Inatel

C06/C206 – Programação Orientada a Objetos com Java

Modificadores de Acesso

Prof. Christopher Lima

christopher@inatel.br

Objetivos

- 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***)

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;
    }
```

Controlando o Acesso

- O que aconteceria se tentássemos sacar uma quantia maior que o saldo?
- Conseguimos acessar os membros diretamente?
- SIM, como resolver?

Controlando o Acesso

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

```
void saca(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***

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;
    }

}
```

Controlando o Acesso

- Com essa modificação, os membro 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 nao 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 a classe (através de métodos) modique 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

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 um convenção da linguagem Java, iniciando com ***get*** ou ***set*** e o nome do membro

Getters e Setters

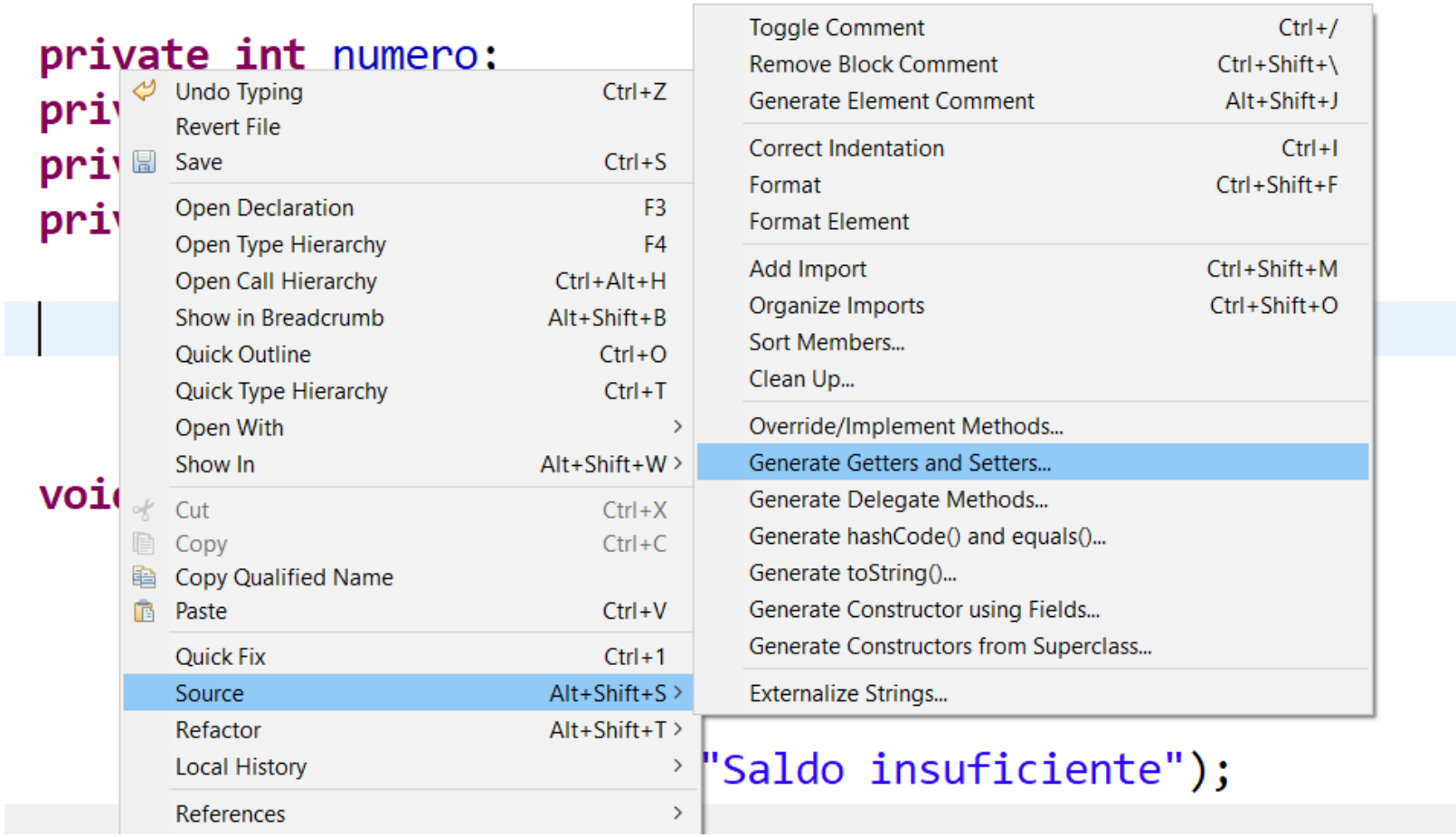
```java
public float getSaldo() {
    return this.saldo;
}

