

Encapsulamento



Prof. Tiago Sombra



Agenda

- Encapsulamento
 - Encapsulando
 - Boas Práticas
-



Encapsulamento



- O que é encapsular:
 - Parte da ideia de colocar em uma capsula
- Em programação:
 - Lida com o conceito de **agrupar partes de um programa, o mais isoladas possível**
 - **Apresentando** para o restante do sistema **somente o que é importante e nada mais**

Encapsulamento



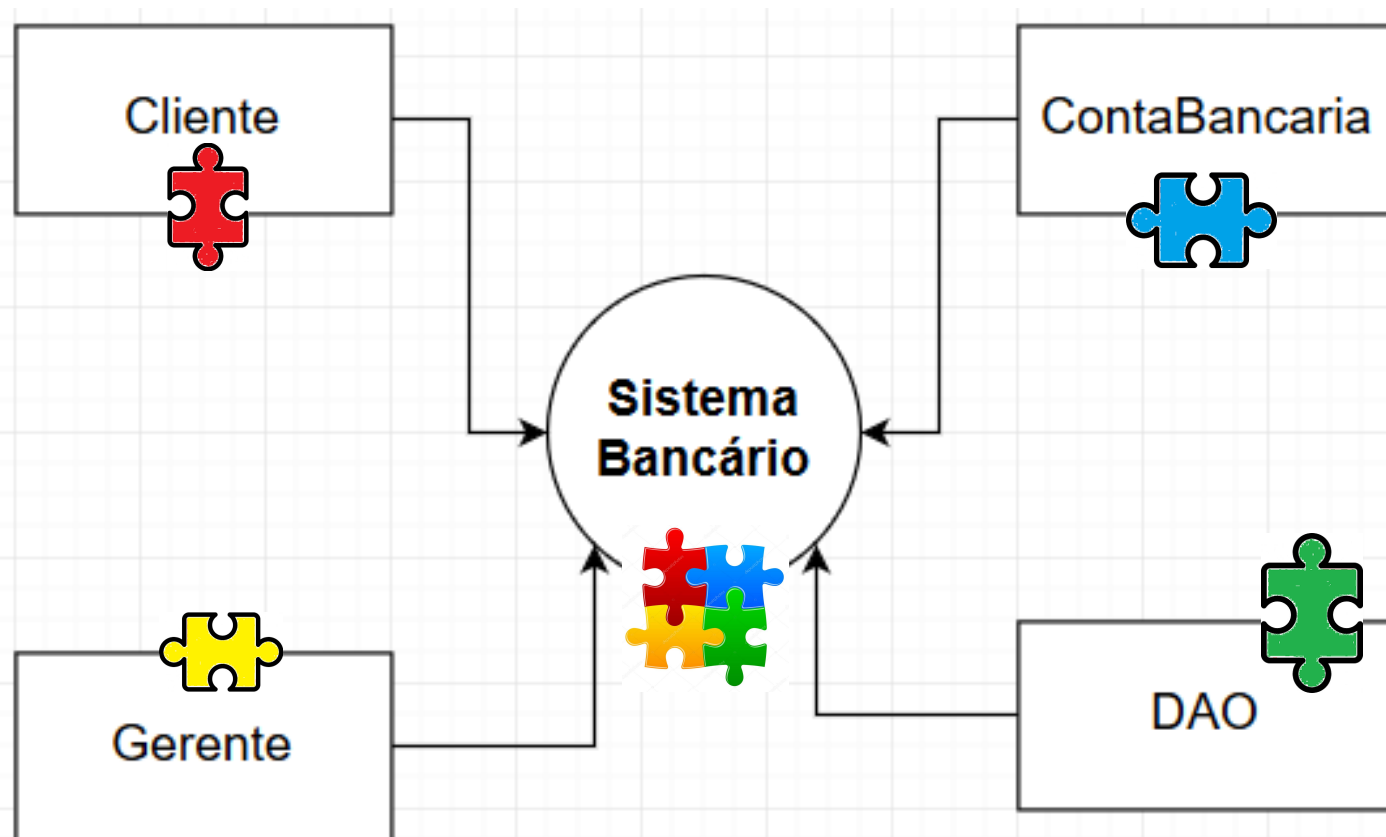
- Imagine um sistema bancário como exemplo.
- Esse sistema deve lidar com várias informações sobre:
 - Clientes
 - Contas
 - Banco de dados
 - Gerentes
 - Funcionários
 - ...
- Cada parte/módulo do sistema pode ser implementada para lidar com essas informações:
 - Classes do sistema

