



**INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA**
RIO GRANDE DO SUL
Câmpus Feliz

Herança

Prof. Moser Fagundes

Programação II

Sumário

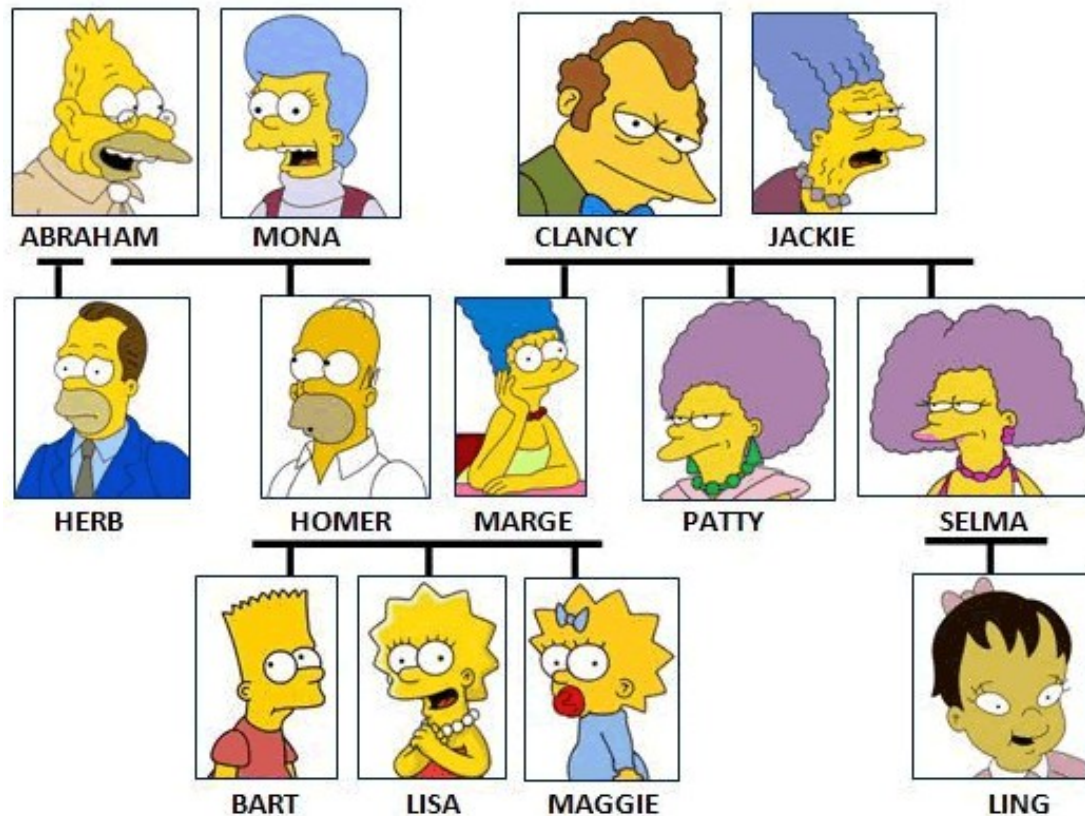
- Herança
- Encapsulamento (no contexto da herança)
- Exercícios

Vantagens da orientação a objetos

- A utilização da orientação a objetos e do encapsulamento do código em classes nos permite uma maior **organização**, mas um dos maiores benefícios que encontramos no paradigma orientado a objetos é o **reuso**.
- A possibilidade de **reusar** partes de código já programadas é o que nos dá agilidade no dia-a-dia, além de **eliminar duplicações e reescritas de código**.

Conceito de herança

- Quando falamos em **herança**, frequentemente pensamos em um tipo de *árvore genealógica* com avós, pais e filhos, e nas *características transmitidas* de uma geração para a próxima.