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

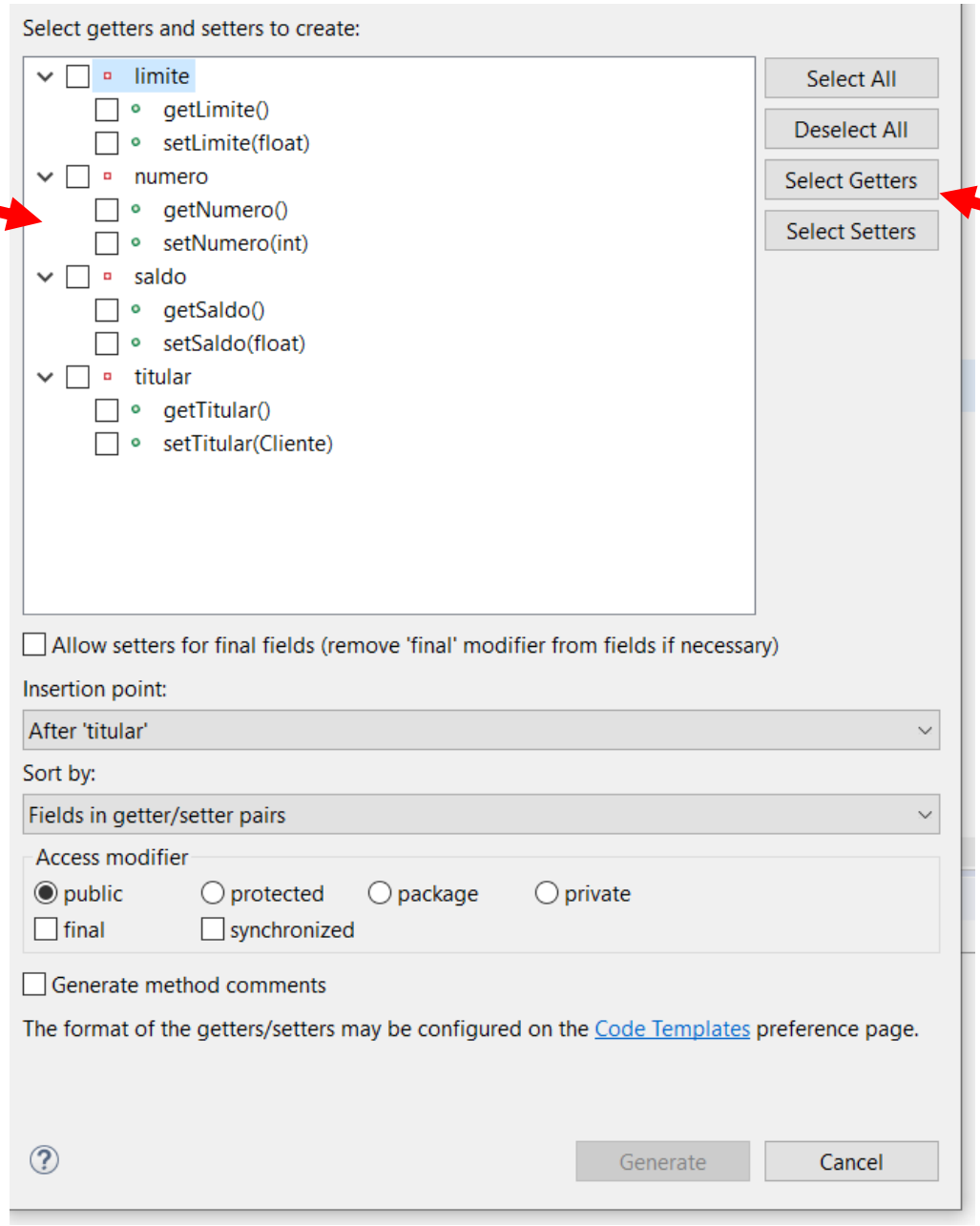
Getters e Setters

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

Getters e Setters



