



C125 – Programação Orientada a Objetos com
Java

Modificadores de Acesso

Prof. Phyllipe Lima
phyllipe@inatel.br

1



Agenda



- ☕ Controlar o acesso aos Métodos e Membros das classes por meio dos modificadores de acesso **public** e **private**
- ☕ Escrever métodos de acesso aos membros (**getters** e **setters**)

2

Controlando o Acesso



A classe **Conta** do início do curso, tinha um método para sacar dinheiro!

```
public class Conta {  
  
    int numero;  
    float saldo;  
    float limite;  
    Cliente titular;  
  
    void saca(float quantia) {  
        this.saldo -= quantia;  
    }  
}
```

C125 – PROGRAMAÇÃO ORIENTADA A OBJETOS COM JAVA – INSTITUTO NACIONAL DE TELECOMUNICAÇÕES

3

3

Controlando o Acesso



- ☕ O que aconteceria se tentássemos sacar uma quantia maior que o saldo?
- ☕ Conseguimos acessar os membros diretamente?
- ☕ É possível sacar uma quantia maior?
- ☕ Se sim, como resolver?

C125 – PROGRAMAÇÃO ORIENTADA A OBJETOS COM JAVA – INSTITUTO NACIONAL DE TELECOMUNICAÇÕES

4

4

Controlando o Acesso



A primeira ideia seria incluir um teste de condição dentro do método **sacar()**

```
void sacar(float quantia) {  
    if(this.saldo > quantia) {  
        this.saldo -= quantia;  
    }  
}
```

Controlando o Acesso



Isso resolve o problema?

E se tentarmos acessar o membro **saldo** diretamente?

```
Conta conta = new Conta();  
//Escrevendo  
conta.saldo = -1000;  
  
//Lendo  
System.out.println(conta.saldo);
```

Controlando o Acesso



- ☕ A situação anterior pode trazer muitos problemas!
- ☕ A melhor forma de resolver essa situação é garantir que a única opção para acessar o **saldo** seja através do método **saca()**
- ☕ Vamos proteger os membros da classe!
- ☕ Para isso vamos utilizar o modificador de acesso **private**

C125 – PROGRAMAÇÃO ORIENTADA A OBJETOS COM JAVA – INSTITUTO NACIONAL DE TELECOMUNICAÇÕES

7

7

Controlando o Acesso



```
private int numero;  
private float saldo;  
private float limite;  
private Cliente titular;  
  
void saca(float quantia) {  
    if(this.saldo > quantia) {  
        this.saldo -= quantia;  
    }  
}
```

C125 – PROGRAMAÇÃO ORIENTADA A OBJETOS COM JAVA – INSTITUTO NACIONAL DE TELECOMUNICAÇÕES

8

8

Controlando o Acesso



☕ Com essa modificação, os membros da classe agora só podem ser acessados de dentro da própria classe.

☕ Tente modificar os valores fora da classe. O que ocorre?

```
Conta conta = new Conta();  
  
conta.saldo = -1000; //Não compila  
conta.saldo = 350; //Continua não compilando
```

Controlando o Acesso



☕ Em Orientação a Objetos, é uma prática comum, talvez obrigatória, declarar todos os membros da classe com o modificador de acesso **private**

☕ A ideia por trás é encapsular o estado do objeto, e deixar que apenas o próprio objeto (através de métodos) modifique seu estado.

Controlando o Acesso



- ☕ O Encapsulamento permite esconder detalhes da implementação.
- ☕ Quem chama o método **sacar()** sabe que existe um teste para checar o saldo?
- ☕ É exatamente isso que queremos.

Controlando o Acesso



```
void saca(float quantia) {  
    if(this.saldo > quantia) {  
        this.saldo -= quantia;  
        System.out.println("Saque realizado");  
    }else  
        System.out.println("Saldo insuficiente");  
}
```

Controlando o Acesso



- ☕ O modificador de acesso **private** também pode ser utilizado em métodos.
- ☕ Normalmente é utilizado como método de apoio interno a própria classe.
- ☕ Apenas a própria classe tem acesso a esse método

Controlando o Acesso



```
private boolean verificaSerasa() {  
    //Faz verificacao no serasa  
}  
  
public void pedirEmprestimo(float quantia) {  
    if(verificaSerasa()) {  
        //De sequencia ao empréstimo  
    }  
}
```

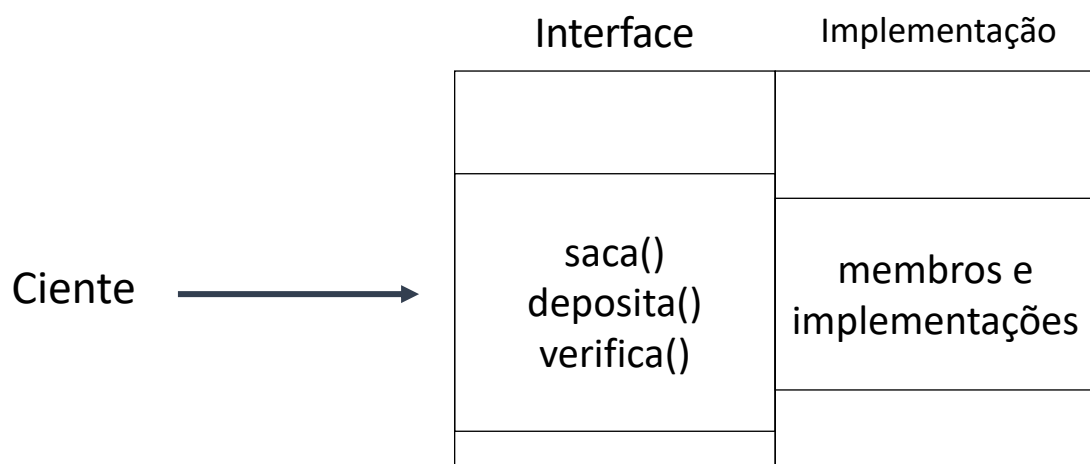


Encapsulamento de Dados

- ☕ O que vimos até esse momento é o conceito de encapsulamento de dados.
 - ☕ Esconder o acesso aos membros
 - ☕ Esconder implementação dos métodos
- ☕ O conjunto de métodos públicos é conhecido como “interface da classe”, pois é a única forma de trocar mensagens com ela.



