


Programação Orientada a Objetos

Aula 3 – Manipulação de objetos



Prof. Ronald Chiesse
ronaldsouza@dcc.ufrj.br



Encapsulamento e modificadores de visibilidade

Encapsulamento em POO

- **Encapsulamento** é isolar a lógica de negócios do programa: a manipulação dos **dados** obedece a regras que estão **centralizadas** (em sua classe) e não espalhadas por todo o programa.

Métodos (interface da classe)

liga
desliga
acelera
freia
abastece

Atributos e corpo dos métodos

Usuário não acessa
essa parte (e nem se
preocupa)

Modificadores de visibilidade

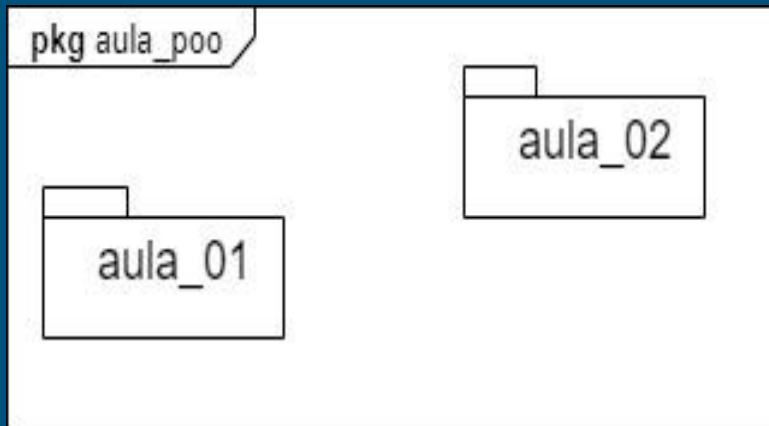
- Funcionam na declaração de classes, atributos e métodos

public	protected	(default)	private
Todas as classes do projeto têm acesso	Somente as classes do próprio pacote tem acesso, assim como classes-filhas	Somente as classes do próprio pacote tem acesso	Somente a própria classe tem acesso

- O que são pacotes em Java?
 - Conjunto de classes localizadas na mesma estrutura hierárquica de diretórios
 - Cada pacote pode conter outros subpacotes dentro

Exemplo de pacote

- Os pacotes só podem ter letras minúsculas no nome



```
\---aula_poo
  +---aula_01
  |       Lista.java
  |
  \---aula_02
       Cliente.java
       Funcionario.java
```

Implementação no Java

- Para definir o pacote de uma classe, colocar ***package*** *nomepacote.completo*; no início do arquivo
- Para chamar classes de um outro pacote, colocar ***import*** *nomepacote.Classe*; (para uma classe específica) ou ***import*** *nomepacote.**; (para todas as classes do pacote)

```
package aula_poo.aula_01;  
  
public class Lista  
{  
    ...  
}
```

```
package aula_poo.aula_02;  
  
public class Cliente  
{  
    ...  
}
```

```
package aula_poo.aula_02;  
import aula_poo.aula_01.*;  
  
public class Funcionario  
{  
    ...  
}
```

Como manipular objetos?

- Similar às linguagens estruturadas, um objeto em POO tem um espaço em memória para armazenar seu estado (conjunto de **atributos**) e um conjunto de operações que podem ser aplicadas ao objeto (conjunto de **métodos**)
- Diferentes objetos podem interagir entre si, pelos seus respectivos métodos
- Objetos são manipulados através de **variáveis**. Cada variável armazena a referência em memória de seu objeto.
 - **this** → referência especial criada dentro de cada objeto apontando para si próprio que pode ser usada dentro dos métodos do objeto

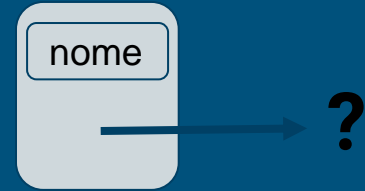
Declaração e instanciação de objetos

Frase
+ valor:String
+ Frase(String)

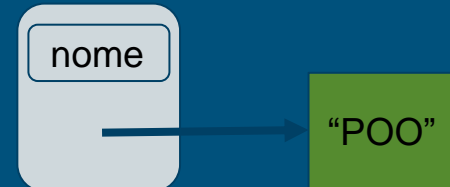
Declaração e instanciação de objetos

- Ao declarar uma variável que tenha como tipo uma classe, estamos declarando somente a referência para o objeto da classe
 - Inicialmente o valor dessa variável não é válido (**Null**)
- Quando instanciamos o objeto (comando **new**), criamos o objeto na memória e retornamos a referência para esse espaço na memória
 - Como um ponteiro em C, mas o programador de POO não pode manipular a memória!
- Como poderíamos, num exemplo básico, usar o **this??**

```
Frase nome;
```

