Universidade Federal do Agreste de Pernambuco

OBJETOS

Thaís Alves Burity Rocha

Agenda

- Construtores
- Referências e aliasing
- String
- Igualdade entre referências e objetos
- Coletor de lixo
- Associação entre classes

Criação de objetos

- Primeiro criamos objetos para poder usá-los
- Sempre que um objeto é criado, ele vai para uma área de memória conhecida como **Heap**
  - O espaço de memória alocado por cada objeto varia
  - O programador não precisa se preocupar com isso
- Objetos são criados através de um construtor

```
Conta conta = new Conta();
```

Construtor

- Toda classe deve ter pelo menos um construtor
- Define como os atributos de um objeto devem ser inicializados
- Quando a classe não define um construtor, o compilador gera o **construtor default**
  - Não fica visível no código
  - Foi o que usamos até o momento!
- Parece um método, mas não possui tipo de retorno

```
classe (parâmetros) {...}
```

Construtor de Conta

```
class Conta {

    String numero;
    double saldo;

    Conta(String numero, double saldo){
        this.numero = numero;
        this.saldo = saldo;
    }

}
```

Construtor default

- Inicializa os atributos com seus valores **default**
  - Atributos são sempre inicializados, variável local não
- Se fossemos escrever o código, seria isso...

```
Conta() {
    this.numero = null;
    this.saldo = 0.0;
}
```

```
Conta() {


}
```

Classe com vários construtores

- Utilidade: Flexibilidade

```
class Conta {
   String numero;
   double saldo;
   Conta(String num, double val){
      numero = num;
      saldo = val;
   }
   Conta(String num){
      numero = num;
   }
}
```

A diferença está na **lista de parâmetros** (quantidade, tipo, ordem)

Construtor chamando construtor

- Utilidade: Reuso de código

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

   Conta(String num){
      numero = num;
   }

   Conta(String num, double val){
      this(num);
      saldo = val;
   }
}
```

Quando utilizado, deve ser o primeiro comando

Chamada do construtor

- Utiliza o operador **new**

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

```
Conta c = new Conta("11139-2",100);
```

Variável de referência

- Pode ser feita no corpo da classe, dentro de métodos ou dentro de construtores de outras classes

Referências

- Manipulam objetos
- **Deve ser de tipo compatível**

