



UNIVERSIDADE  
FEDERAL DO CEARÁ

TÓPICO  
**08**



# PROGRAMAÇÃO ORIENTADA A OBJETOS

Prof. Marcos Vinicius de Andrade Lima  
E-mail: [marcos.vinicius@ufc.br](mailto:marcos.vinicius@ufc.br)



## Olá!

**Sou Marcos Vinicius**

No tópico passado nós aprendemos  
como trabalhar com *arrays*...

Neste tópico aprenderemos como  
manipular **coleções e**  
**mapeamentos!**

“

**Toda empresa precisa de gente que erra, que não tem medo de errar e que aprenda com o erro** (Bill Gates)



**Vale a pena ver de novo**

## INTRODUÇÃO

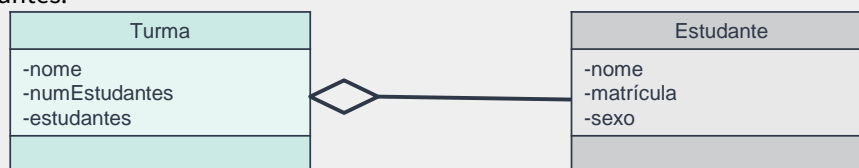
- A agregação é um mecanismo fundamental de reuso de código
- A agregação define uma relação do tipo “tem um” (ou relação todo-parte)
- **Exemplos:**
  - Uma turma **é composta** de estudantes
  - Uma empresa **é composta** de departamentos
  - Um aparelho **é composto** de peças
- Sempre que tal situação acontecer, pode-se aplicar o conceito da **AGREGAÇÃO**

Prof. Marcos Vinicius – UFC/Russas - POO

5/35

## INTRODUÇÃO

- Um objeto agregado é criado a partir de outros objetos constituintes, que são suas partes
- Veja abaixo a representação em UML para uma turma que é uma agregação de estudantes:



- **Código para a agregação Turma-Estudante:**

```

public class Turma {
    String nome;
    int numEstudantes;
    Estudante estudantes[];
    ... }
  
```

Prof. Marcos Vinicius – UFC/Russas - POO

6/35

## PARA SEMPRE LEMBRAR...

**Objetos** em Java **não podem conter** outros **objetos**.  
Eles podem ter **referências** a outros objetos!



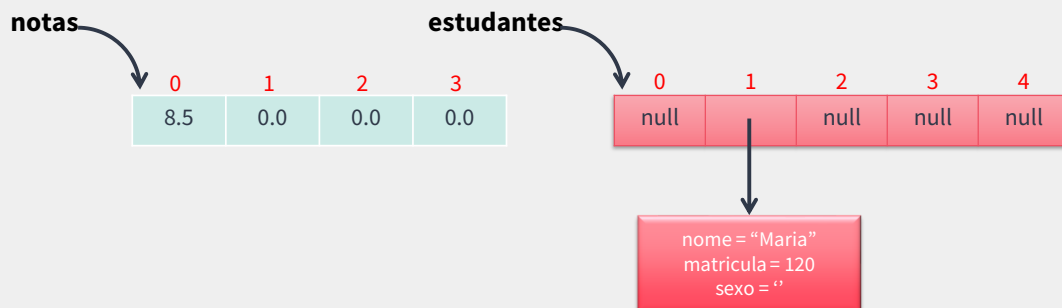
**Java suporta agregação de objetos por REFERÊNCIA!**

Prof. Marcos Vinicius – UFC/Russas - POO

7/35

## MANIPULANDO ARRAYS

```
Estudante estudantes[] = new Estudante[5];
double notas[] = new double[4];
notas[0] = 8.5;
estudantes[1] = new Estudante("Maria");
estudantes[1].setMatricula(120);
```



Prof. Marcos Vinicius – UFC/Russas - POO

8/35

## MANIPULANDO ARRAYS

```
public class Turma {
    Estudante estudantes[] = new Estudante[5];
    String nome;
    int numEstudantes = 0;

    public void matricular (Estudante e) {
        estudantes[numEstudantes] = e;
        numEstudantes++;
    }
}
...
Estudante novoEst = new Estudante("Maria");
Turma turma1 = new Turma("POO");
turma1.matricular( novoEst );
...
```