Conceito de herança

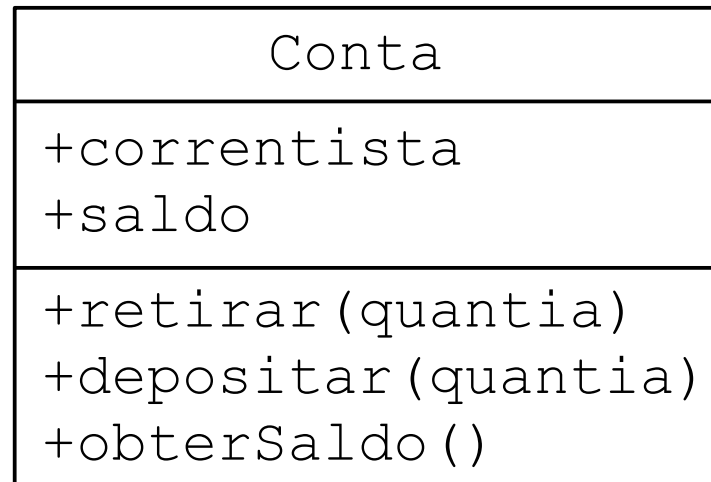
- Na orientação a objetos, vemos a herança como a **passagem** (*transmissão*) de **características** (*atributos*) e **comportamentos** (*métodos*) entre classes de uma mesma **hierarquia** (*árvore*).
- As classes inferiores na hierarquia (*subclasses*) **automaticamente herdam** todos os **atributos** e **métodos** das classes superiores (*superclasses*).

Vantagens da herança

A **herança** tem aplicabilidade muito grande, visto que é muito comum termos de adicionar novas funcionalidades nos programas que estamos desenvolvendo:

Em vez de criarmos uma estrutura totalmente nova (uma nova classe), podemos reaproveitar uma estrutura (classe já existente) que nos forneça uma base provendo atributos e métodos básicos.

Classes em UML



Classes em Java

```
package banco;
```

```
public class Conta {
```

```
    public String correntista;
```

```
    public double saldo;
```