```
Conta c = new Conta("11139-2",100);
```

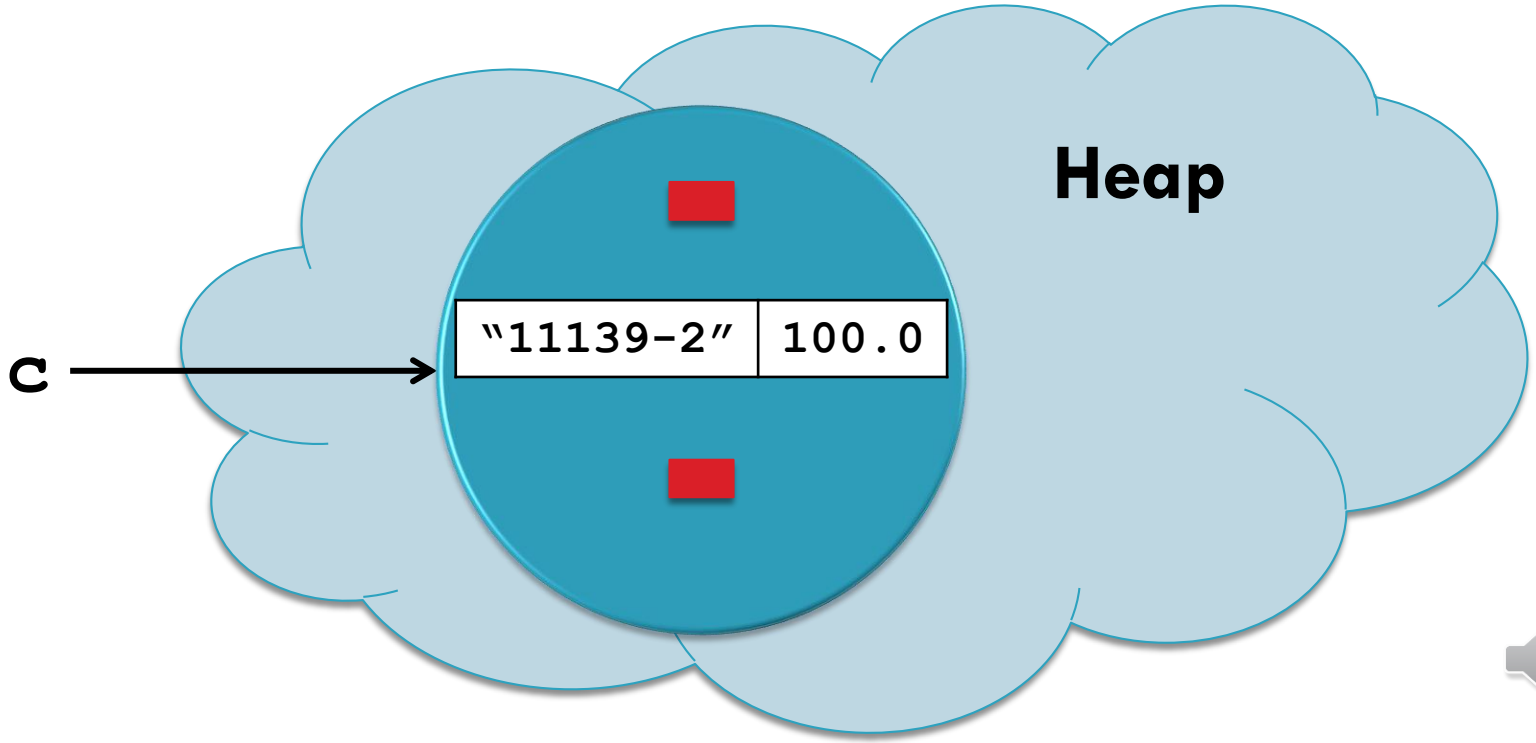
- **c** é um **"controle remoto"** para um objeto Conta
- **c** é usada para acessar o objeto e fazer algo

```
c.saldo = 500; //acesso à atributo
c.debitar(100); //chamada de método
```

Visão detalhada de referências

```
Conta c;
```

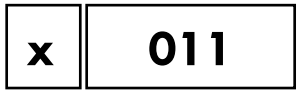
```
c = new Conta("11139-2",100);
```



Primitivos x referenciados

- Variáveis de tipo primitivo guardam o conteúdo (bits que representam o valor)

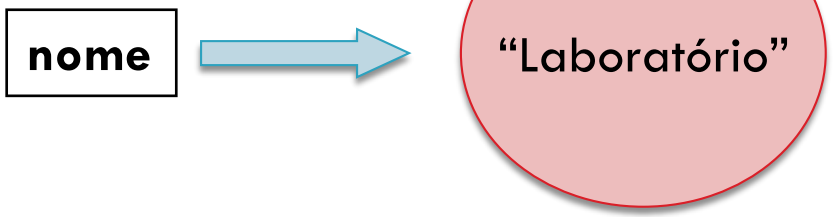
```
int x = 3;
```



- Variáveis de tipo referenciado guardam endereços

