

AULA 1 – REVISÃO ORIENTAÇÃO A OBJETOS

GS1020 – Programação Orientada a Objetos II

Prof. Dr. Murillo G. Carneiro
mgcarneiro@ufu.br



Material baseado nos slides cedidos pelo Prof. Rafael D. Araújo (FACOM/UFU)

O que é Orientação a Objetos (OO)?

- Orientação a Objetos é um paradigma para o desenvolvimento de software baseado na utilização de **componentes individuais** (objetos) que **colaboram** entre si para construir os sistemas computacionais

Orientação a objetos é necessária?

- Nem sempre!

- Há situações onde o modelo de uma tarefa a ser executada é tão simples que a criação de uma classe para representá-lo torna o problema mais complicado

Conceitos fundamentais e terminologia

- Abstração
- Classes e Objetos
- Encapsulamento
- Herança
- Interfaces
- Polimorfismo

Abstração

- Abstração é a capacidade que temos em abstrair o mundo real em objetos. Ou seja, criar representações (por meio de objetos) do mundo real.
- Na programação orientada a objetos (POO), as abstrações são feitas por meio da criação de classes.

Classe

- Molde (formato, estrutura) para criação de objetos
- Na prática:
 - *É a unidade básica de trabalho em um programa orientado a objetos*
 - *Define a estrutura de um tipo de dados (atributos e métodos)*

Classe

- **Propriedades (atributos):** representa as características do objeto. Por exemplo, cor, nome, tamanho, etc.
- **Ações (métodos):** capacidade de seu objeto executar ações que podem ou não modificar suas características. Por exemplo, ligar, desligar, caminhar, correr, etc.

Exemplo - Classe: Conta Bancária

■ Quais seriam os atributos?

■ Quais seriam os métodos?

Classes x objetos

- Uma classe representa um modelo abstrato a partir do qual são criados os objetos (instâncias)
- Um objeto é uma instância de uma classe
- A classe é a descrição de um tipo de objeto
 - *Por exemplo, considere uma classe Carro. Um objeto que representa um Corsa Branco, ano 2017, é um tipo de Carro.*

Encapsulamento

- Consiste na separação dos aspectos internos e externos de um objeto
- É utilizado para impedir o acesso direto ao estado de um objeto (seus atributos) ... disponibilizando apenas métodos para validar e efetivar suas alterações

Encapsulamento

- Permite ignorar os detalhes de implementação (de como as coisas funcionam internamente) permitindo ao desenvolvedor idealizar seu trabalho em um nível mais alto de abstração

Encapsulamento

Exemplo:

- Ninguém precisa conhecer detalhes dos circuitos de um telefone para utilizá-lo
- Sua carcaça encapsula os detalhes e nos provê uma interface amigável ... botões, monofone, sinais de tom ...

