



Programação Orientada a Objetos

Encapsulamento

Ely – ely.miranda@ifpi.edu.br

Encapsulamento

- É uma técnica utilizada para esconder detalhes internos de um objeto para o usuário;
- Com encapsulamento, obtém-se uma visão simplificada e protegida da classe;
- Um objeto deve “encapsular” todo o estado e expõe apenas parte do seu comportamento.

Encapsulamento

- Um objeto deve “encapsular” todo o estado e expõe apenas parte do seu comportamento:
 - Nem todo elemento de uma classe deve ser visível (e alterável indiscriminadamente);
 - Dados importantes devem ser protegidos do acesso indevido (de programadores).

Encapsulamento

- Reator com atributos potência e limite máximo.



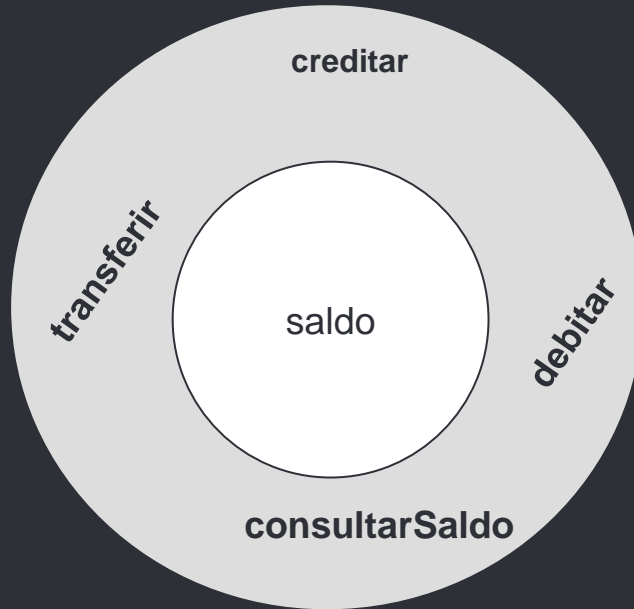
Encapsulamento

- Banco com várias contas.



Encapsulamento

- Conta com o atributo saldo.



Encapsulamento

- E se o acesso a esses atributos fosse restritos?*




Encapsulamento

- Nos três exemplos anteriores, é importante ocultar o estado pois:
 - Nenhum dos atributos mencionados deve ser manipulado diretamente sem validações;
 - Apenas métodos que mantenham integridade devem ter acesso aos dados;
 - A leitura dos dados pode ser liberada, mas a escrita deve ser criteriosa.

Encapsulamento

```
class Conta {  
    numero: string;  
    saldo: number;  
    //...  
    sacar(valor: number): void {  
        if (this.saldo >= valor) {  
            this.saldo = this.saldo - valor;  
        }  
    }  
    //...  
}
```



*Usar o método sacar, previne que o saldo
fique em um estado inconsistente*

No entanto os atributos ainda são acessíveis!

Exemplo de violação

```
let c: Conta = new Conta("1", 100);  
c.saldo = -2000;
```



O acesso direto ao atributo permite validação

Solução

- Obrigar ao utilizador a chamar o método sacar e não permitir o acesso direto ao atributo;
- Para isso, declarar os atributos usando a palavra `private`.

Solução

```
class Conta {  
    private numero: string;  
    private saldo: number;  
    //...  
    sacar(valor: number): void {  
        if (this.saldo >= valor) {  
            this.saldo = this.saldo - valor;  
        }  
    }  
    //...  
}
```

Exemplo de violação

```
let c: Conta = new Conta("1", 100);  
c.saldo = -2000;
```



Erro de compilação: a propriedade 'saldo' é particular e somente é acessível na classe 'Conta'.

Modificadores de acesso

- Modificadores de acesso são palavras reservadas utilizados para garantir o encapsulamento;
- Podem ser aplicados a classes, atributos, métodos e construtores individualmente.

Modificadores de acesso

- Modificadores existentes:
 - public: é visível em qualquer lugar;
 - protected: só é visível na mesma classe e em suas subclasses;
 - private: só é visível dentro da mesma classe.

Modificadores de acesso

- E quando não se especifica um modificador?
 - Tem-se a visibilidade public;
 - É o padrão;

Modificadores de acesso

- Nos construtores, podemos especificar a visibilidade dos atributos:

```
class Alo {  
    constructor(private nome: string) { }  
  
    dizerAlo() {  
        console.log("Alô, " + this.nome);  
    }  
}
```

Métodos de acesso

- Métodos de acesso get e set podem ser criados para ler e escrever em atributos;
- Uma convenção é definir atributos privados com um “_” antes do nome.

Métodos de acesso

```
class Post {  
    constructor(private _text: string) {}  
    get text(): string {  
        return this._text;  
    }  
    set text(text: string) {  
        this._text = text;  
    }  
}
```

Métodos de acesso

- Os métodos de acesso definem algo como “propriedades” ;
- O acesso ao atributo fica então encapsulado pelo nome dado aos métodos de acesso:

```
let p = new Post("post text");  
p.text = "reviewed text";  
console.log(p.text);
```

Modificador readonly

- O modificador readonly cria um atributo somente leitura;
- Nota: não é possível declarar um atributo como const:

```
class Post {  
    constructor(private _text: string,  
                readonly owner: String) {}  
    //métodos de acesso de _texto omitidos  
}
```


Modificador readonly

```
let p = new Post("post text", "Ely");
```

```
p.owner = "new owner";
```

```
console.log(p.text);
```

Cannot assign to 'autor'
because it is a read-only
property



Convenção

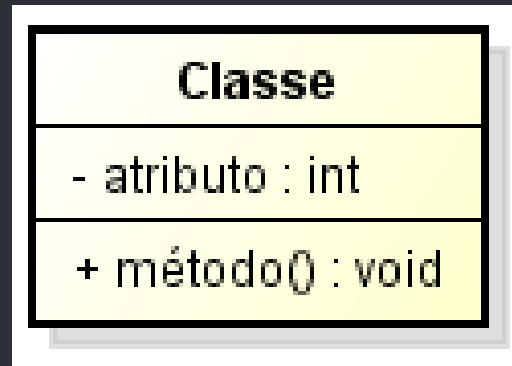
- É muito comum que atributos sejam private, e alguns métodos sejam public;
- Toda conversa de um objeto com outro é feita por troca de mensagens, isto é, acessando seus métodos;
- Entretanto, métodos usados apenas como auxiliares em uma classe devem ser privados.

Representação UML

- Em UML, a visibilidade é definida pelos caracteres

abaixo:

+ Publica
protegida
- privada



Vantagens

- Diminuindo a visibilidade de uma classe, ocultamos detalhes de implementação;
- Facilitam-se alterações na aplicação, pois uma regra só precisa ser modificada em um único lugar;
- Facilita o aprendizado, pois o mínimo de funcionalidades é exposto.

Consequência direta

- Diminui-se o acoplamento;
- Tornamos nosso código mais utilizável;
- Código mais fácil de manter.



Programação Orientada a Objetos

Encapsulamento

Ely – ely.miranda@ifpi.edu.br