

Java Básico

Igor Ebrahim (ies@cin.ufpe.br)



Módulo 3

Programação Orientada a Objetos

+ Programação Orientada a Objetos

■ O que é POO?

- Baseada na representação de objetos do mundo real

■ Vantagens

- Facilidade de Manutenção
- Maior reuso

+ Objetos

- Um objeto possui:
 - Identidade: permite distingui-lo de outros
 - Estados: características
 - Comportamentos: o que pode ser feito com ele (ou nele)
- Por exemplo:
 - Os estados de uma conta bancária são o seu número e o seu saldo
 - Os comportamentos atribuídos a uma conta bancária são a habilidade de realizar depósitos e saques
- Note que o comportamento de um objeto pode modificar seus estados

+ Classes

- Um objeto é definido por uma classe
 - Características (atributos)
 - Comportamentos (métodos)
- Múltiplos objetos podem ser criados (instanciados) por uma única classe

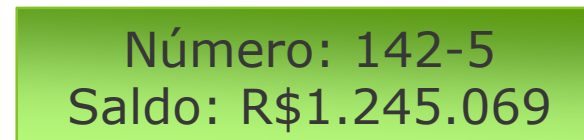
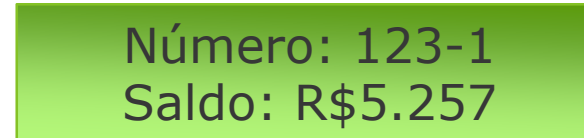
{
Classe → conceito
Objeto → materialização

+ Objetos x Classes

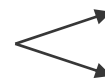
Uma classe
(conceito)



Um objeto
(materialização)



Múltiplos objetos definidos
por uma mesma classe



+ Atributos



- Determinam as características do objeto
- Os modificadores são opcionais
- Vários atributos podem ser declarados na mesma linha
- Um atributo pode ser inicializado na declaração

+ Exemplos

```
class Lapis {  
    String cor;  
}
```

```
class Cadeira {  
    int numeroPernas;  
    String fabricante;  
}
```

```
class Conta{  
    String numero;  
    double saldo;  
}
```


+ Métodos

```
public double calcular(int valor, double outroValor) { ... }
```

modificadores

tipo de retorno

nome

parâmetros

- Operações que realizam ações ou modificam o objeto responsável pela sua execução
- O corpo do método determina o comportamento
- E também pode conter declaração de variáveis
 - Cujas existências e valores são válidos somente dentro do método em que são declaradas.

+ Exemplo (return)

```
class Conta{
    String numero;
    double saldo;

    String getNumero() {
        return numero;
    }

    double getSaldo() {
        return saldo;
    }

    ...
}
```

