🧠 Aula Completa – **Polimorfismo em Java**

🎯 Objetivo da Aula (1 hora)

Ao final da aula, o aluno será capaz de:

- Entender **o que é polimorfismo**
- Saber **por que usar**
- Identificar **quando usar**
- Implementar **polimorfismo corretamente**
- Aplicar em **cenários reais do mercado**
- Testar com **classe Main**

📌 O que é Polimorfismo?

Polimorfismo significa **"muitas formas"**.

Em Java, é a capacidade de **um mesmo método ou referência assumir comportamentos diferentes**, dependendo do **objeto real instanciado**.

> 👉 "Uma referência do tipo pai pode apontar para objetos filhos diferentes."

📌 Tipos de Polimorfismo em Java

1️⃣ Polimorfismo de Sobrescrita (Runtime – MAIS IMPORTANTE)

- Ocorre com **herança ou interface**
- Decidido **em tempo de execução**
- Usa `@Override`

```
Animal animal = new Cachorro();
animal.fazerSom();
```

2️⃣ Polimorfismo de Sobrecarga (Compile Time)

- Métodos com **mesmo nome**, mas **assinaturas diferentes**

- Não é foco de arquitetura (é sintático)

✅ Por que usar Polimorfismo?

✔️ Código **mais flexível**
✔️ Código **mais reutilizável**
✔️ Código **aberto para extensão e fechado para modificação (SOLID – OCP)**
✔️ Facilita manutenção
✔️ Reduz `if/else` e `switch`
✔️ Essencial para **arquitetura profissional**

📍 Onde usar Polimorfismo?

- Regras de negócio
- Serviços
- Pagamentos
- Relatórios
- Integrações
- Camada de domínio
- Frameworks (Spring, Hibernate, Jakarta EE)

🛠️ Como usar Polimorfismo corretamente?

1. Criar **classe abstrata ou interface**
2. Criar **implementações concretas**
3. Trabalhar sempre com o **tipo abstrato**
4. Evitar `instanceof`

5. Deixar o comportamento variar

🔟 EXEMPLOS DO MUNDO REAL (COM CÓDIGO + TESTE)

1️⃣ Pagamento (Cartão, Pix, Boleto)

Interface

```
public interface Pagamento {
    void pagar(double valor);
}
```

Implementações

```
public class PagamentoCartao implements Pagamento {
    public void pagar(double valor) {
        System.out.println("Pagamento no cartão: R$ " + valor);
    }
}
```

```
public class PagamentoPix implements Pagamento {
    public void pagar(double valor) {
        System.out.println("Pagamento via Pix: R$ " + valor);
    }
}
```

Teste (Main)

```
public class Main {
    public static void main(String[] args) {
        Pagamento pagamento = new PagamentoPix();
        pagamento.pagar(150);
    }
}
```

📌 **Explicação:**
A mesma referência `Pagamento` executa comportamentos diferentes.

2️⃣ Funcionário (CLT e PJ)

```
public abstract class Funcionario {
    public abstract double calcularSalario();
}
```

```
public class FuncionarioCLT extends Funcionario {
    public double calcularSalario() {
        return 3000;
    }
}
```

```
public class FuncionarioPJ extends Funcionario {
    public double calcularSalario() {
        return 5000;
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Funcionario f = new FuncionarioPJ();
        System.out.println(f.calcularSalario());
    }
}
```

3️⃣ Transporte (Carro, Moto)

```
public abstract class Transporte {
    public abstract void mover();
}
```

```
public class Carro extends Transporte {
    public void mover() {
        System.out.println("Carro andando");
    }
}
```

```
public class Moto extends Transporte {
    public void mover() {
        System.out.println("Moto andando");
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Transporte t = new Moto();
        t.mover();
    }
}
```

4️⃣ Notificação (Email, SMS)

```
public interface Notificacao {
    void enviar(String msg);
}
```

```
public class Email implements Notificacao {
    public void enviar(String msg) {
        System.out.println("Email: " + msg);
    }
}
```

```
public class SMS implements Notificacao {
    public void enviar(String msg) {
        System.out.println("SMS: " + msg);
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Notificacao n = new SMS();
        n.enviar("Olá!");
    }
}
```

5️⃣ Relatórios (PDF, Excel)

```
public abstract class Relatorio {
    public abstract void gerar();
}
```

```
public class RelatorioPDF extends Relatorio {
    public void gerar() {
        System.out.println("Gerando PDF");
    }
}
```

java

```
public class RelatorioExcel extends Relatorio {
    public void gerar() {
        System.out.println("Gerando Excel");
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Relatorio r = new RelatorioExcel();
        r.gerar();
    }
}
```

6️⃣ Animal (Som diferente)

```
public abstract class Animal {
    public abstract void emitirSom();
}
```

```
public class Cachorro extends Animal {
    public void emitirSom() {
        System.out.println("Latindo");
    }
}
```

```
public class Gato extends Animal {
    public void emitirSom() {
        System.out.println("Miando");
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Animal a = new Gato();
        a.emitirSom();
    }
}
```

7️⃣ Arma em um Jogo

```
public interface Arma {
    void atacar();
}
```

```
public class Espada implements Arma {
    public void atacar() {
        System.out.println("Ataque com espada");
    }
}
```

```
public class Arco implements Arma {
    public void atacar() {
        System.out.println("Ataque com arco");
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Arma arma = new Arco();
        arma.atacar();
    }
}
```

8️⃣ Frete (Normal, Expresso)

```
public abstract class Frete {
    public abstract double calcular();
}
```

```
public class FreteNormal extends Frete {
    public double calcular() {
        return 20;
    }
}
```

```
public class FreteExpresso extends Frete {
    public double calcular() {
        return 50;
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Frete frete = new FreteExpresso();
        System.out.println(frete.calcular());
    }
}
```

9️⃣ Log (Arquivo, Console)

```
public interface Logger {
    void log(String msg);
}
```

```
public class ConsoleLogger implements Logger {
    public void log(String msg) {
        System.out.println(msg);
    }
}
```

```
public class FileLogger implements Logger {
    public void log(String msg) {
        System.out.println("Gravando em arquivo: " + msg);
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Logger logger = new ConsoleLogger();
        logger.log("Erro encontrado");
    }
}
```

🔟 Sistema Bancário (Conta)

```
public abstract class Conta {
    public abstract void sacar(double valor);
}
```

```
public class ContaCorrente extends Conta {
    public void sacar(double valor) {
        System.out.println("Saque conta corrente: " + valor);
    }
}
```

```
public class ContaPoupanca extends Conta {
    public void sacar(double valor) {
        System.out.println("Saque conta poupança: " + valor);
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Conta conta = new ContaPoupanca();
        conta.sacar(100);
    }
}
```

🧩 Resumo Final para o Aluno

- ✔ Polimorfismo = comportamento variável
- ✔ Sempre programar para **interfaces ou classes abstratas**
- ✔ Evitar `if/else`
- ✔ Essencial para código profissional
- ✔ Base de frameworks como Spring e Jakarta EE