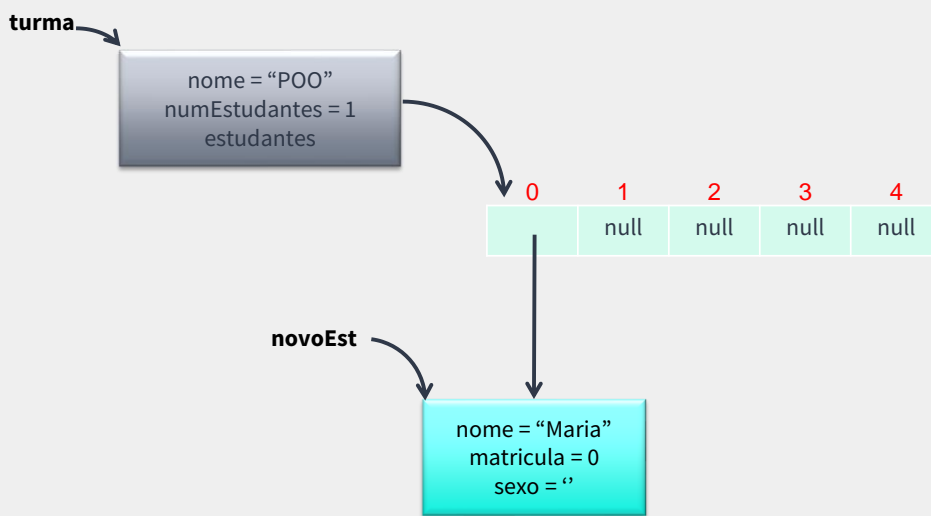
**Cuidado que o método matricular. Ele poderá apresentar problemas!**

**Como corrigir isso?**

Prof. Marcos Vinicius – UFC/Russas - POO

9/35

## MANIPULANDO ARRAYS



Prof. Marcos Vinicius – UFC/Russas - POO

10/35



# CUIDADO!!!

Em Java **NÃO** é possível **redimensionar *arrays***. Caso seja necessário, use **LISTAS**!



## Coleções e Listas

## COLEÇÕES E LISTAS

- Uma coleção (*collection*) permite que um grupo de objetos seja tratados como uma unidade
- Objetos arbitrários podem ser armazenados, recuperados e manipulados como elementos de coleções
- Coleções estão disponíveis no pacote `java.util` e podem aparecer na forma de listas e conjuntos

Prof. Marcos Vinicius – UFC/Russas - POO

13/35

## LISTAS

- Listas são coleções que mantêm seus elementos em ordem (também chamado de sequência), e permite elementos duplicados
- Em uma lista não vazia, o primeiro elemento tem índice 0 e o último **size()-1**
- Diferente dos *arrays* tradicionais do Java, uma Lista simples **não tem** um tipo determinado, ou seja, o primeiro elemento pode ser de um tipo diferente do segundo elemento.
- Todos os elementos da lista devem ser objetos – **tipos primitivos não são permitidos!**

