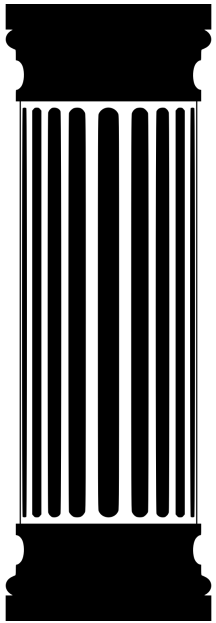
Programação Orientada a Objetos

Os 4 Pilares

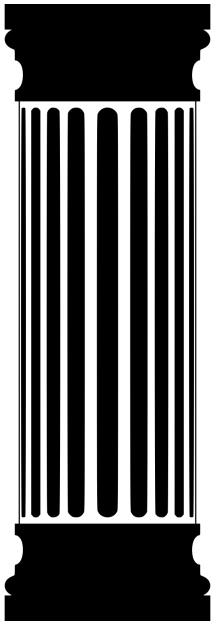
Prof. Leandro Rodrigues Pinto
<leandrorodp@gmail.com>

Programação Orientada a Objetos

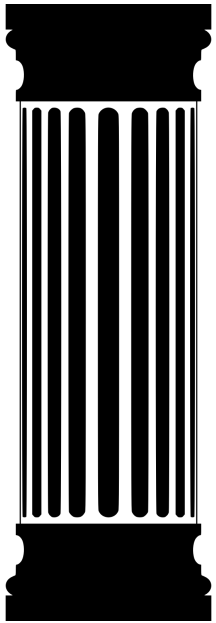
Abstração



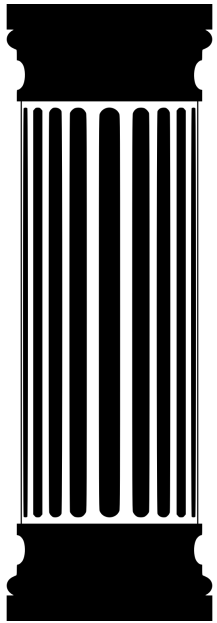
Encapsulamento



Herança

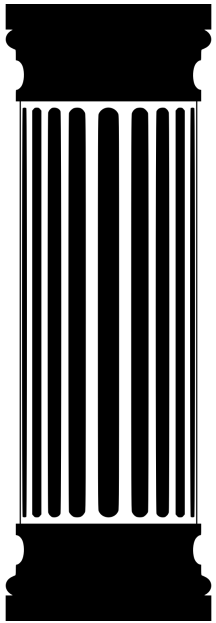


Polimorfismo

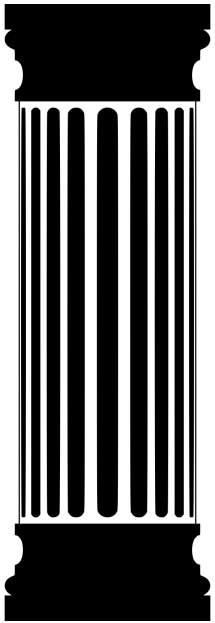


Programação Orientada a Objetos

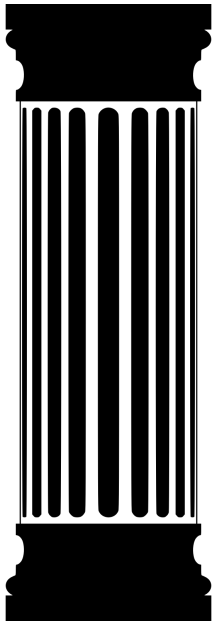
Abstração



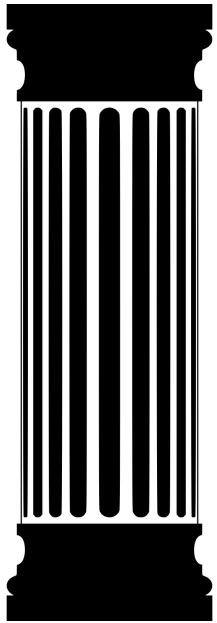
Encapsulamento



Herança



Polimorfismo



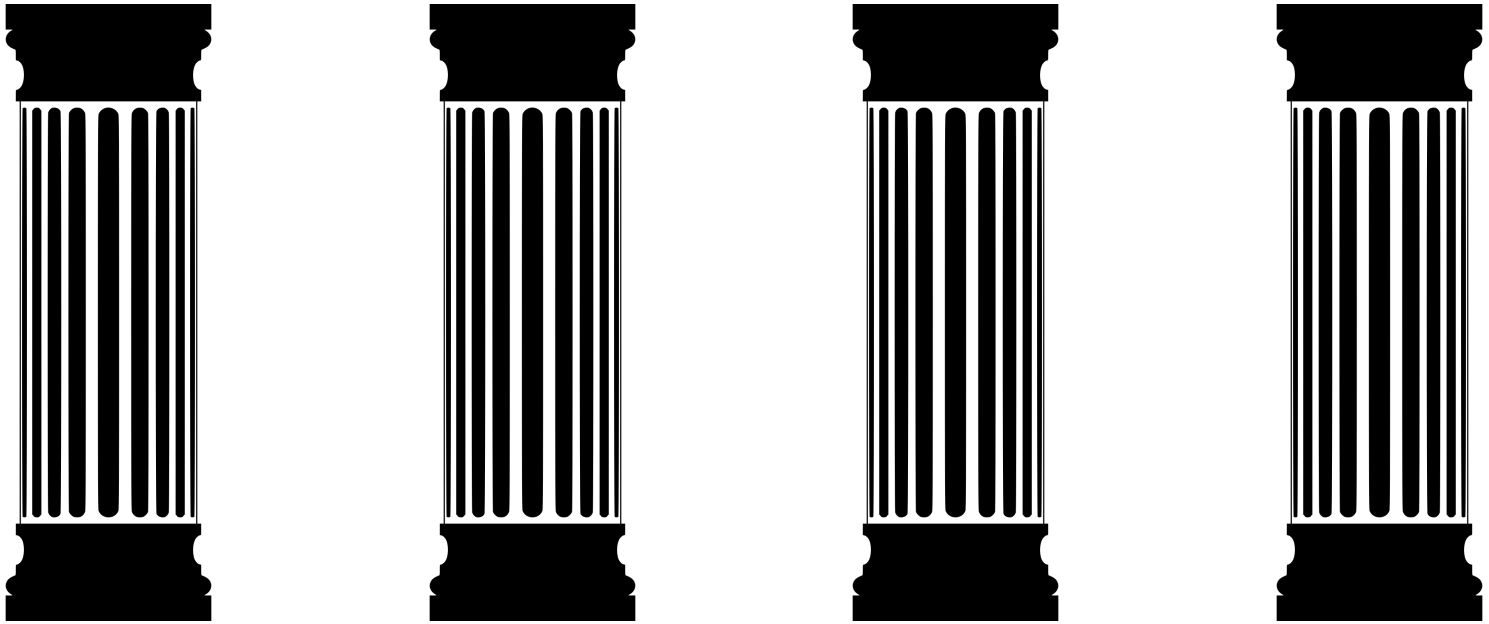
Programação Orientada a Objetos

Abstração

Encapsulamento

Herança

Polimorfismo



Encapsulamento

- Encapsular significa separar o programa em partes, em ‘cápsulas’ autocontidas.
- Os detalhes da implementação ficam **ocultos** ao usuário que passa a utilizar os serviços da classe sem saber como isso ocorre internamente.
- Por quê encapsular?
  - **Melhor controle** dos atributos e métodos da classe
  - Atributos da classe podem ser **read-only** (se implementar somente o método get), or **write-only** (se implementar somente o método set)
  - **Flexibilidade**: o programador pode alterar parte do código sem afetar outras partes (melhor manutenibilidade)
  - Aumento da **segurança** dos dados

Encapsulamento

- As informações só devem estar visíveis para quem deve vê-las
- O encapsulamento envolve diversos aspectos e técnicas de programação em Java, como:
  - modificadores de acesso (Público, Protegido, Padrão e Privado)
  - métodos SET e GET
  - construtores e outros

Encapsulamento

Modificadores de acesso: **Público**

- O nível público é aplicado quando o modificador ***public*** é utilizado.
- O que pode ser público? **Atributos**, **construtores**, **métodos**, **classes** de todos os tipos e **interfaces** de todos os tipos.
- Os itens em nível de visibilidade público podem ser acessados de qualquer lugar do código da aplicação.