```
String nome = null;
```


```
String nome = "Laboratório";
```



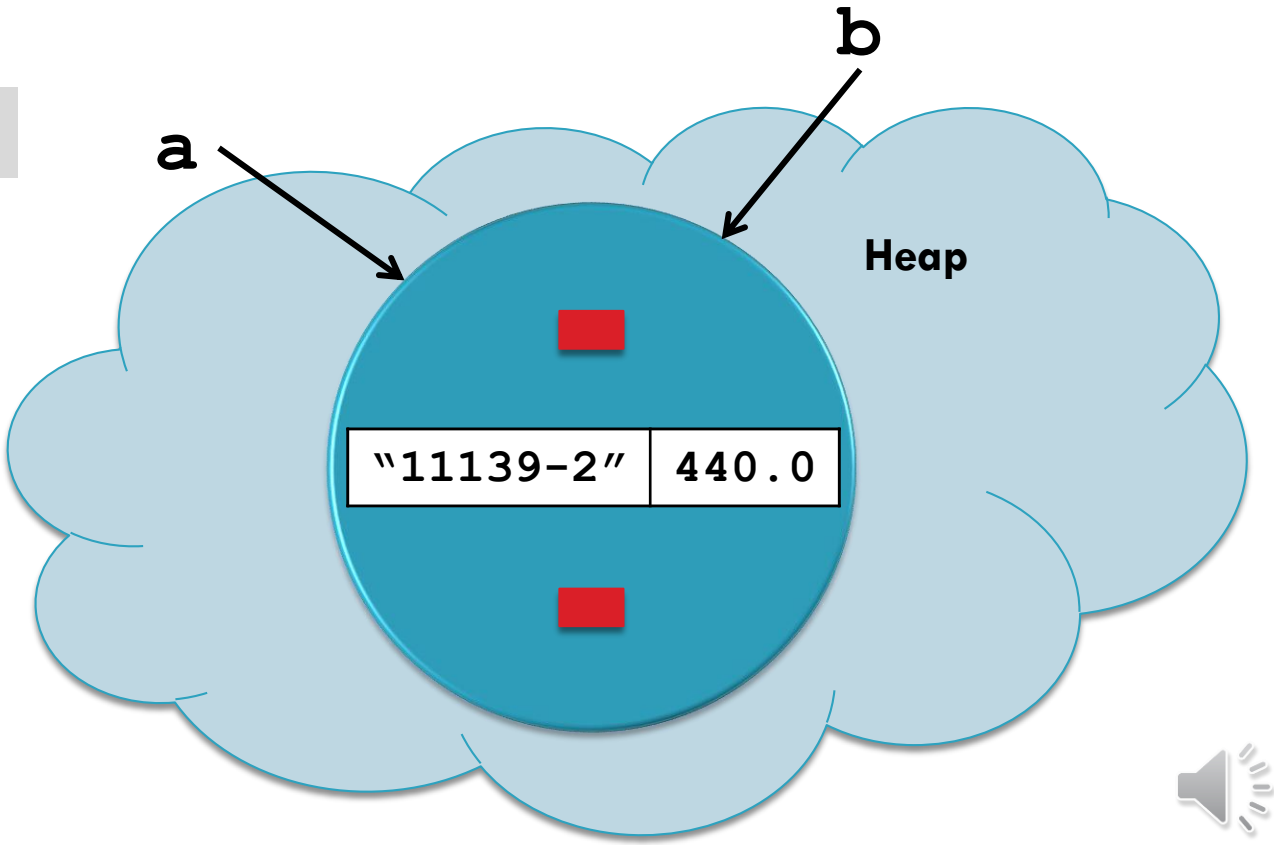
Aliasing

```
Conta a;
```

```
a = new Conta("11139-2",340.0);
```

```
Conta b = a;
```

```
b.creditar(100);
```



String

- Classe especial
- Há 2 formas de criar um objeto String
  - Atribuindo um literal simplesmente

```
String nome = "Maria";
```

O objeto é criado no **Pool de Strings**

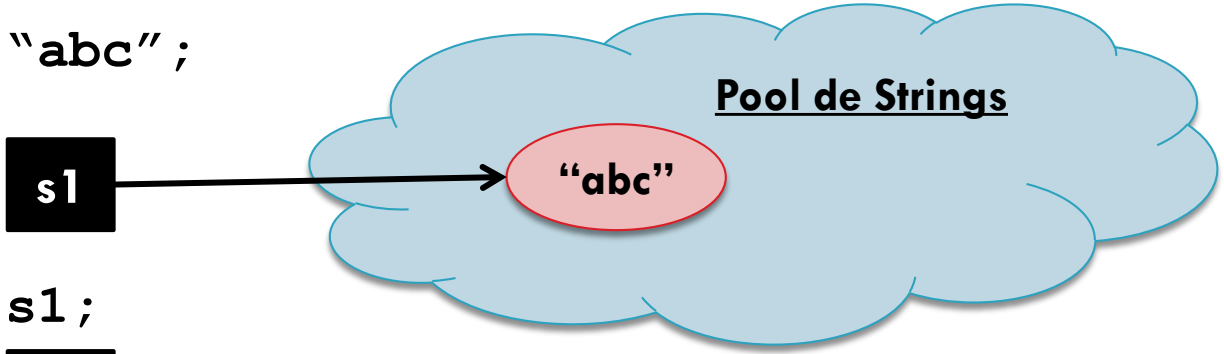
  - Chamando o construtor

