Encapsulamento em Java

1️⃣ O que é Encapsulamento?

Encapsulamento é o princípio da POO que determina que:

> **Os dados de um objeto devem ser protegidos contra acesso direto e modificações indevidas.**

Ou seja:

- O **estado interno** do objeto fica **escondido**
- O acesso acontece **apenas por métodos controlados**

🔒 Em Java, encapsulamento é feito com:

- `private` → protege os atributos
- `public` → expõe comportamentos
- `protected` / *default* → controle de acesso avançado
- **Getters e Setters**
- **Validações internas**

2️⃣ Por que usar Encapsulamento?

❌ Sem encapsulamento (ERRADO)

```
public class Conta {
    public double saldo;
}
```

Qualquer um pode fazer:

```
conta.saldo = -100000;
```

💥 **Problemas**:

- Dados inválidos
- Código frágil
- Falta de regras
- Sistema inseguro

✅ Com encapsulamento (CORRETO)

```
public class Conta {
    private double saldo;

    public void depositar(double valor) {
        if (valor > 0) {
            saldo += valor;
        }
    }
}
```

✔️ Regras centralizadas
✔️ Segurança
✔️ Manutenção fácil
✔️ Código profissional

3️⃣ Vantagens do Encapsulamento

✅ Proteção dos dados
✅ Regras de negócio no lugar certo
✅ Código mais seguro
✅ Facilidade de manutenção
✅ Facilita testes
✅ Facilita refatoração
✅ Base para SOLID
✅ Essencial em APIs, Spring, JPA

👉 **No mercado, código sem encapsulamento é considerado amador.**

4️⃣ Onde usar Encapsulamento?

Você usa encapsulamento em:

✔️ Entidades
✔️ DTOs
✔️ Services
✔️ APIs REST
✔️ Domínio de negócio
✔️ Frameworks (Spring, Hibernate, Jakarta EE)
✔️ Sistemas bancários
✔️ Sistemas públicos (prefeituras, saúde, educação)

📌 **Resumo:**
👉 **Toda classe que representa algo do mundo real deve ser encapsulada.**

5️⃣ Como usar Encapsulamento corretamente?

Regras práticas:

1. **Atributos sempre** `private`
2. Expor apenas o necessário
3. Validar dados no setter ou método
4. Nunca deixar o estado inválido

5. Pensar em comportamento, não só dados

6️⃣ 10 Exemplos Reais de Encapsulamento (com Teste)

🧩 Exemplo 1 – Conta Bancária

Classe `ContaBancaria`

```
public class ContaBancaria {

    private double saldo;

    public double getSaldo() {
        return saldo;
    }

    public void depositar(double valor) {
        if (valor <= 0) {
            throw new IllegalArgumentException("Valor inválido");
        }
        saldo += valor;
    }

    public void sacar(double valor) {
        if (valor > saldo) {
            throw new IllegalArgumentException("Saldo
insuficiente");
        }
        saldo -= valor;
    }
}
```

Classe `Main`

```
public class MainConta {
    public static void main(String[] args) {
        ContaBancaria conta = new ContaBancaria();
        conta.depositar(500);
        conta.sacar(200);

        System.out.println(conta.getSaldo());
    }
}
```

🧠 **Encapsulamento protege o saldo e garante regras financeiras.**

🧩 Exemplo 2 – Usuário do Sistema

Classe `Usuario`

```
java

public class Usuario {

    private String login;
    private String senha;

    public Usuario(String login, String senha) {
        setLogin(login);
        setSenha(senha);
    }

    public String getLogin() {
        return login;
    }

    private void setLogin(String login) {
        if (login == null || login.isBlank()) {
            throw new IllegalArgumentException("Login inválido");
        }
        this.login = login;
    }

    private void setSenha(String senha) {
        if (senha.length() < 6) {
            throw new IllegalArgumentException("Senha fraca");
        }
        this.senha = senha;
    }
}
```

Classe `Main`

```
public class MainUsuario {
    public static void main(String[] args) {
        Usuario user = new Usuario("admin", "123456");
        System.out.println(user.getLogin());
    }
}
```