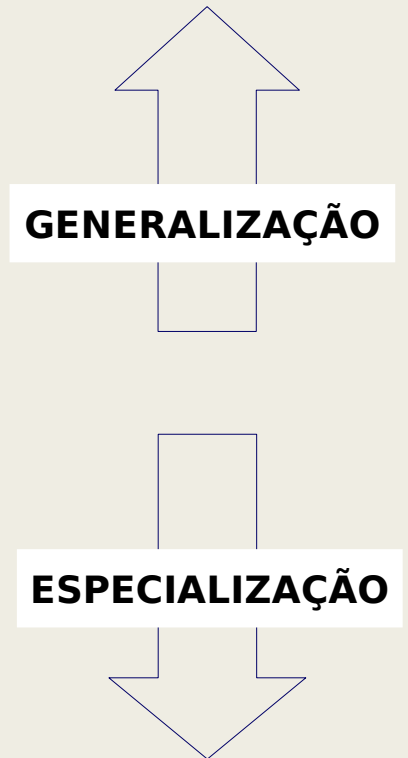
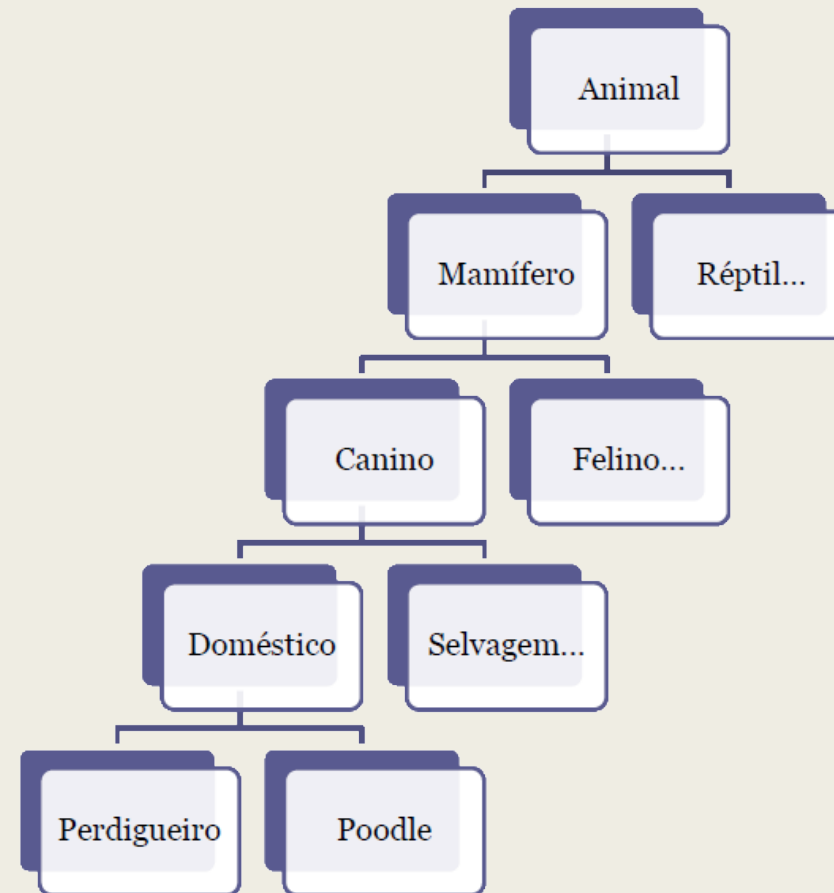


Herança

- É o mecanismo pelo qual uma classe pode estender outra aproveitando seus comportamentos (métodos) e estados possíveis (atributos)
- Permite que elementos mais específicos incorporem a estrutura e o comportamento de elementos mais genéricos
- Reuso e reaproveitamento de código

Herança

- Quando um objeto da classe filho é criado ele herda todas as propriedades da classe pai, além das propriedades definidas na própria classe filho



Herança

■ Um objeto de uma subclasse (classe filha) **é um** **tipo de** objeto da superclasse (classe pai)



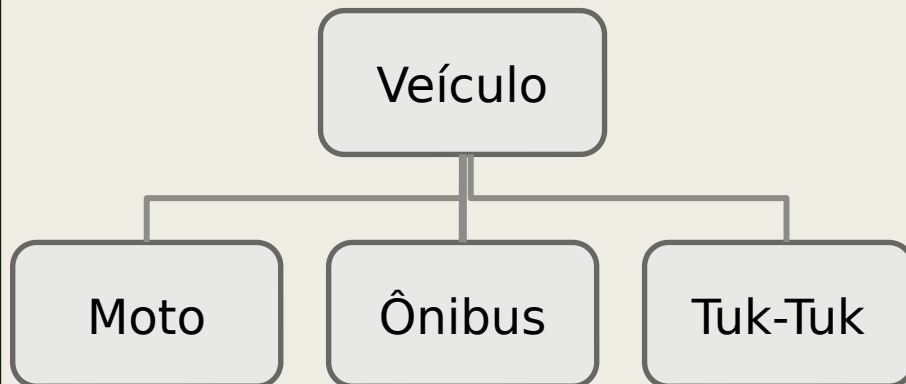
Moto é um tipo de veículo



Ônibus é um tipo de veículo



Tuk-Tuk é um tipo de veículo



Herança

- Se um objeto de uma subclasse **não** possui todos os atributos e operações da superclasse, ela (classe deste objeto) **não pode ser** uma subclasse



Gato não é um tipo de coelho!