Prof. Marcos Vinicius – UFC/Russas - POO

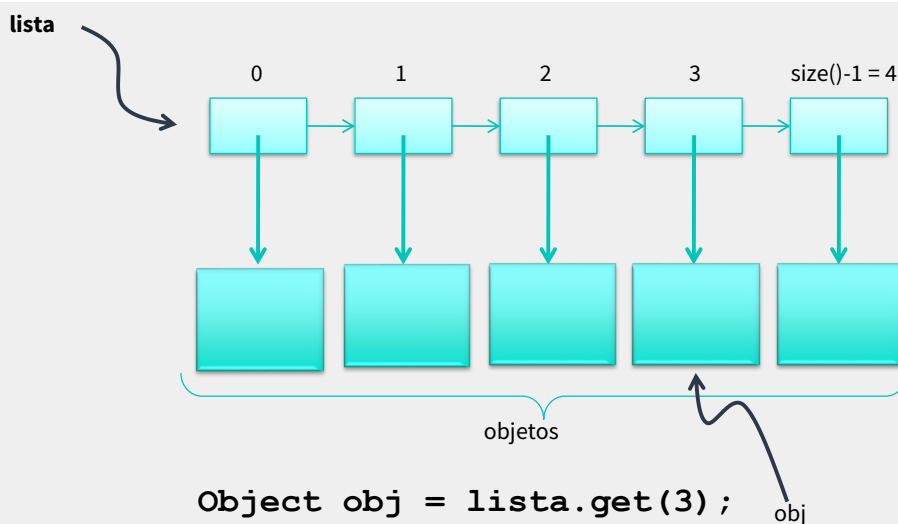
14/35



# CUIDADO!!!

Uma operação sob um índice de lista que não existe gera uma **IndexOutOfBoundsException** (talvez uma velha conhecida)

## VISUALIZANDO UMA LISTA





## PRINCIPAIS MÉTODOS EM UMA LISTA



<b>int size();</b>	Retorna o tamanho da lista.
<b>boolean isEmpty();</b>	Retorna true se a lista estiver vazia.
<b>boolean add(Object element);</b>	Adiciona um objeto ao final da lista.
<b>Object get(int index);</b>	Retorna o elemento no índice especificado.
<b>Object set(int index, Object element);</b>	Substitui o elemento no índice especificado com o parâmetro passado.
<b>boolean contains(Object element);</b>	Verifica se o objeto passado como parâmetro é membro da coleção.
<b>int indexOf(Object o);</b>	Retorna o índice da primeira ocorrência do elemento especificado na lista, se o elemento existir, caso contrário retorna -1.
<b>void clear();</b>	Remove todos os elementos da list.
<b>Object remove(int index);</b>	Remove o elemento no índice especificado, reduzindo o tamanho da lista.

Prof. Marcos Vinicius – UFC/Russas - POO

17/35

## INSERINDO UM ELEMENTO NA LISTA

```
import java.util.*;
public class TesteArrayList {
    public static void main (String args[])
    {
        ArrayList al = new ArrayList();
        al.add( "POO-Java" );
        al.add( new Integer("10") );
        al.add( new ContaCorrente() );
        al.add( new Professor("Marcos") );
        al.add( new Estudante("Maria") );
        new TesteArrayList().print( al );
    }
    ...
}
```

Prof. Marcos Vinicius – UFC/Russas - POO

18/35

## PERCORRENDO UMA LISTA

```
...  
public void print( ArrayList a )  
{  
    for( int i = 0; i < a.size(); i++ )  
    {  
        System.out.println( a.get(i) );  
    }  
}  
}
```

Prof. Marcos Vinicius – UFC/Russas - POO

19/35

## PERCORRENDO UMA LISTA COM “FOR EACH”

```
...  
public void print( ArrayList a )  
{  
    for( Object o : a )  
    {  
        System.out.println( o.toString() );  
    }  
}  
...  
}
```

Prof. Marcos Vinicius – UFC/Russas - POO

20/35

## PRESTA ATENÇÃO!

- O método **get( int index )** retorna um **Object**, portanto, pode ser necessário realizar conversões (*casting*) se o tipo for diferente de **Object**
- Recuperando elementos do **ArrayList** e realizando *casting*:


```
public class TesteArrayList4 {
    public void print( ArrayList al )
    {
        for( int i = 0; i < al.size(); i++ )
        {
            Estudante e = (Estudante)al.get(i);
            e.exibir();
        }
    }
}
```

OMG!



Prof. Marcos Vinicius – UFC/Russas - POO

21/35



# Happens!

O estagiário  
colocou de novo um  
estudante na lista  
de professores!!!