Encapsulamento

Modificadores de acesso: **Protegido**

- O nível protegido é aplicado com o modificador ***protected***.
- O que pode ser protegido? **Atributos**, **construtores**, **métodos**, **classes** aninhadas ou **interfaces** aninhadas.
- Os itens em nível de visibilidade protegido só podem ser acessados por código escrito em classes do mesmo pacote da classe na qual eles foram declarados ou por classes derivadas.

Encapsulamento

Modificadores de acesso: **Padrão**

- O nível padrão é aplicado quando **nenhum** modificador é utilizado.
- O que pode ser padrão? **Atributos**, **construtores**, **métodos**, **classes** de todos os tipos e **interfaces** de todos os tipos.
- Os itens em nível de visibilidade padrão só podem ser acessados por código escrito em classes do mesmo pacote da classe na qual eles foram declarados.

Encapsulamento

Modificadores de acesso: **Privado**

- O nível privado é aplicado com o modificador ***private***.
- O que pode ser privado? **Atributos**, **construtores**, **métodos**, **classes aninhadas** ou **interfaces aninhadas**.
- Os itens em nível de visibilidade privado só podem ser acessados por código escrito na mesma classe na qual eles foram declarados.

Encapsulamento

Modificadores de acesso

Modificador	Classe	Pacote	Subclasse	Global
public	SIM	SIM	SIM	SIM
protected	SIM	SIM	SIM	NÃO
“nenhum”	SIM	SIM	NÃO	NÃO
private	SIM	NÃO	NÃO	NÃO

Métodos Get e Set

Getter e Setters

- métodos que têm como finalidade acessar ou alterar as propriedades de um objeto.
- A convenção para esses métodos é de colocar a palavra get ou set antes do nome do atributo.

Construtores

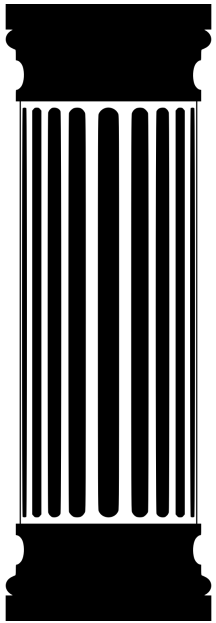
- O Construtor é um método:
  - Obrigatório para toda classe
  - Utilizado para inicializar as variáveis da classe
  - Que pode chamar outros métodos
  - Sem tipo de retorno
  - Que aceita parâmetros
  - Que aceita sobrecarga
- **OBS.: ao criar qualquer construtor, o Java não criará o construtor padrão Classe()**

Atividade prática - Encapsulamento

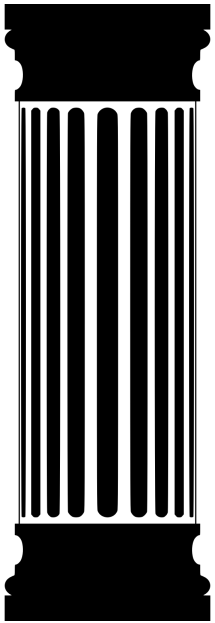
CLASSE CARRO		OBJETO CARRO A	OBJETO CARRO B
Atributos de objeto	Marca	Ford	Mitsubishi
	Modelo	Fiesta	L-200
	Cor	branco	azul royal
	Combustível	gasolina	diesel
Métodos	ligar		
	acelerar		
	frear		