Interfaces

- Significa que uma classe herda apenas a assinatura (definição) dos métodos sem implementação
- Dizemos que uma classe implementa uma ou mais interfaces

Herança – Vantagens

- Reaproveitamento de código
- Captura o que é comum e isola o que é diferente
- Organização hierárquica

Herança – Desvantagens

- Forte acoplamento entre as classes: uma alteração na superclasse, afeta diretamente as subclasses
- É um relacionamento estático, ou seja, não pode ser alterado em tempo de execução. Exemplo: Passageiro e Agente

Polimorfismo

- Significa “várias formas”
- Permite que uma mesma operação possa ser definida para diferentes tipos de classes, e cada uma delas a implementa como quiser

Polimorfismo

- Permite a um mesmo objeto se manifestar de diferentes formas
- A decisão sobre qual método deve ser selecionado é tomada em tempo de execução, de acordo com o tipo do **objeto instanciado**

Polimorfismo

Na prática:

- Sobrescrita de métodos (ou polimorfismo de inclusão): quando um método sobrescreve um método herdado de uma classe. A assinatura do método deve ser idêntica (mesmo nome, valor de retorno e argumentos).
- Sobrecarga de métodos: métodos com mesmo nome porém assinaturas diferentes (variando no número e tipo de argumentos)

Implementação de Classes em Java


Palavra reservada

Nome da classe


```
class ContaBancaria {  
    public int numero;  
    private int senha;  
    protected double saldo;  
    public String nome titular;  
  
    public void depositar(double valor) {  
        this.saldo += valor;  
    }  
}
```

Implementação de Classes em Java

Escopo da classe



```
class ContaBancaria {  
    public    int    numero;  
    private  int    senha;  
    protected double saldo;  
    public   String nometitular;  
  
    public void depositar(double valor) {  
        this.saldo += valor;  
    }  
}
```



Implementação de Classes em Java

```
class ContaBancaria {  
    public int numero;  
    private int senha;  
    protected double saldo;  
    public String nome titular;  
  
    public void depositar(double valor) {  
        this.saldo += valor;  
    }  
}
```

Atributos

Métodos

Atributos

```
class ContaBancaria {  
    public int numero;  
    private int senha;  
    protected double saldo;  
    public String nome titular;  
  
    public void depositar(double valor) {  
        this.saldo += valor;  
    }  
}
```

Tipo de dado

Nome

- Não pode começar com números
- Não pode ser palavra reservada

Atributos

Modificadores de acesso

private: acessível apenas dentro da classe

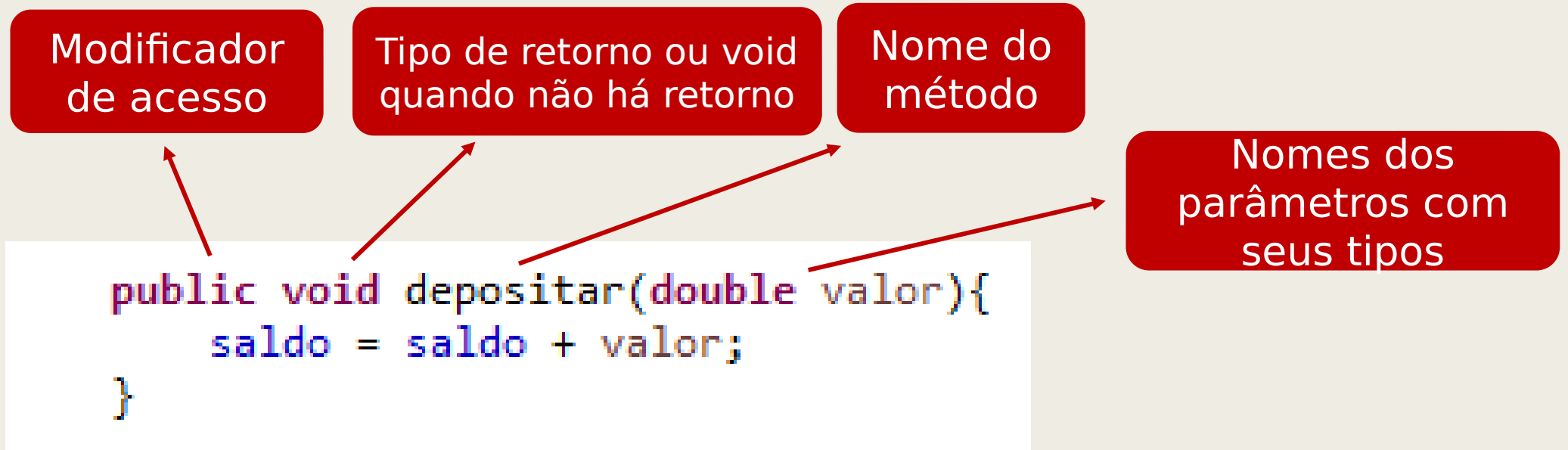
public: acessível por qualquer outra classe