```
String nome = new String("Maria");
```

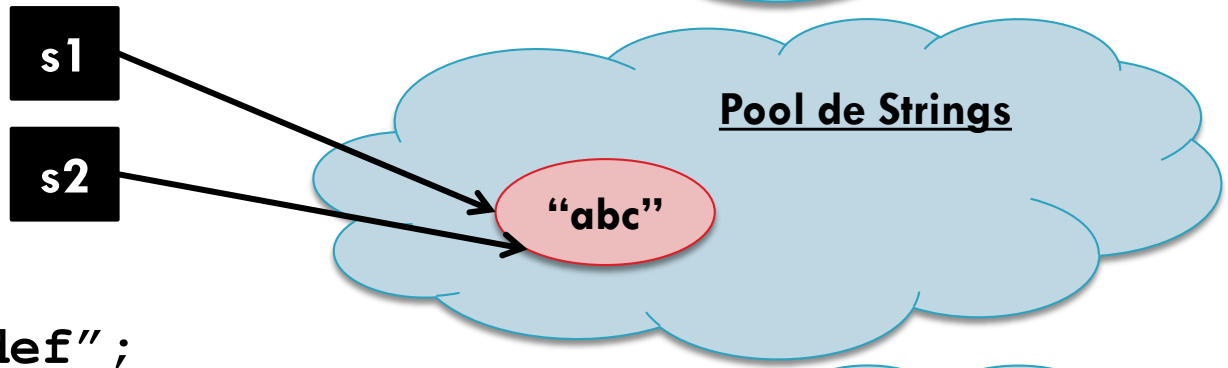
O argumento é criado no **Pool de Strings** (caso ainda não exista) e outro objeto é criado no Heap

Strings são imutáveis

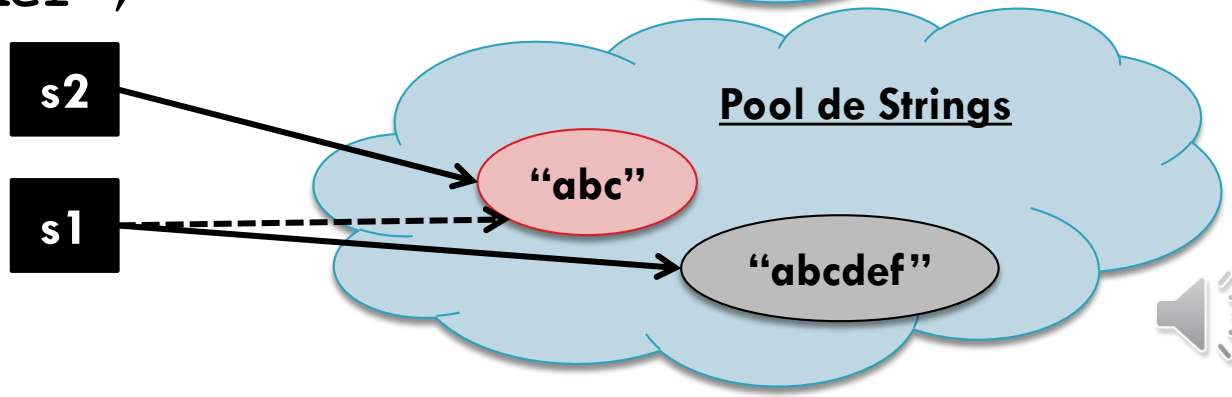
```
String s1 = "abc";
```