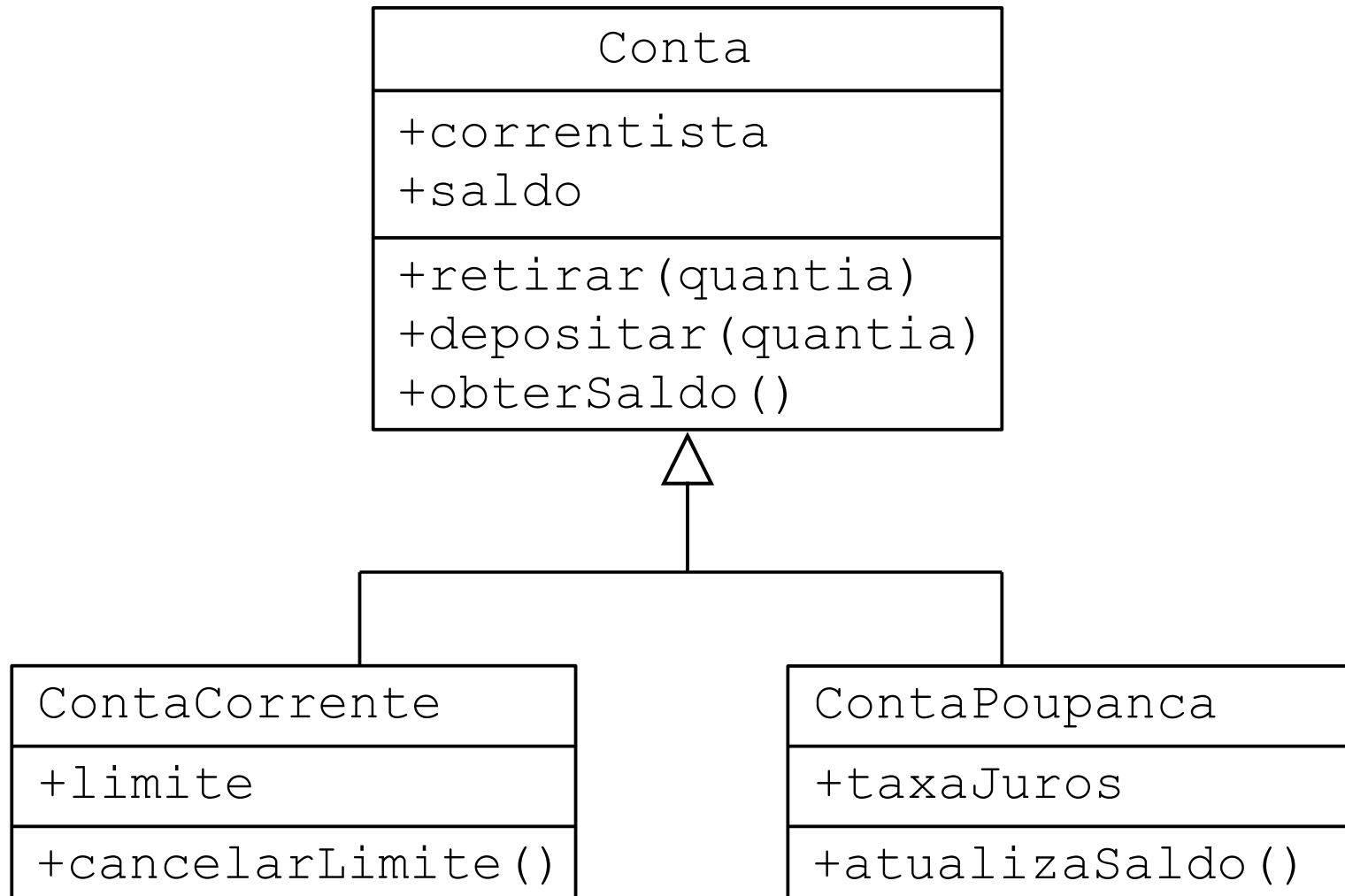
```
    public void retirar(double quantia) {  
        saldo = saldo - quantia;  
    }
```

```
    public void depositar(double quantia) {  
        saldo = saldo + quantia;  
    }
```

```
    public double obterSaldo() {  
        return saldo;  
    }
```

```
}
```


Herança de classes em UML



Herança de classes em Java

// Neste exemplo, a classe ContaCorrente herda
// da classe Conta. Ou seja, Conta é a superclasse,
// enquanto ContaCorrente é a subclasse.

```
package banco;
```

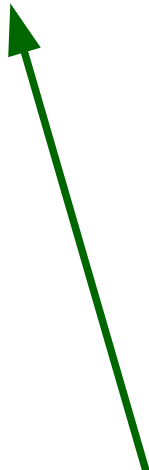
```
public class ContaCorrente extends Conta {
```

```
    public double limite;
```

```
    public void cancelarLimite() {  
        limite = 0;
```

```
    }
```

```
}
```



O **extends** determina que ContaCorrente é uma subclasse de Conta

Herança de classes em Java

// Neste exemplo, a classe ContaPoupanca também herda
// da classe Conta. Note que o método atualiza saldo
// usa o atributo saldo que foi herdado de Conta.

```
package banco;
```

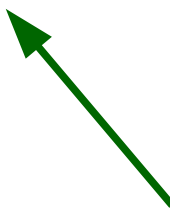
```
public class ContaPoupanca extends Conta {
```

```
    public double taxaJuros;
```

```
    public void atualizaSaldo() {  
        saldo = saldo * taxaJuros;
```

```
    }
```

```
}
```



Posso usar atributos públicos que foram herdados da superclasse!