Programação Orientada a Objetos

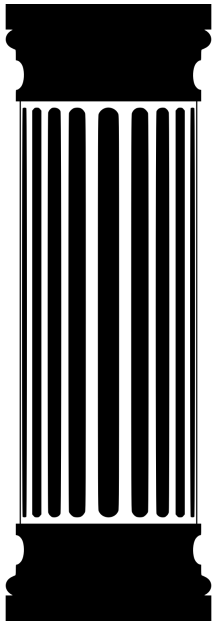
Abstração



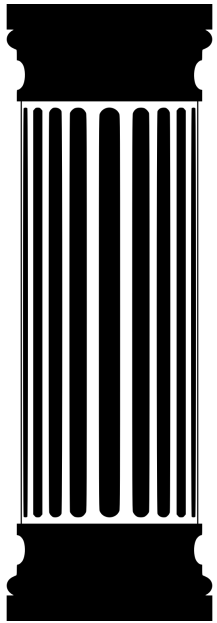
Encapsulamento



Herança



Polimorfismo



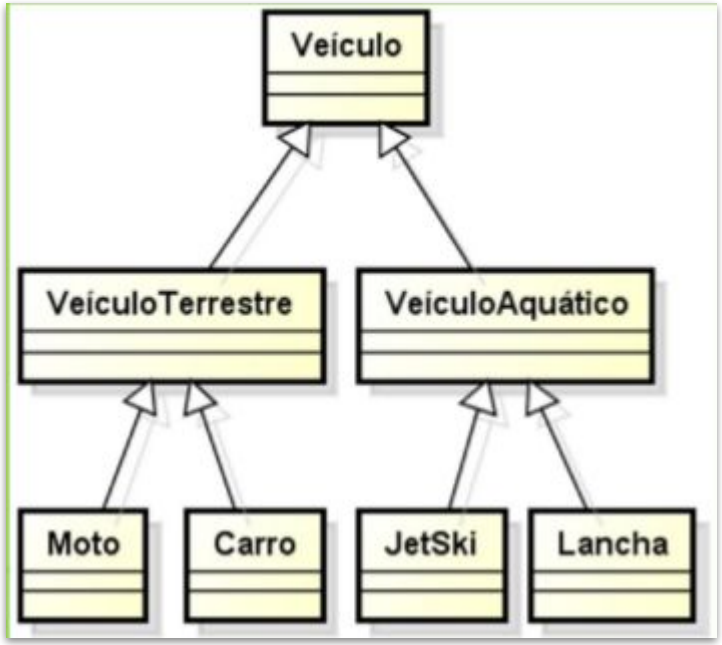
Herança

- O termo se refere a algo herdado.
- Ocorre quando uma classe passa a herdar características (variáveis e métodos) definidas em outra classe.
- A classe genérica é denominada superclasse, classe base ou classe mãe.
- As classes específicas são denominadas subclasses, classes derivadas ou classes filhas.
- Possibilita o compartilhamento ou reaproveitamento de recursos definidos em outra classe.
- Outro termo relacionado com a herança é a especialização.
- Permite implementar características e comportamentos específicos na subclasse.

Herança

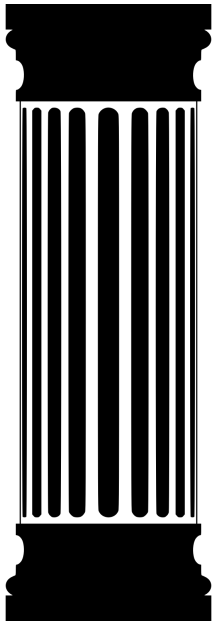
- Conceitos:
  - **Classe Abstrata:** não pode ser instanciada. Só pode servir como superclasse (Mãe / Progenitora).
  - **Método Abstrato:** declarado, mas não implementado na superclasse. Colocado dentro da interface ou classe abstrata.
  - **Classe Final:** Não pode ser herdada por outra classe. Obrigatoriamente uma folha.
  - **Método Final:** não pode ser sobrescrito pelas suas subclasses. Obrigatoriamente herdado.
  - **Herança de implementação (Pobre):** Herda tudo de sua superclasse.
  - **Herança para Diferença:** Adiciona características e comportamentos além dos herdados. Pode também modificar comportamentos da superclasse.