```
String s2 = s1;
```




```
s1 = s1 + "def";
```



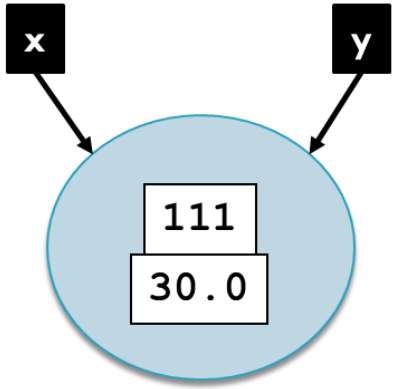
Qual a vantagem em ser imutável?

- Economia de memória
- Sempre que uma nova String vai ser criada, a JVM verifica se ela já existe
- Se existir, apenas é criada uma nova referência (aliasing)

Igualdade entre referências

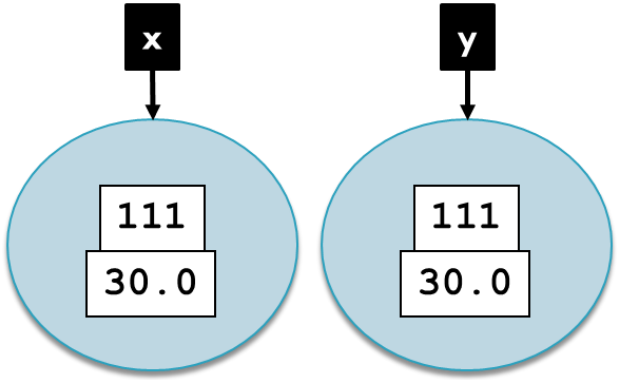
```
Conta x = new Conta("111",30);
Conta y = x;
if(x == y) {...}
```

TRUE




```
Conta x = new Conta("111",30);
Conta y = new Conta("111",30);
if(x == y) {...}
```

FALSE



- Compara endereços de memória apenas

Igualdade entre objetos

- Usar o método **equals()**
- **Por hora, só iremos usar esse método com String**

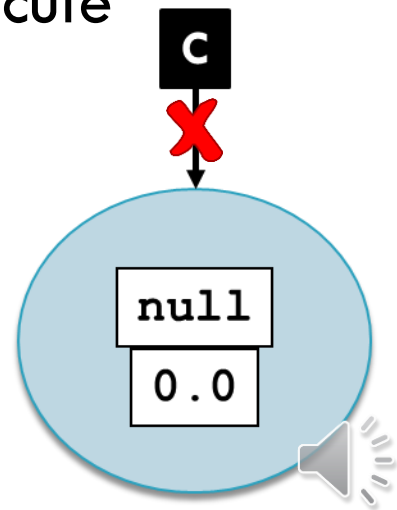
```
String s1 = "casa";
String s2 = "casa";
String s3 = new String("casa");

