Programação Orientada a Objetos: POO

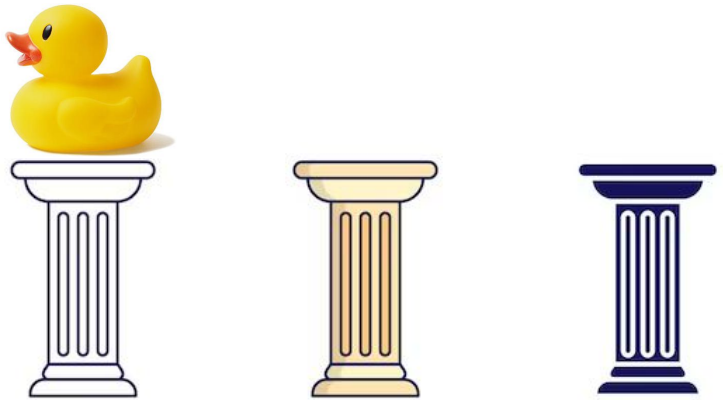
Encapsulamento / Interface

Programação Orientada a Objetos: POO

Encapsulamento / Interface

Encapsulamento / Interface

O 1º Pilar da POO



Encapsulamento

Proteção

Interna > Externa
Externa > Interna

Encapsulamento

Padronização

Formato padrão e comum para diversas outras coisas que dependem deste item;

Mesmo que de outra marca, externos, o padrão se mantém, então a utilização é igual.

Encapsulamento

Facilidade

Não preciso me preocupar com o que tem dentro, com a forma que está estruturada ou seus componentes;

Somente preciso saber como utilizar a sua função, como usar a cápsula.

Encapsulamento

Comunicação

É possível se comunicar e trocar informações com a cápsula

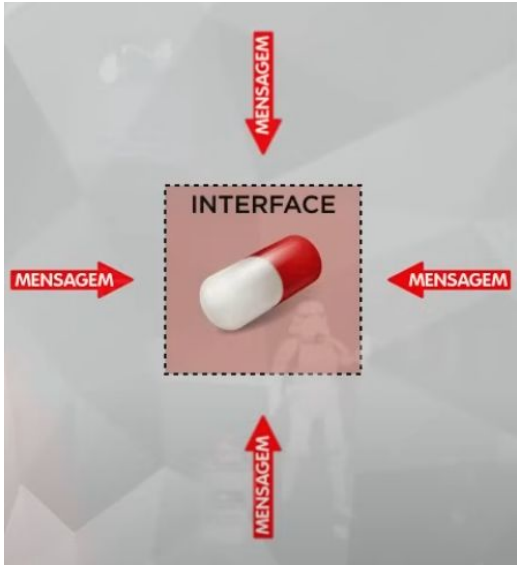
- Através de Mensagens
  - Não acesso o interior da cápsula
  - Solicito informação (energia/ medicamento) e recebo um retorno (energia/ medicamento)

Encapsulamento

Interface - o 1º pilar da Programação Orientada a Objetos

Lista de serviços fornecidos por um componente;

É o contato com o mundo exterior, que define o que pode ser feito com uma objeto dessa classe.



Encapsulamento

Interface

Interface é o nome que damos para a forma/ para as instruções de como eu devo me comunicar com a cápsula;

Um bom objeto encapsulado, possui uma interface bem definida;

- Na pilha, a interface é: Polo de Entrada / Polo de Saída;
- Tenho de conectar cada polo no seu devido lugar, mas somente isso.

Encapsulamento

Interface

Interface: Não é obrigatória, mas possui vantagens:

- Tornar mudanças invisíveis - posso alterar a cápsula por outra de mesmo modelo, sem alterar nada em meu código;
- Facilitar a reutilização do código - a cápsula se encaixa em diversos códigos que esperam aquele modelo;
- Reduzir os efeitos colaterais - como o código não encosta diretamente no interior da cápsula, erros por alterações são evitados;

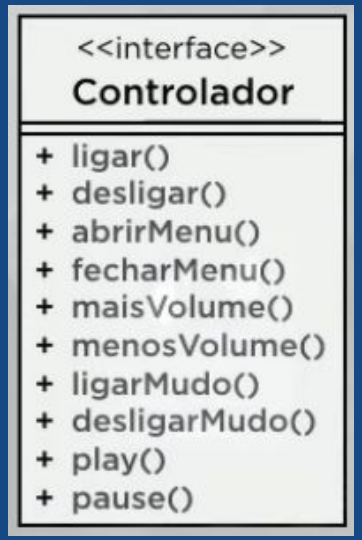
Encapsulamento

Interface

Mais um exemplo:

Encapsulamento

Como Encapsular?



