

Encapsulamento

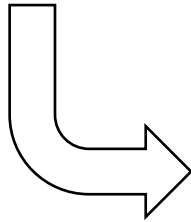
Encapsulamento - Motivação

Considerar a classe abaixo, a ser utilizada para representar contas bancárias:

ContaBancaria
titular : String saldo : double
depositar(valor : double) : void sacar(valor : double) : void

Encapsulamento – Motivação

ContaBancaria
titular : String saldo : double
depositar(valor : double) : void sacar(valor : double) : void



```
1 public class ContaBancaria {  
2  
3     String titular;  
4     double saldo;  
5  
6     void depositar(double valor) {  
7         saldo = saldo + valor;  
8     }  
9  
10    void sacar(double valor) {  
11        saldo = saldo - valor;  
12    }  
13  
14 }
```

Encapsulamento - Motivação

```
1 public class CaixaEletronico {  
2  
3     public static void main(String[] args) {  
4  
5         ContaBancaria conta1 = new ContaBancaria();  
6         conta1.titular = "Sandro da Silva";  
7         conta1.depositar(500);  
8         conta1.sacar(100);  
9         System.out.println(conta1.saldo);  
10  
11     }  
12  
13 }
```

Mostra 400

Encapsulamento - Motivação

```
1 public class CaixaEletronico {  
2  
3     public static void main(String[] args) {  
4  
5         ContaBancaria conta1 = new ContaBancaria();  
6         conta1.titular = "Sandro da Silva";  
7         conta1.depositar(500);  
8         conta1.sacar(100);  
9         conta1.saldo = 10000;  
10  
11     }  
12  
13 }
```

Este comando permite que
seja definido um valor de
saldo, sem que seja feito um
depósito correspondente

Encapsulamento

- O acesso ao atributo deve ser “controlado”, para garantir integridade dos dados
 - Isto é, o estado do objeto precisa ser “controlado”
- Somente o próprio objeto deveria manipular o valor de seus atributos
- Esta técnica se chama **encapsulamento** de dados
- Em Java, para aplicar o encapsulamento é preciso tornar o atributo **privado**.

Encapsulamento

Em UML, para expressar que um atributo é encapsulado, será apresentado um sinal de “-” na frente do atributo, como no exemplo:

ContaBancaria
- titular : String - saldo : double
depositar(valor : double) : void sacar(valor : double) : void

Encapsulamento

Outros símbolos podem ser utilizados nos membros de uma classe (atributos e operações) para indicar seu grau de visibilidade:

ContaBancaria - titular : String - saldo : double depositar(valor : double) : void sacar(valor : double) : void	Símbolo UML	Nome	Palavra reservada*	Significado
	-	Privado	private	Somente visível pela própria classe
	+	Público	public	Visível para qualquer classe
	#	Protegido	protected	Estudaremos mais tarde
	~	“de pacote”	(ausência de símbolo)	Estudaremos mais tarde

* Conhecido também como “modificador de acesso”

Sintaxe para traduzir um atributo em Java:

```
Modificador de acesso tipo de dado identificador;
```