boolean b1 = s1 == s2;          //b1 vale true
boolean b2 = s1.equals(s2);//b2 vale true
boolean b3 = s1 == s3;          //b3 vale false
boolean b4 = s1.equals(s3);//b4 vale true
```

É mais seguro comparar Strings com equals()

Remoção de objetos

- Em Java, não existe mecanismo de remoção explícita de objetos, como o método **free()** de C++
- A JVM gerencia a remoção de objetos através do **Garbage Collection** (coletor de lixo)
  - A JVM decide quando fazer a coleta de lixo
  - O programador **não pode** obrigar que execute
  - Lógica: Remover objetos inacessíveis

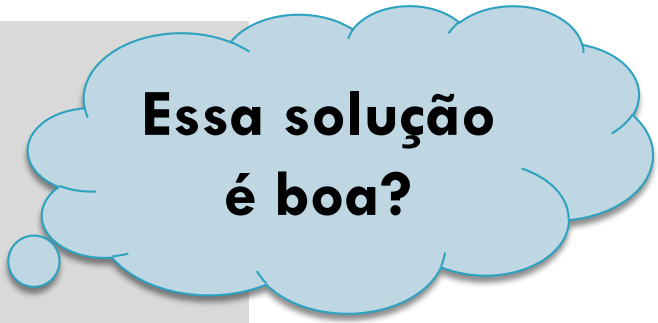
```
Conta c = new Conta();
c = null;
```



Problema

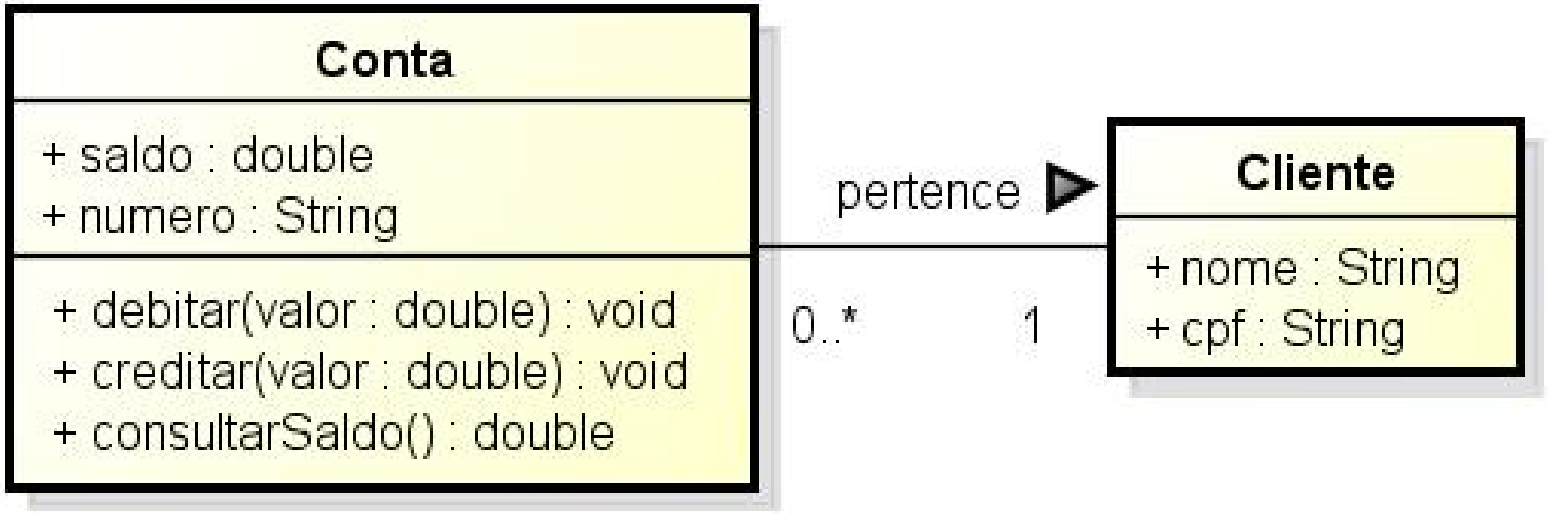
- Um cliente pode ter mais de uma conta
- Como descobrir o proprietário de uma conta?
- Como descobrir todas as contas de um cliente?

```
class Conta {

   String numero;
   double saldo;
   String nome, sobrenome;
   ...
}
```



Associando classes

- Um cliente pode ter mais de uma conta
- Como descobrir o proprietário de uma conta?
- Como descobrir todas as contas de um cliente?



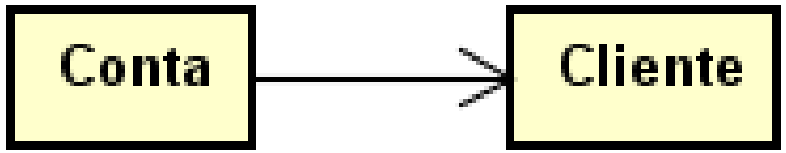
Notação do diagrama de classes da UML

Relacionamento TEM-UM (HAS-A)

- É uma forma de reutilizar classes

Uma possível solução

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

    Conta(String n, double s, Cliente c){
        numero = n;
        saldo = s;
        cliente = c;
    }
    ...
}
```