Encapsulamento

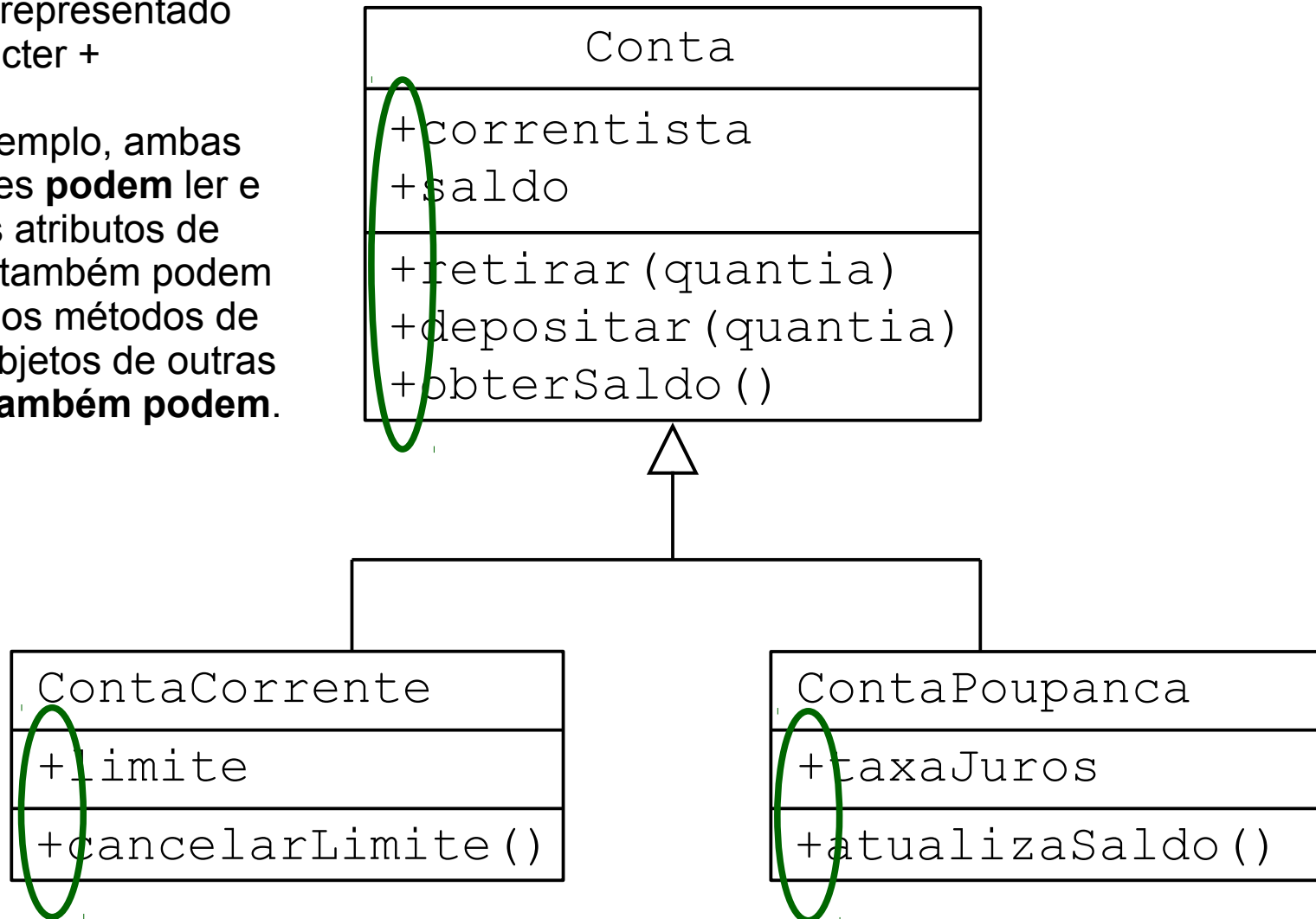
O **encapsulamento**, um dos recursos mais importantes da orientação a objetos, provê **proteção de acesso** aos atributos e métodos internos de um objeto.