🧠 **A senha nunca é exposta. Segurança aplicada.**

🧩 Exemplo 3 – Produto com Estoque

```
public class Produto {

    private int estoque;

    public int getEstoque() {
        return estoque;
    }

    public void adicionarEstoque(int quantidade) {
        if (quantidade <= 0) return;
        estoque += quantidade;
    }

    public void removerEstoque(int quantidade) {
        if (quantidade > estoque) {
            throw new RuntimeException("Estoque insuficiente");
        }
        estoque -= quantidade;
    }
}
```

```
public class MainProduto {
    public static void main(String[] args) {
        Produto p = new Produto();
        p.adicionarEstoque(10);
        p.removerEstoque(3);
        System.out.println(p.getEstoque());
    }
}
```

🧩 Exemplo 4 – CPF Validado

```
public class Pessoa {

    private String cpf;

    public void setCpf(String cpf) {
        if (cpf == null || cpf.length() != 11) {
            throw new IllegalArgumentException("CPF inválido");
        }
        this.cpf = cpf;
    }

    public String getCpf() {
        return cpf;
    }
}
```

```
public class MainPessoa {
    public static void main(String[] args) {
        Pessoa p = new Pessoa();
        p.setCpf("12345678901");
        System.out.println(p.getCpf());
    }
}
```

🧩 Exemplo 5 – Temperatura Controlada

```
java

public class SensorTemperatura {

    private double temperatura;

    public double getTemperatura() {
        return temperatura;
    }

    public void atualizar(double novaTemp) {
        if (novaTemp < -50 || novaTemp > 100) {
            throw new IllegalArgumentException("Temperatura
inválida");
        }
        temperatura = novaTemp;
    }
}
```

```
public class MainSensor {
    public static void main(String[] args) {
        SensorTemperatura s = new SensorTemperatura();
        s.atualizar(25);
        System.out.println(s.getTemperatura());
    }
}
```

🧩 Exemplo 6 – Pedido com Status

```
public class Pedido {

    private String status = "ABERTO";

    public String getStatus() {
        return status;
    }

    public void finalizar() {
        status = "FINALIZADO";
    }
}
```

```
public class MainPedido {
    public static void main(String[] args) {
        Pedido p = new Pedido();
        p.finalizar();
        System.out.println(p.getStatus());
    }
}
```

🧩 Exemplo 7 – Limite de Crédito

```
public class Cliente {

    private double limiteCredito;

    public void setLimiteCredito(double valor) {
        if (valor < 0) return;
        limiteCredito = valor;
    }

    public double getLimiteCredito() {
        return limiteCredito;
    }
}
```

```
public class MainCliente {
    public static void main(String[] args) {
        Cliente c = new Cliente();
        c.setLimiteCredito(1000);
        System.out.println(c.getLimiteCredito());
    }
}
```

🧩 Exemplo 8 – Salário Protegido

```
public class Funcionario {

    private double salario;

    public double getSalario() {
        return salario;
    }

    public void reajustar(double percentual) {
        salario += salario * percentual;
    }
}
```

```
public class MainFuncionario {
    public static void main(String[] args) {
        Funcionario f = new Funcionario();
        f.reajustar(0.1);
        System.out.println(f.getSalario());
    }
}
```

🧩 Exemplo 9 – Configuração do Sistema

```
public class Configuracao {

    private static String ambiente = "PRODUCAO";

    public static String getAmbiente() {
        return ambiente;
    }

    public static void setAmbiente(String novo) {
        ambiente = novo;
    }
}
```

```
public class MainConfig {
    public static void main(String[] args) {
        Configuracao.setAmbiente("HOMOLOG");
        System.out.println(Configuracao.getAmbiente());
    }
}
```

🧩 Exemplo 10 – API Token

```
public class ApiToken {

    private String token;

    public ApiToken(String token) {
        this.token = token;
    }

    public boolean isValido() {
        return token != null && token.length() > 20;
    }
}
```

```
public class MainToken {
    public static void main(String[] args) {
        ApiToken token = new ApiToken("abc12345678901234567890");
        System.out.println(token.isValido());
    }
}
```

✅ Conclusão da Aula

📌 **Encapsulamento não é opcional**
📌 É base de:

- Segurança
- Performance
- Arquitetura
- Clean Code
- APIs profissionais

👉 **Quem não encapsula, não escreve código de mercado.**