Encapsulamento de Dados



Getters e Setters



- ☕ O modificador ***private*** torna os membros da classe acessível somente a própria classe
- ☕ Para prover uma forma controlada e segura de acesso aos membros da classe, utilizaremos os ***getters*** e ***setters***
- ☕ Eles fazem parte de uma convenção da linguagem Java, iniciando com ***get*** ou ***set*** e o nome do membro

Getters e Setters



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

Getters e Setters



 Atalho do Eclipse para gerar **Getters e Setters**

C125 – PROGRAMAÇÃO ORIENTADA A OBJETOS COM JAVA – INSTITUTO NACIONAL DE TELECOMUNICAÇÕES

19

19

Getters e Setters



private int numero:

private

private

private

void

Undo Typing	Ctrl+Z	Toggle Comment	Ctrl+/ Ctrl+Shift+\ Alt+Shift+J
Revert File		Remove Block Comment	
Save	Ctrl+S	Generate Element Comment	
Open Declaration	F3	Correct Indentation	Ctrl+I
Open Type Hierarchy	F4	Format	Ctrl+Shift+F
Open Call Hierarchy	Ctrl+Alt+H	Format Element	
Show in Breadcrumb	Alt+Shift+B	Add Import	Ctrl+Shift+M
Quick Outline	Ctrl+O	Organize Imports	Ctrl+Shift+O
Quick Type Hierarchy	Ctrl+T	Sort Members...	
Open With	>	Clean Up...	
Show In	Alt+Shift+W >	Override/Implement Methods...	
Cut	Ctrl+X	Generate Getters and Setters...	
Copy	Ctrl+C	Generate Delegate Methods...	
Copy Qualified Name		Generate hashCode() and equals()...	
Paste	Ctrl+V	Generate toString()...	
Quick Fix	Ctrl+1	Generate Constructor using Fields...	
Source	Alt+Shift+S >	Generate Constructors from Superclass...	
Refactor	Alt+Shift+T >	Externalize Strings...	
Local History	>		
References	>		

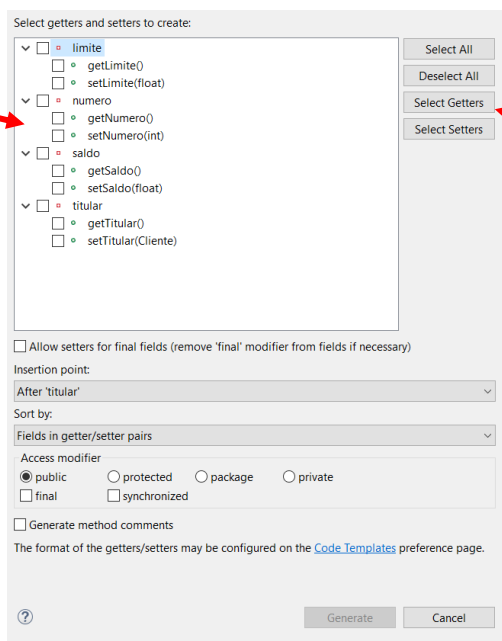
"Saldo insuficiente");

C125 – PROGRAMAÇÃO ORIENTADA A OBJETOS COM JAVA – INSTITUTO NACIONAL DE TELECOMUNICAÇÕES

20

20

Escolha os **Getters** e **Setters** que deseja



Opcionalmente, você pode escolher todos



C125 – PROGRAMAÇÃO ORIENTADA A OBJETOS COM JAVA – INSTITUTO NACIONAL DE TELECOMUNICAÇÕES

21

21

PARA! PARA!
PARA! PARA! PARA!
PARA! PARA!



☕ Crie **Getters** e **Setters** apenas se realmente for necessário. Você pode criar apenas **Getters** ou **Setters** em variáveis que necessitem acesso externo. No exemplo da classe **Conta**, faz sentido o **Setter** para saldo?

C125 – PROGRAMAÇÃO ORIENTADA A OBJETOS COM JAVA – INSTITUTO NACIONAL DE TELECOMUNICAÇÕES

22

22



Exercício 1 – Circuit Breaker



☕ Com a crise econômica que assola o mundo, não resta alternativa, precisamos de *software* mais robusto e seguro. Os banco solicitaram que adequemos as classes *Conta* e *Cliente* para atender a demanda de encapsulamento de dados.

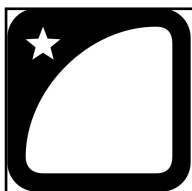
☕ Considere que a *Conta* pode possuir mais de um *Cliente*

Resolução dos Exercícios




<https://github.com/phillima-inatel/C125>





Material Complementar



 Capítulo 5 da apostila FJ-11

 Modificadores de Acesso

 Até o item 5.3

C125 – PROGRAMAÇÃO ORIENTADA A OBJETOS COM JAVA – INSTITUTO NACIONAL DE TELECOMUNICAÇÕES

25

25

Inatel

C125 – Programação Orientada a Objetos com
Java

Modificadores de Acesso

Prof. Phyllipe Lima

phyllipe@inatel.br



26