protected: acessível dentro da classe e em suas subclasses (herança)

default (ausência de modificador):
acessível dentro do mesmo pacote

```
class ContaBancaria {  
    public int numero;  
    private int senha;  
    protected double saldo;  
    public String nome titular;  
  
    public void depositar(double valor) {  
        this.saldo += valor;  
    }  
}
```

Métodos



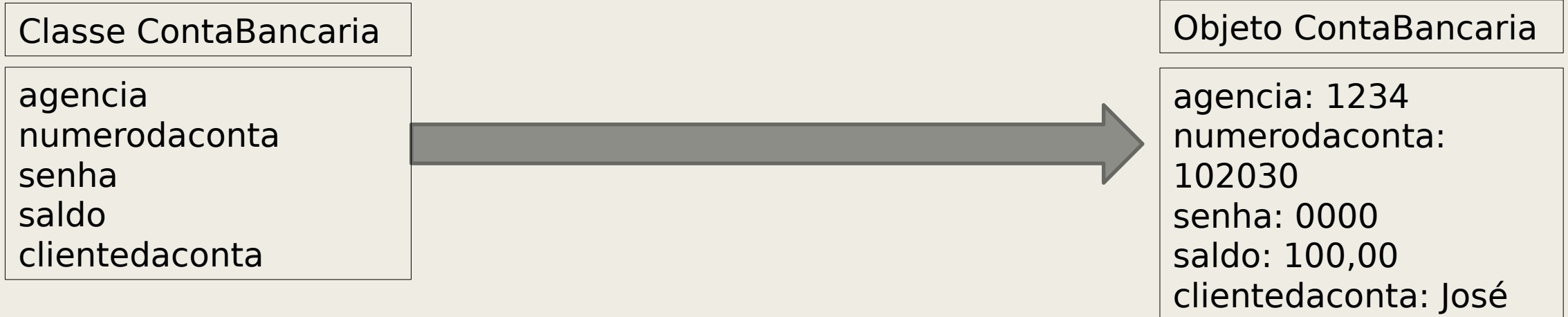
Encapsulamento

```
class ContaBancaria {  
    public int numero;  
    private int senha;  
    protected double saldo;  
    public String nome titular;  
  
    public int getSenha() {  
        return this.senha;  
    }  
  
    public void setSenha(int pwd) {  
        this.senha = pwd;  
    }  
  
    public void depositar(double valor) {  
        this.saldo += valor;  
    }  
    ...  
}
```

Getter

Setter

Classes x objetos



Classes x objetos

```
ContaBancaria conta = new ContaBancaria();  
conta.setAgencia("1234");  
conta.setNumerodaconta("102030");  
conta.setSenha("0000");  
conta.setSaldo(1000.00);
```