+ Exemplo (void)

```
class Conta{
    String numero;
    double saldo;

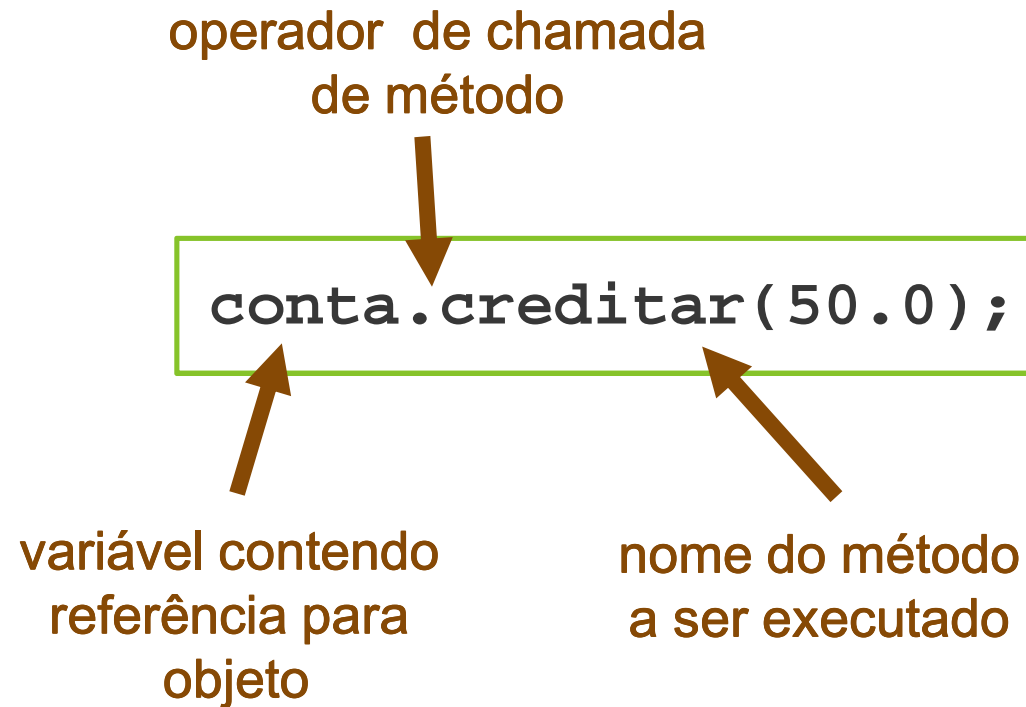
    void creditar(double valor) {
        saldo = saldo + valor;
    }

    ...
}
```

+ Métodos

- Por que no método creditar não temos o número da conta como parâmetro?
- Métodos são invocados por instâncias (objetos)
 - Também podem ser invocados pela classe (métodos estáticos)
- Parâmetros são passados por cópia

+ Chamada de Métodos



+ Instanciando Objetos

- Objetos precisam ser criados antes de serem utilizados
- A criação é feita com o operador **new**

```
Conta c = new Conta();
```

construtor



+ Construtores

- Constutores definem como os atributos do objeto vão ser inicializados
- São **semelhantes** a métodos, mas não têm tipo de retorno
- O nome do construtor deve ser **exatamente** o nome da classe
- Um classe pode ter diversos construtores, diferenciados pelos parâmetros

```
public Conta(String numero) {  
    this.numero = numero;  
}
```

+ Construtor default

- Caso não seja definido um construtor, um construtor default é fornecido implicitamente
- O construtor default inicializa os atributos com seus valores padrões
- O construtor default não tem parâmetros
- Quando um construtor é definido, o construtor default não é mais gerado

```
public Conta() {  
    ...  
}
```


+ Valores padrões para atributos

Tipo	Valor Padrão
byte, short, int, long	0
float	0.0f
double	0.0
char	'\u0000'
Tipos referência (String, arrays, objetos em geral)	null
boolean	false

+ Prática

1. Escreva a classe Conta. Ela deve possuir número, saldo e limite, e deve ser possível creditar, debitar e transferir valores.
 - Não deve ser possível debitar um valor maior que o saldo;
2. Instancie 3 contas diferentes e utilize pelo menos uma vez cada uma das operações implementadas na questão anterior.

+ Remoção de Objetos

- Em Java, não temos um método para remoção explícita de objetos da memória (como o `free()` de C++);
- Gargabe Collector (coletor de lixo) elimina objetos da memória quando eles não são mais referenciados;
- A JVM que decide a hora que será feita a coleta de lixo;

+ Modificadores

■ Acesso

- public
- protected
- private
- default

■ Outros

- static
- final
- native
- transient
- synchronized

+ Modificadores de Acesso

- São aplicados a:
 - Classes;
 - Atributos;
 - Métodos ;
 - Construtores;

- Não se aplicam a variáveis locais;

+ public

- Classe: pode ser instanciada por qualquer outra classe
- Atributos: podem ser acessados por objetos de qualquer classe
- Métodos: podem ser chamados por métodos de qualquer classe

```
public class Conta{  
    public String numero;  
    ...  
    public void debitar(double valor) {  
        saldo = saldo - valor;  
    }  
    ...  
}
```