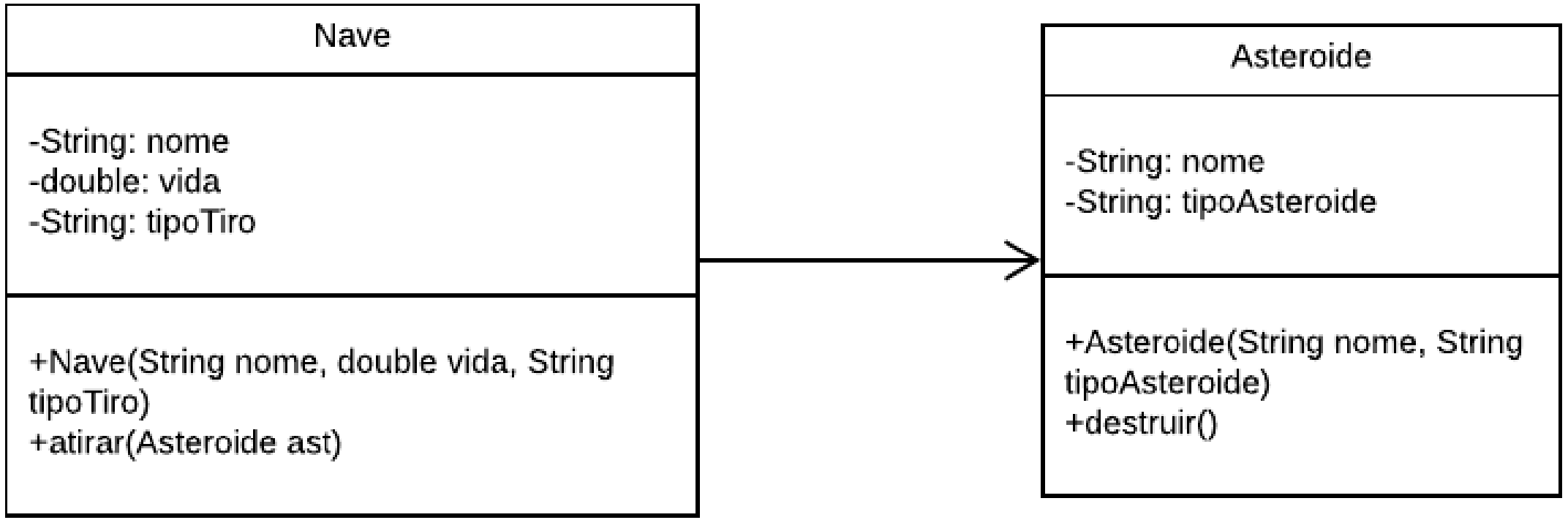
Escolha os ***Getters*** e ***Setters*** que deseja



Opcionalmente, você pode escolher todos

- 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?