Classe ContaBancaria

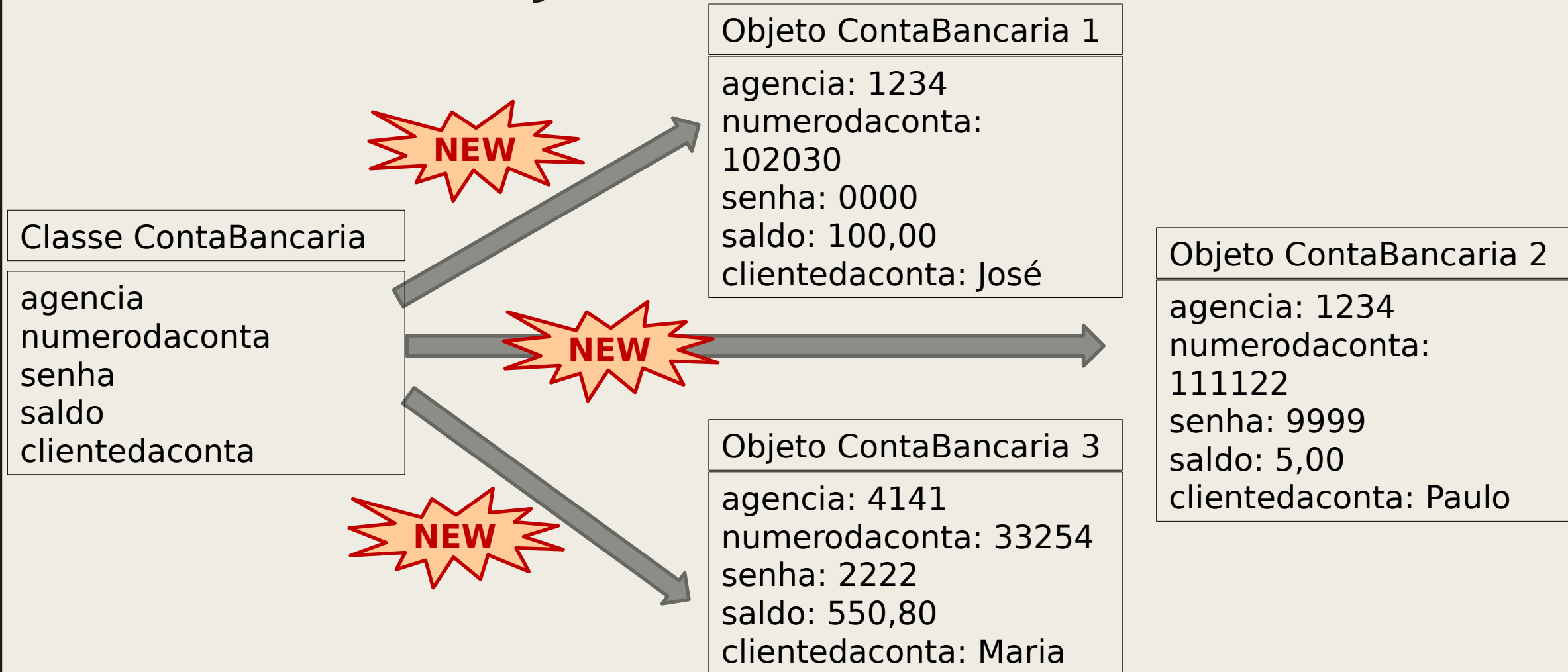
agencia
numerodaconta
senha
saldo
clientedaconta



Objeto ContaBancaria

agencia: 1234
numerodaconta:
102030
senha: 0000
saldo: 100,00
clientedaconta: José

Classes x objetos



Método construtor

- Método utilizado para criar objetos do tipo da classe em questão

- Construtor padrão:

```
public ContaBancaria(){  
  
}
```

Possui o mesmo
nome da classe

Método construtor

- Pode alterar valores dos atributos no momento da criação (instanciação) do objeto

```
public ContaBancaria(int numero, int pwd, String titular) {  
    this.numero = numero;  
    this.senha = pwd;  
    this.nometitular = titular;  
    this.saldo = 0.0;  
}
```