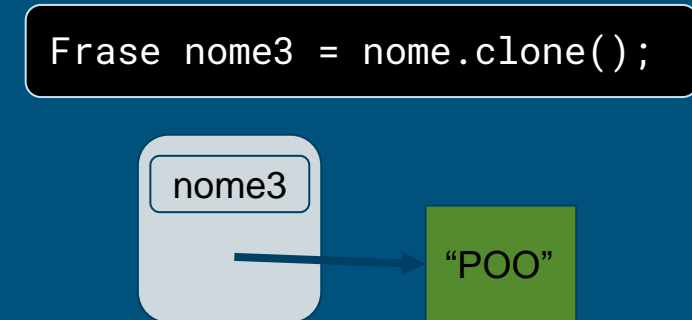
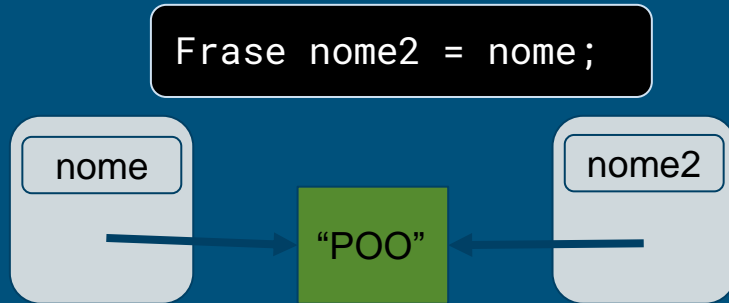


```
nome = new Frase("POO");
```



Tipos de cópias de objetos

- Como as variáveis de tipo de classe armazenam somente a referência, ao atribuímos uma variável a outra, copiamos somente a referência do objeto
 - **Shallow copy** ou cópia rasa
- Caso queira fazer uma cópia do objeto, usar o método **clone()** ou criar um método específico
 - **Deep copy** ou cópia profunda



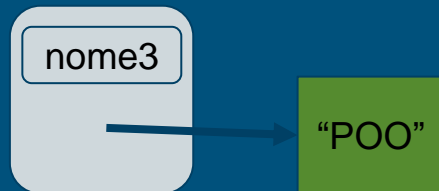
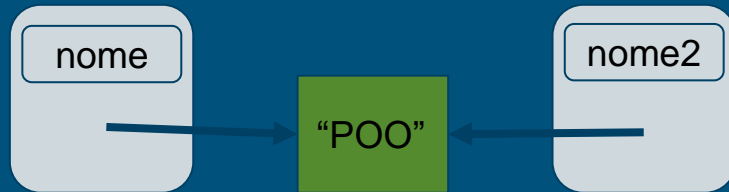
Shallow copy x Deep copy

Shallow copy	Deep copy
Armazena a referência para o endereço de memória original	Armazena uma cópia dos valores do objeto (novo endereço de memória)
Mudanças na cópia refletem no objeto original	Mudanças na cópia não refletem no objeto original
Cópia rápida	Cópia lenta

Igualdade de objetos

Igualdade de referência vs. igualdade semântica

- Ao compararmos as variáveis diretamente, estamos olhando para a igualdade de referência, que retorna **true** se ambas variáveis apontam para o mesmo objeto (`nome == nome2`)
 - `nome == nome3` → **false**
- Para compararmos os valores internos, é preciso sobrescrever a função **equals()** para comparar o valor de cada atributo
 - `nome.equals(nome3)` → **true**



Remoção de objetos

Remoção de objetos

- Como a criação de objetos é um processo de alocação dinâmica de memória, é preciso excluir os objetos e devolver a memória usada para o SO ao final da execução do programa
- Duas maneiras de se devolver a memória:
 - O programador precisa explicitamente deletar e destruir os objetos (e.x.: operador *delete* da linguagem C++)
 - A própria linguagem tem um **garbage collector** que verifica quais objetos não tem nenhuma referência válida (Java)

Garbage Collector (GC)

- Como podemos assinalar um objeto para o GC coletar?
 - Removendo todas as referências ao objeto existentes
 - Chamar o método `java.lang.System.gc()` (sugere a execução do GC, mas não força)
- Caso seja necessário excluir objetos criados dentro do objeto pai, podemos sobrescrever o método **finalize()**
 - Chamar o método `java.lang.System.runFinalizaion()` para sugerir a execução do método `finalize` dos objetos descartados