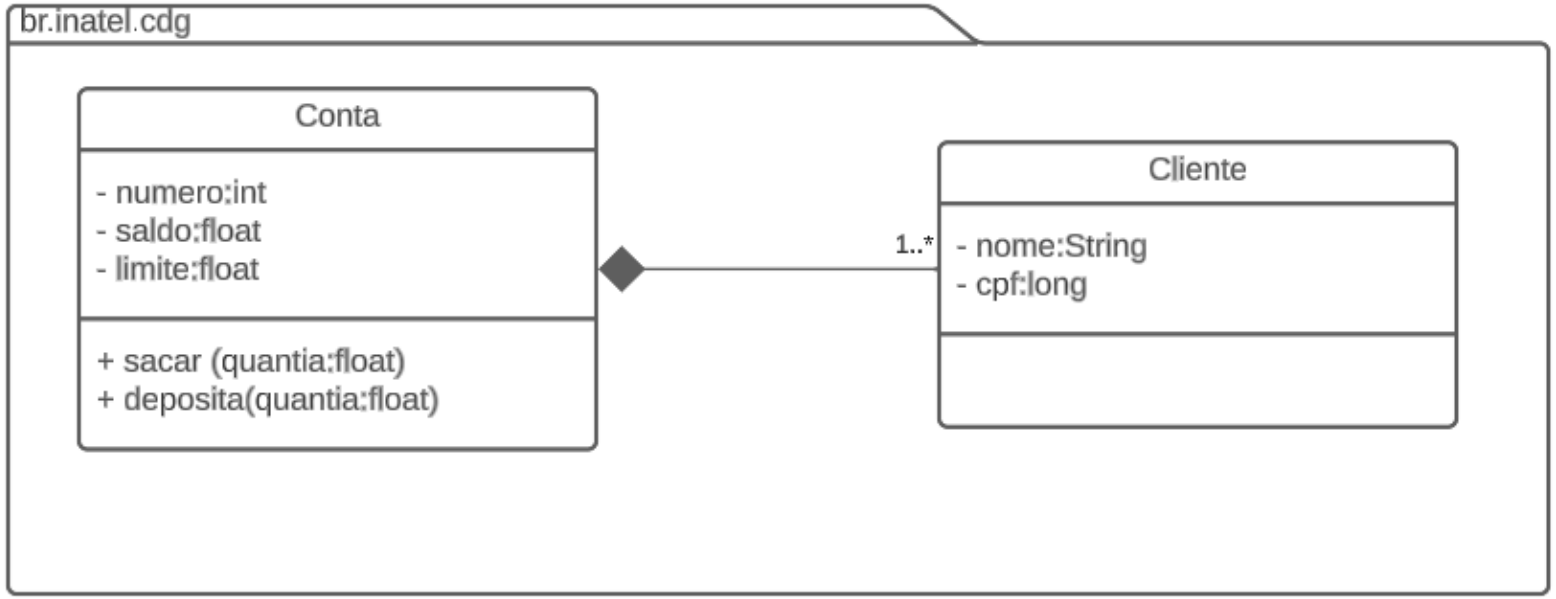
UML e modificadores de acesso



+ public
- private

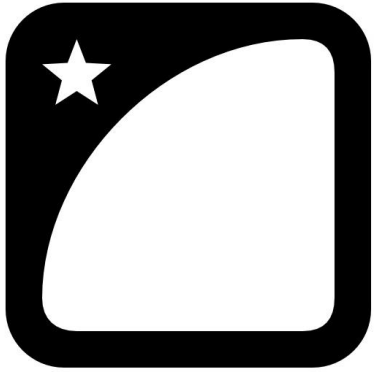
Exercício 1 – Conta e Cliente (novamente)

- Modele o seguinte UML.
- Faça as verificações necessárias no método sacar().
- Crie *getters* e *setters* quando necessário.
- Crie a classe Main e utilize as classes Conta e Cliente.



Resolução dos Exercícios

https://github.com/chrislima-inatel/C206_C125



Material Complementar

- Capítulo 5 da apostila FJ-11
  - Modificadores de Acesso
  - Até o item 5.3