Encapsulamento

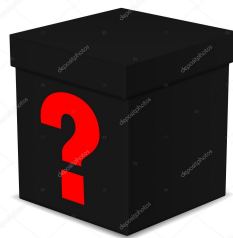


A classe **ContaBancaria** é responsável por manter informações sobre uma conta

A classe **Cliente** é responsável por manter informações sobre um cliente



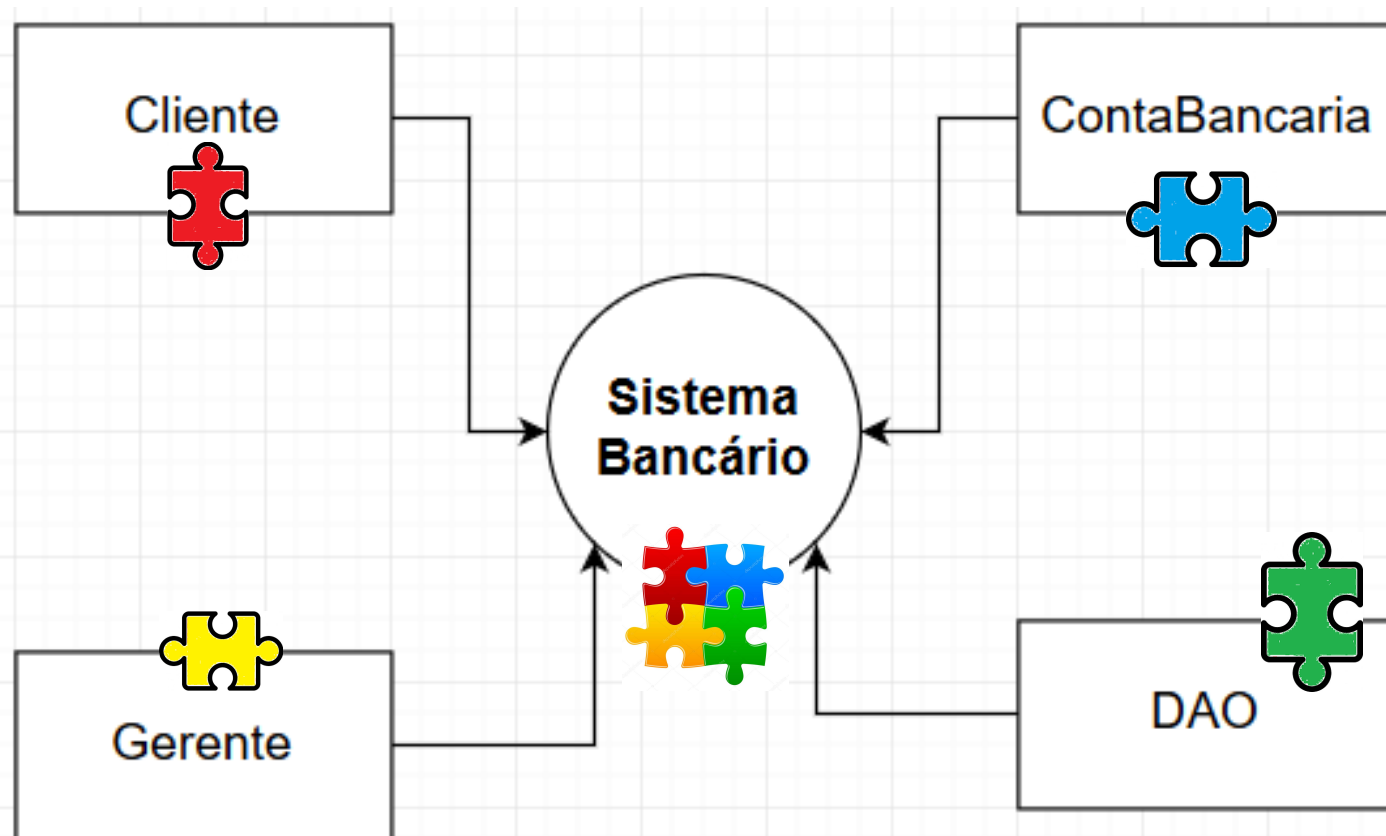
Encapsulamento



A classe **ContaBancaria** é responsável por manter informações sobre uma conta

A classe **Cliente** é responsável por manter informações sobre um cliente

Cada classe **encapsula** uma função



Encapsulando



Encapsulando

- Para garantir o encapsulamento é necessário tornar cada parte o mais isolada possível:
 - Mantendo suas informações “seguras”
- Ao alterar a visibilidade de um atributo da classe outras partes do programa (outras classes) podem deixar de ver as suas propriedades
 - Perdem o acesso direto à informação
 - Conceito de caixa-preta



Encapsulando

- Para cada atributo *private*, visível somente dentro da própria classe, são utilizados os métodos de acesso:
 - GET – para obter/ler as informações do objeto
 - SET – para alterar/escrever informações no objeto
- Essa não é uma regra de programação e sim uma boa prática
 - O uso vai depender do contexto o qual está sendo manipulado



Encapsulando

```
public class ContaBancaria{  
    public float saldo;  
    ContaBancaria(float saldo){  
        this.saldo = saldo  
    }  
}
```



Encapsulando

```
public class ContaBancaria{  
    public float saldo;  
    ContaBancaria(float saldo){  
        this.saldo = saldo  
    }  
}
```



Atributos públicos vão contra as boas práticas!!