+ protected

- Classe: não se aplica
- Atributos: podem ser acessados por objetos de classes dentro do mesmo pacote ou de qualquer subclasse da classe ao qual ele pertence
- Métodos: podem ser chamados por objetos de classes dentro do mesmo pacote ou de qualquer subclasse da classe ao qual ele pertence

```
public class Conta{  
    protected String numero;  
    ...  
    protected void debitar(double valor) {  
        saldo = saldo - valor;  
    }  
    ...  
}
```

+ default

- Classe: visível apenas por classes do mesmo pacote
- Atributos: podem ser acessados por objetos de classes dentro do mesmo pacote
- Métodos: podem ser chamados por objetos de classes dentro do mesmo pacote

```
class Conta{
    String numero;
    ...
    void debitar(double valor) {
        saldo = saldo - valor;
    }
    ...
}
```


+ private

- Atributos: podem ser acessados apenas por objetos da mesma classe
- Métodos: podem ser chamados por objetos da mesma classe

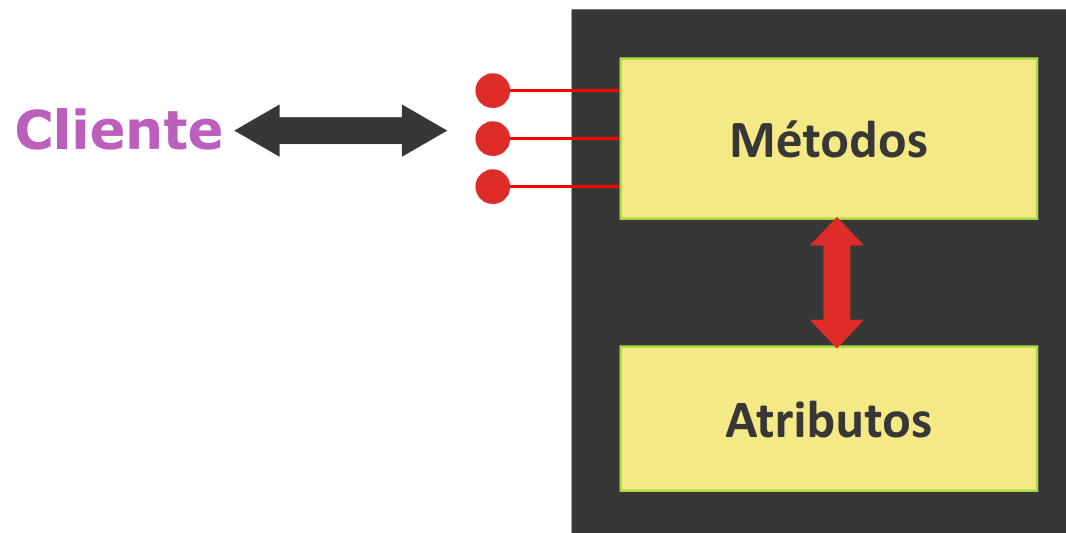
```
class Conta{
    private String numero;
    ...
    private void debitar(double valor) {
        saldo = saldo - valor;
    }
    ...
}
```

+ Encapsulamento

- Visões de objetos:
 - Interna: atributos e métodos da classe que o define;
 - Externa: os serviços que um objeto proporciona e como ele interage com o resto do sistema;
- Um objeto pode usar os serviços providos por outro – mas não precisa saber como estes são implementados;

+ Encapsulamento

- O uso de `private` nos atributos não é obrigatório, mas é recomendado para a programação orientada a objetos;
- Use `private` para atributos!



+ Prática

1. Refatore o código da classe Conta, utilizando os modificadores de acesso onde for possível.
2. Crie a classe Cliente. Ele deve possuir nome e cpf.
3. Refatore a classe Conta para que ela possua um Cliente.