Método construtor

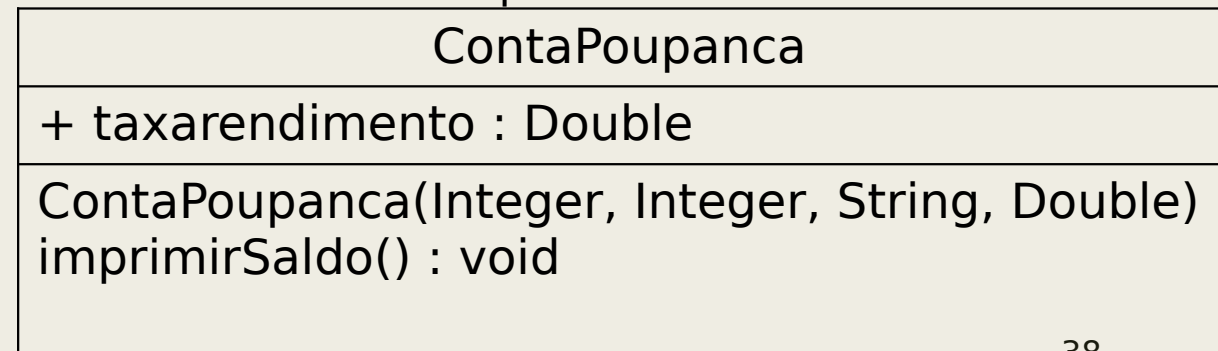
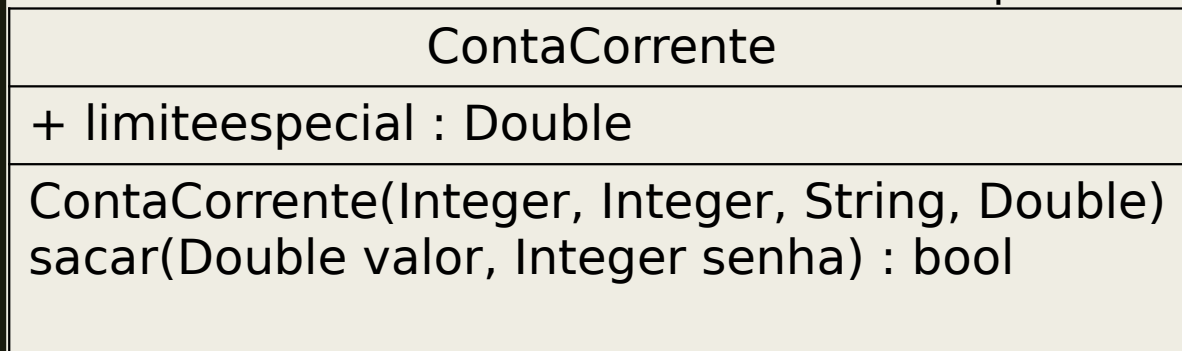
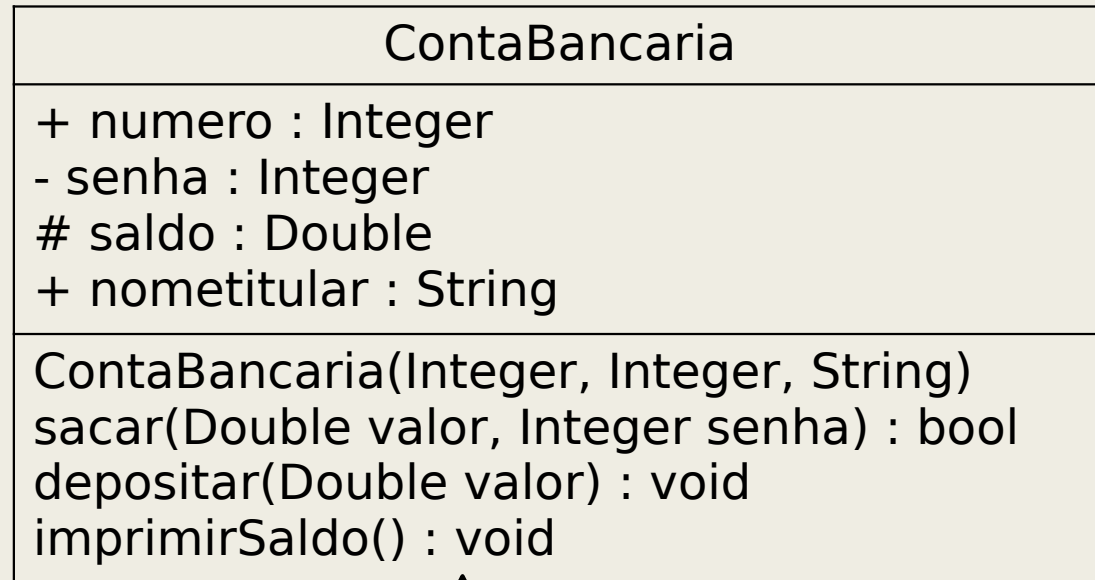
■ Posso ter mais de um método construtor?