Atividade prática - Herança

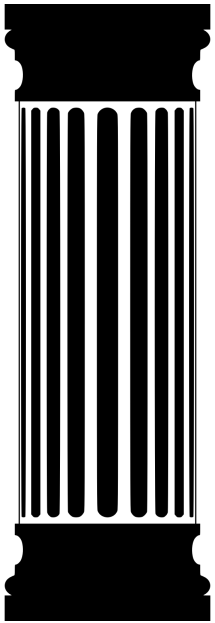


Programação Orientada a Objetos

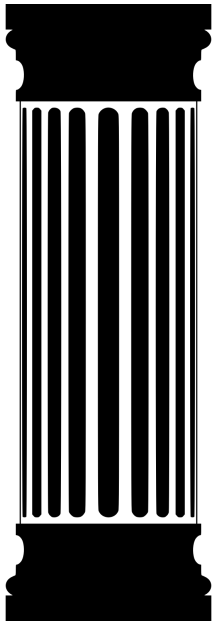
Abstração



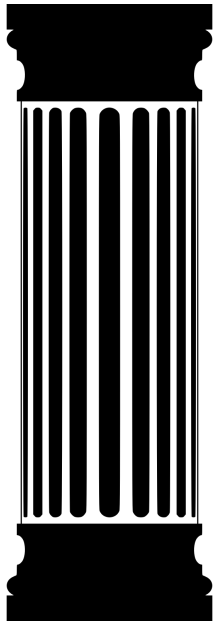
Encapsulamento



Herança



Polimorfismo



Polimorfismo

- É a capacidade de um objeto poder ser referenciado de várias formas.
- Não confundir com **sobrecarga** (**overload**).
- Habilidade de um mesmo tipo de objeto poder realizar ações diferentes ao receber uma mesma mensagem.
- Criação de múltiplas classes com os mesmos métodos e propriedades, mas com funcionalidades e implementações diferentes.
- Ele permite que classes pertencentes a uma mesma linha de herança possuam comportamentos diferentes para o mesmo método
- Oferece um mecanismo para a **generalização**.

Polimorfismo

- O uso de Polimorfismo pressupõe duas condições:
  - A existência de herança.
  - E a redefinição de métodos em todas as classes.
- Todas as classes devem possuir métodos com a mesma assinatura (nome e parâmetros), porém com funcionalidades diferentes.
- Esse mecanismo de redefinição de métodos é conhecido como **overriding**.
- A identificação do método em overriding é feita em tempo de execução (runtime) pelo JAVA

Atividade prática - Polimorfismo

Atividade prática - Polimorfismo

Analise os código java que utiliza **herança** e **polimorfismo** e explique detalhadamente.

```
public class Funcionario {

    private String nome;
    private double salario;

    public double ganhoAnual(){
        double ganho = this.salario * 12;
        return ganho;
    }

    public void exibeDados(){
        System.out.println("Nome: " + nome
                + " Salário: " +salario);
    }
```

Atividade prática - Polimorfismo

```
public class Tecnico extends Funcionario{

    private double bonus = 100;

    public double ganhoAnual(){
        double ganho = (super.getSalario()+bonus)*12;
        return ganho;
    }
```

Atividade prática - Polimorfismo

```
public class TesteFuncionario {

    public static void main(String[] args) {
        Funcionario f = new Tecnico();
        f.setNome("Nickerson");
        f.setSalario(1000);
        f.exibeDados();
        System.out.println(f.ganhoAnual());

        Funcionario f2 = new Funcionario();
        f2.setSalario(1000);
        System.out.println(f2.ganhoAnual());
    }
}
```

Bibliografia

Cadenhead, Rogers; LEMAY, Laura, Aprenda em 21 dias Java 2. 4. ed. São Paulo: Campus, 2005.

DEITEL, Harvey H .; DEITEL, Paul J. Java: como Programar. 8. Ed. São Paulo: Pearson / Prentice-Hall, 2010.

Perguntas?

Obrigado!