Todo atributo deve ser privado e devemos criar métodos GET para obter as informações e SET para alterar.



Encapsulando

```
public class ContaBancaria{  
    public float saldo;  
    ContaBancaria(float saldo){  
        this.saldo = saldo  
    }  
}
```



Isso permitiria por exemplo, que na classe Sistema o valor da conta fosse alterado diretamente

```
ContaBancaria conta = new ContaBancaria(0);
```

```
conta.saldo = 200.0f;
```



Encapsulando

```
public class ContaBancaria{  
    private float saldo;  
    ContaBancaria(float saldo){  
        this.saldo = saldo;  
    }  
    public void setSaldo(float saldo){  
        this.saldo = saldo;  
    }  
    public float getSaldo(){  
        return saldo;  
    }  
}
```



E agora tudo bem!?!?



Encapsulando

```
public class ContaBancaria{  
    private float saldo;  
    ContaBancaria(float saldo){  
        this.saldo = saldo;  
    }  
    public void setSaldo(float saldo){  
        this.saldo = saldo;  
    }  
    public float getSaldo(){  
        return saldo;  
    }  
}
```



Sim em parte.

Ainda é possível alterar o atributo sem qualquer verificação e garantia de consistência

- Imaginem que toda vez que o cliente realizar um depósito será necessário alterar o saldo da conta.



Encapsulando

```
public class ContaBancaria{  
    private float saldo;  
    ContaBancaria(float saldo){  
        this.saldo = saldo;  
    }  
    public void setSaldo(float saldo){  
        this.saldo = saldo;  
    }  
    public float getSaldo(){  
        return saldo;  
    }  
}
```



E agora tudo bem!?!? Sim em parte!

Ainda é possível alterar o atributo sem qualquer verificação e garantia de consistência


- Imaginem que toda vez que o cliente realizar um depósito será necessário alterar o saldo da conta.

```
ContaBancaria conta = new ContaBancaria(0);  
conta.setSaldo(200.0f);
```

Como realizar um depósito ou saque de forma segura?



```
public class ContaBancaria{  
    private float saldo;  
    ContaBancaria(float saldo){  
        this.saldo = saldo;  
    }  
    public void setSaldo(float saldo){  
        this.saldo = saldo;  
    }  
    public float getSaldo(){  
        return saldo;  
    }  
}
```



```
public void deposito(float deposito){  
    if(deposito>0)  
        saldo = saldo + deposito;  
}  
public boolean saque(float saque){  
    if(saque>saldo){  
        return false;  
    }else{  
        saldo = saldo-saque;  
        return true;  
    }  
}  
}
```

O método setSaldo já não é mais necessário. As operações de deposito e saque estão encapsuladas nos respectivos métodos


```
public class ContaBancaria{  
    private float saldo;  
    ContaBancaria(float saldo){  
        this.saldo = saldo;  
    }  
    public float getSaldo(){  
        return saldo;  
    }  
    public void deposito(float deposito){  
        if(deposito>0)  
            saldo = saldo + deposito;  
    }  
}
```

```
public boolean saque(float saque){  
    if(saque>saldo){  
        return false;  
    }else{  
        saldo = saldo-saque;  
        return true;  
    }  
}  
}
```



Encapsulando

```
ContaBancaria conta = new ContaBancaria(0);
```

```
conta.deposito(200);
```

```
System.out.println(conta.getSalto());
```

```
if(conta.saque(100)){
```

```
    System.out.println("Saque realizado");
```

```
}else{
```

```
    System.out.println("Saldo insuficiente");
```

```
}
```

A classe ContaBancaria
agora encapsula as
operações sobre uma
conta bancária de um
cliente do banco





Boas Práticas!

Boas Práticas!

- O uso do *private* e a implementação de métodos setters e getters **não garantem totalmente o encapsulamento**
 - O objetivo de utilizar o encapsulamento é **criar mecanismos mais inteligentes** para acessar os atributos da classe
- Uma **boa prática** para ter certeza se o encapsulamento está correto é olhar para o código
 - Está encapsulado quando conseguir disser **o que** o método faz mas não conseguir dizer **como** ele faz



Boas Práticas!

- **Boas práticas:**

- Não implemente todos os métodos GETTERS e SETTERS logo de cara
 - Busque identificar onde e como cada atributo será utilizado
 - Desenvolva métodos com uma regra de negócio adequada
-
- O conceito de encapsulamento facilita o desenvolvimento
 - Evita que várias partes do objeto sejam alteradas fora da classe
 - Concentra as modificações de um objeto em apenas um local! Encapsulando o comportamento dos métodos e atributos da classe