Visibilidade	Descrição
public	Membros declarados como <u>public</u> poderão ser acessados livremente a partir da própria classe, a partir das classes descendentes e a partir do programa que faz uso da classe.
protected	Membros <u>protected</u> poderão ser acessados a partir da própria classe, das classes descendentes e classes no mesmo pacote.
private	Membros declarados como <u>private</u> poderão ser acessados apenas da própria classe. Não podem ser acessados de classes descendentes, nem do programa que usa a classe.

public em UML

public é representado pelo caracter +

Neste exemplo, ambas subclasses **podem** ler e alterar os atributos de Conta, e também podem executar os métodos de Conta. Objetos de outras classes **também podem**.



public na subclasse

```
// O atributo saldo foi declarado como public na  
// classe Conta. Logo, podemos usar ele na subclasse  
// ContaPoupanca sem restrições!
```

```
package banco;
```

```
public class ContaPoupanca extends Conta {
```

```
    public double taxaJuros;
```

```
    public void atualizaSaldo() {  
        saldo = saldo * taxaJuros;
```

```
    }
```

```
}
```



Nesse caso, `saldo` foi declarado `public` na superclasse!

Acesso externo com **public**

// Acesso externo OK

```
package banco;
```

```
public class Exemplo {  
    public static void main(String[] args) {  
        ContaPoupanca cp = new ContaPoupanca();  
        cp.correntista = "Bob Esponja";  
        cp.saldo = 1000.0;  
        cp.taxaJuros = 1.05;  
        cp.atualizaSaldo();  
    }  
}
```



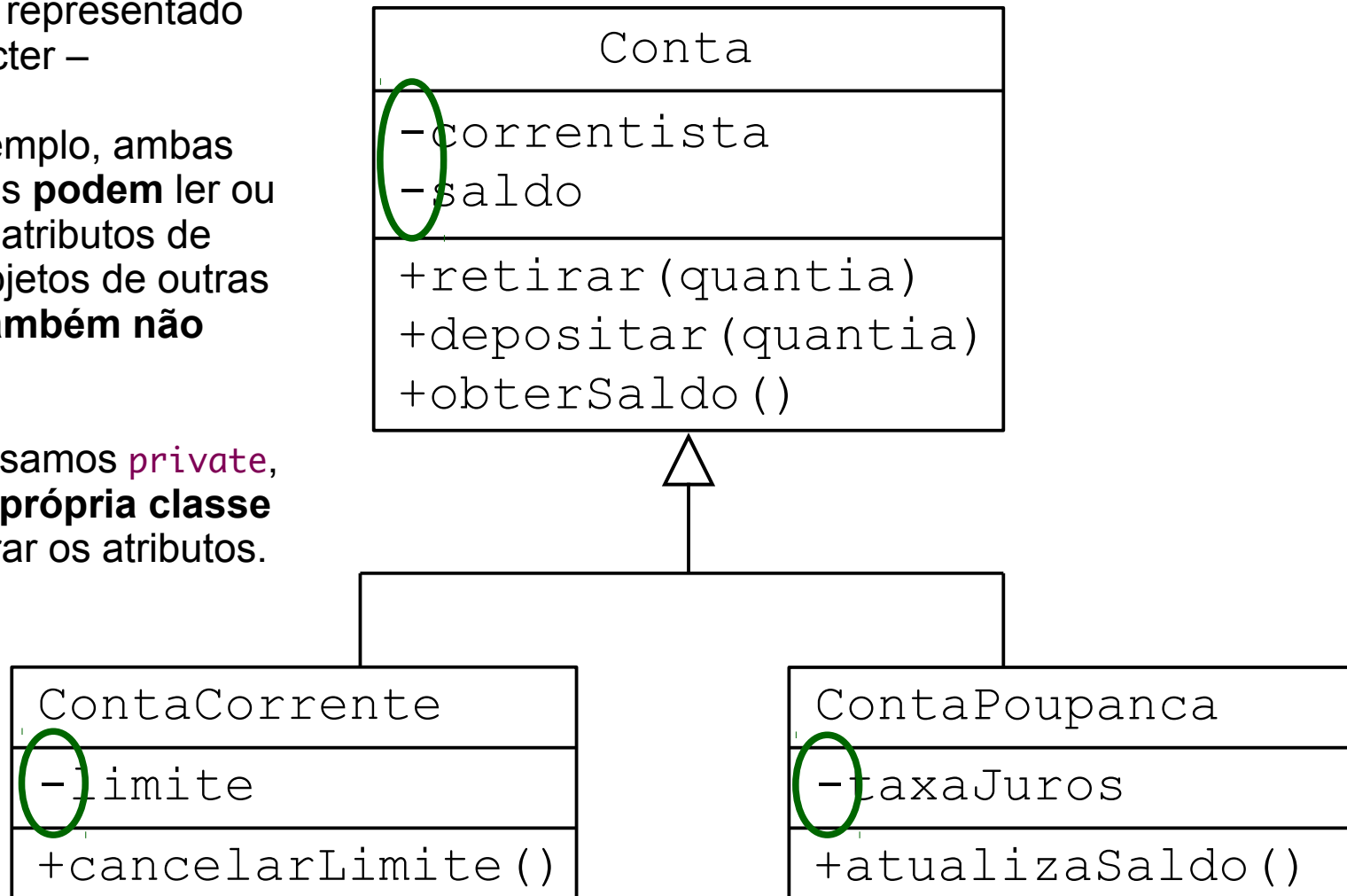
Atributos e métodos **public** são acessados externamente sem problemas.