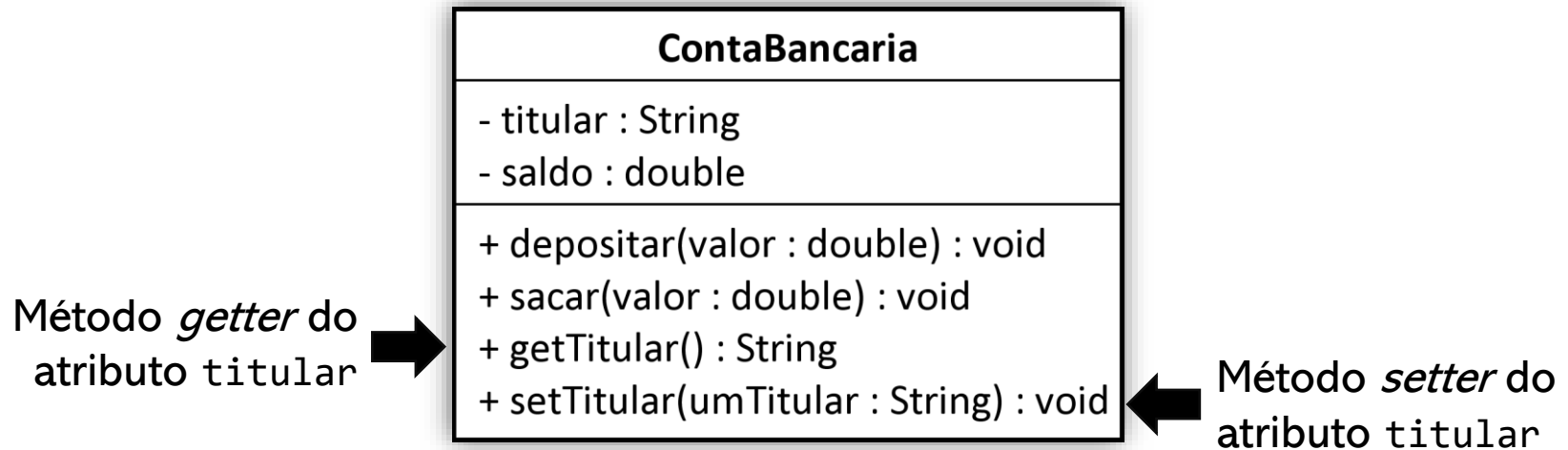
Exemplo:

```
private double saldo;
```


Encapsulamento – Métodos de acesso

- Todos os atributos de um objeto deveriam ser encapsulados.
- Os atributos que precisam ser acessados por outras classes poderão ser acessados por meio de **métodos de acesso**.
- Os métodos de acesso são métodos, geralmente públicos. Dividem-se em:
 - **Getters**: métodos usados para recuperar valor de atributos de um objeto da classe. O nome de um *método getter* deve ser escrito com o prefixo *get*, seguido do nome do atributo com a inicial maiúscula.
Exceção: se o atributo for lógico (booleano), usar o prefixo “is”.
 - **Setters**: métodos usados para atribuir valor de atributos da classe. O nome de um *método setter* deve ser escrito com o prefixo *set*, seguido do nome do atributo com a letra inicial maiúscula.

Exemplo



Métodos *getter* :

- nunca têm parâmetro
- sempre são do tipo função
- sempre retornam um dado cujo tipo é igual ao da variável em que é *getter*

Métodos *setter* :

- sempre têm um parâmetro
- sempre são do tipo procedimento
- o parâmetro deve ser declarado com um tipo de dado igual ao da variável que é *setter*

Exemplo de *getter* e *setter*

```
public class ContaBancaria {  
  
    private String titular;  
    private double saldo;  
  
    void depositar(double valor) {  
        saldo = saldo + valor;  
    }  
  
    void sacar(double valor) {  
        saldo = saldo - valor;  
    }  
  
    public void setTitular(String novoTitular) {  
        titular = novoTitular;  
    }  
  
    public String getTitular() {  
        return titular;  
    }  
  
}
```

this

- **this** refere-se ao objeto corrente – o objeto na qual o método foi chamado.
- O principal motivo em usar a palavra **this** é quando há algum parâmetro de método que possui o mesmo nome de uma variável de instância.
 - Quando há dois identificadores com o mesmo nome, por padrão Java sempre reconhece que o identificador utilizado no comando é aquele com menor escopo

Exemplo de *getter* e *setter*

```
public class ContaBancaria {  
  
    private String titular;  
    private double saldo;  
  
    void depositar(double valor) {  
        saldo = saldo + valor;  
    }  
  
    void sacar(double valor) {  
        saldo = saldo - valor;  
    }  
  
    public void setTitular(String titular) {  
        this.titular = titular;  
    }  
  
    public String getTitular() {  
        return titular;  
    }  
  
}
```

Encapsulamento

- Sempre dar preferência por encapsular todos os atributos de uma classe
- Somente é admissível utilizar **public** para constantes
- Se for necessário expor o valor de atributo para outros objetos/classes, implementar um método *getter* para o atributo
- Se for necessário permitir que outros objetos/classes definam o valor de um atributo, é necessário implementar um método *setter* para o atributo

Encapsulamento de métodos

- Ao utilizar POO, é possível ocultar a complexidade do trabalho interno executado pelo objeto:
 - Criar uma forma simplificada e compreensível de utilizar o objeto – favorece a reutilização
 - Por exemplo: o motorista não precisa compreender como o mecanismo interno de combustão funciona para ligar o carro.