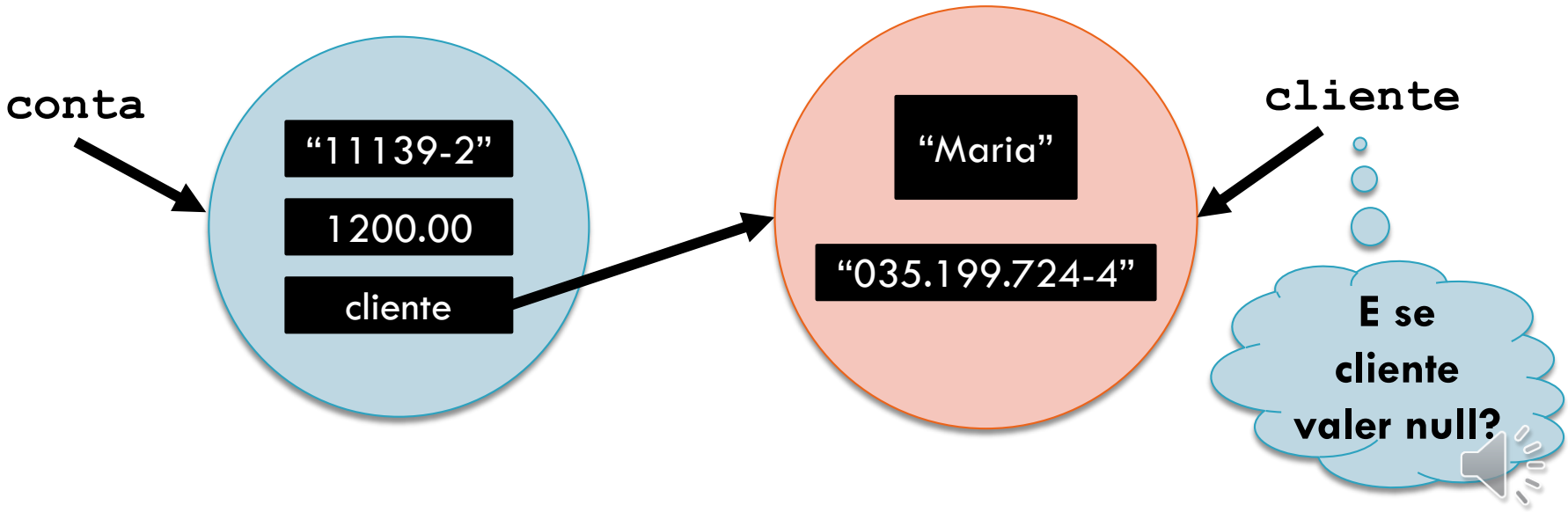


Notação do diagrama de classes da UML

Ilustrando...

```
Cliente cliente = new Cliente("Maria", "035.199.724-4");
Conta conta = new Conta("11139-2", 1200, cliente);
conta = null;
```

Mesmo que a conta não exista mais, o cliente continua existindo

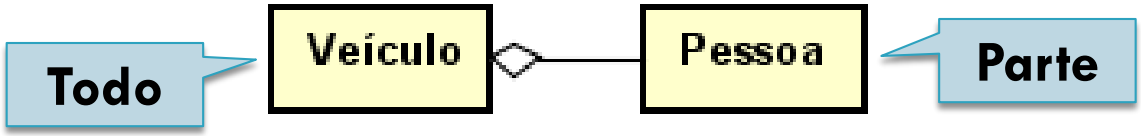


Analisando associação entre classes

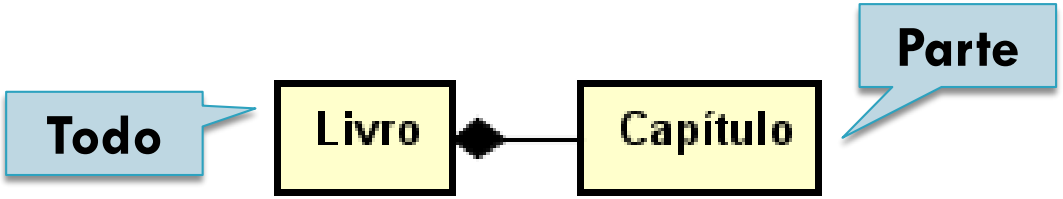
- Nem toda associação significa posse/propriedade no mundo real
- Veículo e pessoa
  - A pessoa pode ser dono do veículo
  - A pessoa poder ser motorista do veículo
  - A pessoa pode ser passageiro do veículo
  - O veículo e a pessoa são independentes: Existe pessoa sem veículo e existe veículo sem pessoa

Agregação e composição UML

- Notação do diagrama de classes
- Agregação: As partes existem independente do todo



- Composição: As partes não existem sem o todo, ou seja, se o todo é destruído, as partes também o são



A associação entre Cliente e Conta é de qual tipo?