private em UML

private é representado pelo caracter –

Neste exemplo, ambas subclasses **podem** ler ou alterar os atributos de Conta. Objetos de outras classes **também não podem**.

Quando usamos **private**, apenas a **própria classe** pode alterar os atributos.



Classe Conta com atributos **private**

```
package banco;
```

```
public class Conta {
```

```
    private String correntista;  
    private double saldo;
```

```
    public void retirar(double quantia) {  
        saldo = saldo - quantia;  
    }
```

```
    public void depositar(double quantia) {  
        saldo = saldo + quantia;  
    }
```

```
    public double obterSaldo() {  
        return saldo;  
    }
```

```
}
```

Membros **private**
são acessíveis na
própria classe.



ContaPoupanca com atributos **private**

// Neste exemplo, a classe ContaPoupanca também tem
// atributo private. O método atualiza pode acessar
// taxaJuros, mas não pode acessar o atributo saldo
// herdado de Conta pois saldo é private.

```
package banco;
```

```
public class ContaPoupanca extends Conta {
```

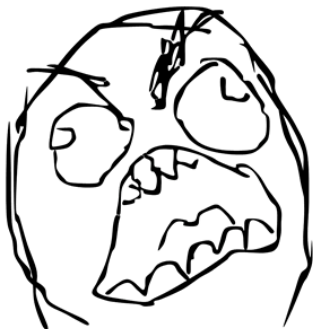
```
    private double taxaJuros;
```

```
    private void atualizaSaldo() {
```

```
        saldo = saldo * taxaJuros;
```

```
    }
```

```
}
```



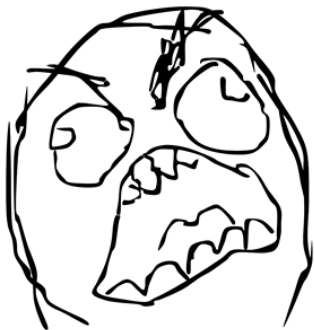
Essa linha vai apresentar um **ERRO**,
pois atributos **private** **não** são
acessíveis nas subclasses!

Acesso externo com **private**

// Acesso externo negado!

```
package banco;
```

```
public class Exemplo {  
    public static void main(String[] args) {  
        ContaPoupanca cp = new ContaPoupanca();  
        cp.correntista = "Bob Esponja";  
        cp.saldo = 1000.0;  
        cp.taxaJuros = 1.05;  
        cp.atualizaSaldo();  
    }  
}
```



Atributos e métodos **private** não são acessíveis externamente.

Exercício

Lista de exercícios no Moodle.