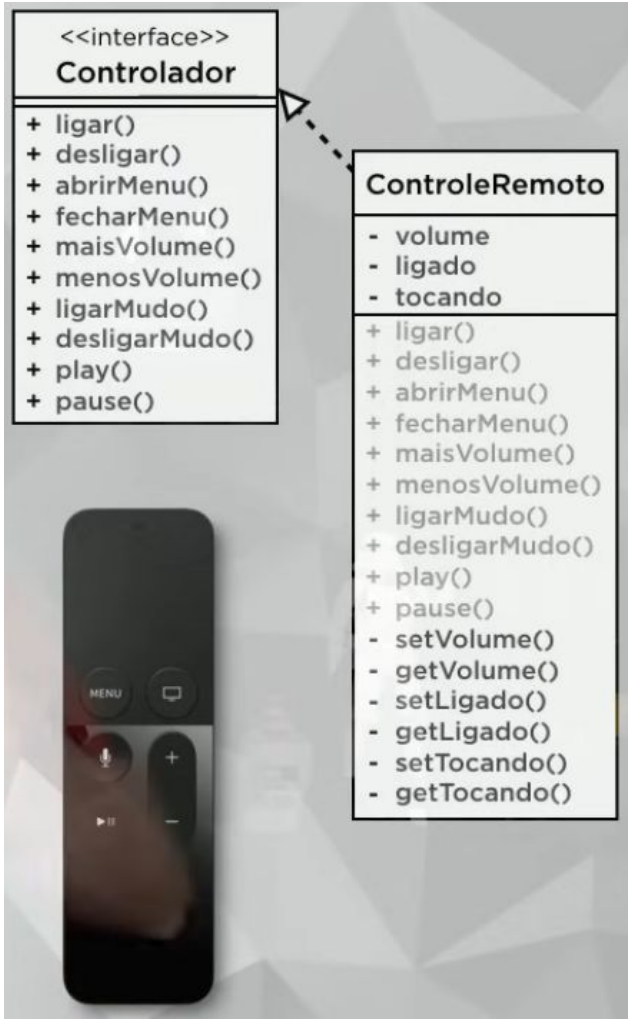
Utilizamos UML:

- Não possui atributos, apenas métodos;
- Inicia com <<interface>> para referenciar/diferenciar do UML de Classes;
- Logo abaixo, o nome da Interface;
- No corpo, os métodos;
- **A Interface não diz como faz, apenas diz o que faz;**
- Todos os métodos são públicos.

Encapsulamento

Como Utilizar?

- Definir os atributos como privados ou, no máximo, protegidos;
- Dizer qual Classe vai Implementar qual Interface (através do "*implements*");
- Implementar os métodos abstratos (vindos da Interface);
- Implementar os métodos especiais (getters e setters);
- métodos abstratos: Não serão desenvolvidos neste local, apenas utilizados;


<<interface>>
Controlador
+ ligar()
+ desligar()
+ abrirMenu()
+ fecharMenu()
+ maisVolume()
+ menosVolume()
+ ligarMudo()
+ desligarMudo()
+ play()
+ pause()
ControleRemoto
- volume
- ligado
- tocando
+ ligar()
+ desligar()
+ abrirMenu()
+ fecharMenu()
+ maisVolume()
+ menosVolume()
+ ligarMudo()
+ desligarMudo()
+ play()
+ pause()
- setVolume()
- getVolume()
- setLigado()
- getLigado()
- setTocando()
- getTocando()
MENU

```
// pseudo código
interface Controlador
    // definição dos métodos abstratos (todos públicos)
    publico abstrato Metodo ligar()
    publico abstrato Metodo desligar()
    publico abstrato Metodo abrirMenu()
    publico abstrato Metodo fecharMenu()
    publico abstrato Metodo maisVolume()
    publico abstrato Metodo menosVolume()
    publico abstrato Metodo ligarMudo()
    publico abstrato Metodo desligarMudo()
    publico abstrato Metodo play()
    publico abstrato Metodo pause()
FimInterface
```

https://github.com/lucassasse/AulasPOO

```
// pseudo código
// implement: obriga a Classe a implementar a Interface
classe ControleRemoto implement Controlador
    // atributos
    privado inteiro volume
    privado boolean ligado
    privado boolean tocando

    // métodos especiais
    publico Metodo Construtor()
        volume = 50
        ligado = falso
        tocando = falso
    fimMetodo

    privado Metodo getVolume()
        retorne this.volume
    fimMetodo
    privado Metodo getLigado()
        retorne this.ligado
    fimMetodo
    privado Metodo getTocando()
        retorne this.tocando
    fimMetodo
    privado Metodo setVolume(v: int)
        this.volume = v
    fimMetodo
    privado Metodo setLigado(l: boolean)
        this.ligado = l
    fimMetodo
    privado Metodo setTocando(t: boolean)
        this.tocando = t
    fimMetodo
```

```
    // sobrescrevendo métodos
    publico Metodo ligar()
        setLigado(verdadeiro)
    fimMetodo
    publico Metodo desligar()
        setLigado(falso)
    fimMetodo
    publico Metodo abrirMenu()
        Escreva(getLigado())
        Escreva(getVolume())
        Escreva(getTocando())
    fimMetodo
    publico Metodo fecharMenu()
        Escreva("Menu fechado...")
    fimMetodo
    publico Metodo maisVolume()
        Se(getLigado()) então
            setVolume(getVolume + 1)
            Excreva(getVolume())
        FimSe
    fimMetodo
    publico Metodo menosVolume()
        Se(getLigado()) então
            setVolume(getVolume - 1)
        Excreva(getVolume())
        FimSe
    fimMetodo
    publico Metodo ligarMudo()
        Se(getLigado() e 'getVolume() > 1') então
            setVolume(0)
        FimSe
    fimMetodo
    publico Metodo desligarMudo()
        Se(getLigado() e 'getVolume() == 0) então
            setVolume(50)
        FimSe
    fimMetodo
    publico Metodo play()
        Se(getLigado() e 'não getTocando()') então
            setTocando(verdadeiro)
        fimSe
    fimMetodo
    publico Metodo pause()
        Se(getLigado() e getTocando()) então
            setTocando(falso)
        fimSe
    fimMetodo
FimClasse
```

Atividade avaliativa n1.4

a. Crie uma Classe **BancoFinanceiro**;

b. Crie uma Interface para essa Classe;

c. Defina os Atributos, Métodos Especiais e Métodos Abstratos dentro da Classe, implementando a Interface;

d. Instancie pelo menos uma conta (na Main, utilizando a Classe Banco) e realize algumas operações, através dos métodos abstratos implementados na Classe através da Interface;