preço.
  - O Setter de Preço deve possuir uma validação na qual, caso o usuário digite um valor igual ou menor que 0, uma mensagem de erro deve ser exibida.
- Crie uma classe Aluno com os atributos nome, matricula e curso, todos privados.

Exercício 3 - Aluno

## Orientação a Objetos

### Getters e Setters

Implemente getters e setters para os atributos, garantindo que o nome seja preenchido e a matrícula seja um número válido.

Dica: Para verificar se a matrícula é válida, você pode utilizar uma estrutura condicional que valida se o usuário digitou apenas '0' como valor desse atributo.

```
header1_initialDistance  
if ($(window).scrollTop() > header1_initialDistance) {  
    if (parseInt(header1.css('padding-top'), 10) > header1_initialPadding) {  
        header1.css('padding-top', '' + $(window).scrollTop() - header1_initialDistance + header1_initialPadding);  
    }  
} else {  
    header1.css('padding-top', '' + header1_initialPadding);  
}
```

```
header2_initialDistance  
if ($(window).scrollTop() > header2_initialDistance) {  
    if (parseInt(header2.css('padding-top'), 10) > header2_initialPadding) {  
        header2.css('padding-top', '' + $(window).scrollTop() - header2_initialDistance + header2_initialPadding);  
    }  
} else {  
    header2.css('padding-top', '' + header2_initialPadding);  
}
```

- A classe filha é chamada de **subclasse** ou **classe derivada**. - A classe pai é chamada de **superclasse** ou **classe base**.

## Herança

- A classe filha pode adicionar seus próprios membros, além de herdar os da classe pai. - A herança é representada pela palavra-chave **extends** em Java.

## Conceito









Definição: Herança é um princípio da Orientação a Objetos onde uma classe (classe filha) herda atributos e métodos de outra classe (classe pai).

Vantagem: Promove reutilização de código e estabelece uma relação "é um" entre classes.

Objetivo: Reutilizar código, criar hierarquias de classes e promover a organização e modularidade do código.

## Herança

## Conceito

	Bob Esponja	Pikachu	Homer
Bob Esponja			
Pikachu			
Homer			

## Conceito - Exemplo de código Herança

Superclasse

`Animal {public void latir() {`

`class Animal { public String nome;`

`public void emitirSom() {`

Classe derivada `class Cachorro extends`

`System.out.println("Au au");`

`System.out.println("Som de  
animal");`

`}`

```
}
```

Arquivo 'Animal.java'

```
}}
```

Arquivo Cachorro.java

**Conceito - Exemplo de código** Como ficaria a classe Main com o conceito de herança aplicado? Desenvolva o código de instância do objeto Cachorro, de seu atributo e métodos.

## Herança

Lembre-se de que o objeto Cachorro herda as propriedades e métodos definidos na superclasse Animal.



```
header1_initialDistance  
if ($(window).scrollTop() > header1_initialDistance) {  
    if (parseInt(header1.css('padding-top'), 10) > header1_initialPadding) {  
        header1.css('padding-top', '' + $(window).scrollTop() - header1_initialDistance + header1_initialPadding);  
    }  
} else {  
    header1.css('padding-top', '' + header1_initialPadding);  
}
```

```
header2_initialDistance  
if ($(window).scrollTop() > header2_initialDistance) {  
    if (parseInt(header2.css('padding-top'), 10) > header2_initialPadding) {  
        header2.css('padding-top', '' + $(window).scrollTop() - header2_initialDistance + header2_initialPadding);  
    }  
} else {  
    header2.css('padding-top', '' + header2_initialPadding);  
}
```

# Orientação a Objetos

## Encapsulamento em Interfaces

### Conceito

Interfaces definem contratos que as classes devem implementar, garantindo uma certa estrutura e comportamento.

### Encapsulamento

Interfaces não possuem implementação, apenas definem os métodos a serem implementados pelas classes.

### Benefícios

- Flexibilidade: Permite a implementação de interfaces por diferentes classes.
- Reutilização: Reutilizar a interface para garantir a implementação de um conjunto de métodos.
- Polimorfismo: Facilita o uso de polimorfismo, permitindo chamar métodos de diferentes classes com a mesma interface.

## Interfaces

### Exercício Guiado

## Orientação a Objetos

Crie um código usando uma interface chamada Produto. Essa interface definirá métodos

que todas as classes de produtos devem implementar. A seguir, criamos duas classes: Livro e Eletronico, que representam produtos diferentes e implementam a interface.

## Orientação a Objetos