Método construtor

- O que acontece se todos os métodos construtores de uma classe forem privados (private)?
 - *A classe não pode ser instanciada!*

Prática 1

■ Implemente, em Java, as seguintes classes de acordo com o diagrama UML:



Prática 1

No método main, implementar as seguintes operações:

1. Instanciar 1 conta corrente (CC) com R\$ 1000,00
2. Instanciar 1 conta poupança (CP) com R\$ 500,00
3. Depositar R\$ 390,00 na CC
4. Imprimir saldo da CC
5. Depositar R\$ 900,00 na CP
6. Imprimir saldo da CP
7. Sacar R\$ 40,00 da CC
8. Imprimir saldo da CC
9. Sacar R\$ 50,00 da CP
10. Imprimir saldo da CP

Implementação de Classes em Java

```
class ContaBancaria {  
    public    int    numero;  
    private  int    senha;  
    protected double saldo;  
    public   String nome titular;  
    ...  
}
```

Falta incluir mais dados do cliente da conta (sobrenome, endereço, etc.)!

Implementação de Classes em Java

```
class ContaBancaria {  
    public    int    numero;  
    private  int    senha;  
    protected double saldo;  
    public    String nometitular;  
    public    String sobrenometitular;  
    public    String endereco;  
    ...  
}
```


Implementação de Classes em Java

```
class ContaBancaria {  
    public    int    numero;  
    private  int    senha;  
    protected double saldo;  
    public    String nometitular;  
    public    String sobrenometitular;  
    public    String endereco;  
    ...  
}
```

Estas informações não são de responsabilidade da ContaBancaria. Por isso, esta não é a melhor solução!

Implementação de Classes em Java

Criamos, então, uma classe para representar o Cliente.

```
class Cliente {  
    public String nome titular;  
    public String sobrenome titular;  
    public String endereco;  
}
```



Para utilizá-la como um tipo de dados.

```
class ContaBancaria {  
    public int numero;  
    private int senha;  
    protected double saldo;  
    public Cliente titular;  
}
```

Prática 2

1. No mesmo programa criado na Prática 1, implemente outras três classes: Cliente, ClientePF (herda da classe Cliente), ClientePJ (herda da classe Cliente) e seus respectivos atributos.
2. Altere o atributo nome titular da classe ContaBancaria para considerar as classes criadas.
3. Altere os métodos construtores das classes da Prática 1 para receber os dados do cliente corretamente.
4. Altere o método main para criar os clientes corretamente.

Interface

- Define um “contrato” que indica o comportamento de uma classe (o objeto **deve** fazer e não como fazer)
- Não possui implementação, somente a definição dos métodos
- As classes que “assinam” esse “contrato” se responsabilizam por implementar tais métodos

Implementação de interfaces

- Basta utilizar a palavra-chave `interface` na definição da classe.

```
public interface Autenticavel {  
    //aqui contém apenas a assinatura dos métodos  
}
```

- Outras classes podem “assinar o contrato” da interface utilizando a palavra-chave `implements`.

```
public class Cliente implements Autenticavel {  
}
```

Prática 3

- O banco precisa tributar o dinheiro de alguns bens. No caso, a conta corrente é tributável com 1% do saldo.
- Para isso, “todas as classes que quiserem ser tributável precisam saber retornar o valor do imposto, devolvendo um double”.
- Implemente a interface Tributavel

```
public interface Tributavel {  
    public double getValorImposto();  
}
```

Métodos estáticos

- São métodos que podem ser invocados sem a existência de uma instância da classe em questão.
- Para que isso seja possível, a implementação do método não pode depender de valores das variáveis de instância (nem métodos de instância) da classe, ou seja, é um componente isolado.
- Por exemplo, o método `min(x, y)` da classe `Math` retorna o menor valor entre `x` e `y`.

Métodos estáticos

■ Para implementar um método estático:

– *public static int min(int x, int y){*
}

Nome da
classe

Método
estático

Parâmetr
os

■ Para executar o método: `Math.min(10, 20);`

Prática 4

- Altere o método imprimirSaldo() da classe ContaBancaria para ser estático.
- O que deve ser alterado?

Referências

- DEITEL, H. M. Java: como programar, 8. ed. São Paulo: Prentice Hall, 2010. Capítulos 1, 3, 9 e 10.