## java.lang.ClassCastException

## ARRAYLIST DE TIPO ESPECÍFICO

- A versão 1.5 do Java introduziu o conceito de listas de tipos específicos. Assim, o *casting* anterior não é mais necessário
- Para isso, use parênteses angulares ( <Tipo> ) para definir o tipo de sua lista

```
public class TesteArrayList {
    public void static main( String args [] ) {
        ArrayList<Integer> li = new ArrayList<Integer>();
        li.add( new Integer(1) );
        li.add( 2 );
        for (int i=0; i < li.size(); i++) {
            int n = li.get(i);
            System.out.println( n );
        }
    }
}
```

Prof. Marcos Vinicius – UFC/Russas - POO

23/35

## LISTAS SINCRONIZADAS (RIP VECTOR)

```
List lista =
    Collections.synchronizedList(new ArrayList());

synchronized(lista) {
    Iterator i = lista.iterator();
    while (i.hasNext()) {
        System.out.println(i.next());
    }
}
```



Prof. Marcos Vinicius – UFC/Russas - POO

24/35



# Mapeamentos

## MAPEAMENTOS

- Um Map (mapeamento) define mapeamentos de chaves para valores.
- Um Map não permite chaves duplicadas, ou seja, as chaves são únicas.
- Cada chave mapeia para no máximo um valor.

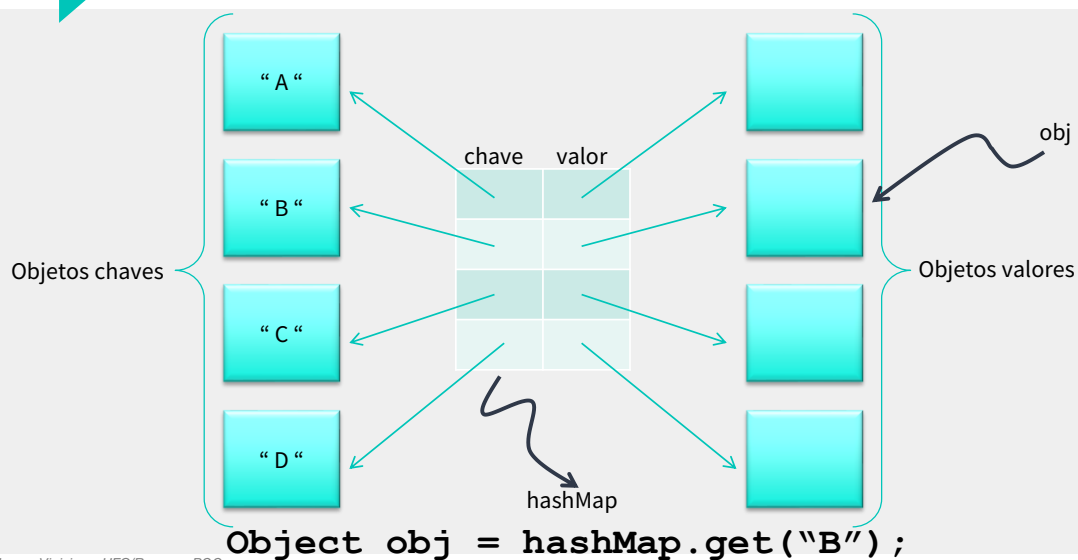
## HASHMAP

- **HashMap** é um dos mapeamentos mais utilizados!
- Fica a dica:
  - ✓ Se você precisa **acessar a lista sequencialmente**, use um **ArrayList**
  - ✓ Se você precisa **acessar a lista aleatoriamente**, utilizando uma chave de busca, use **HashMap**

Prof. Marcos Vinicius – UFC/Russas - POO

27/35

## VISUALIZANDO UM MAPEAMENTO



Prof. Marcos Vinicius – UFC/Russas - POO

28/35

## PRINCIPAIS MÉTODOS PARA MAPEAMENTOS

<b>int size();</b>	Retorna o tamanho do mapeamento.
<b>boolean isEmpty();</b>	Retorna true se o mapeamento estiver vazio.
<b>boolean put(Object key, Object value);</b>	Insere um mapeamento, isto é, um par <key, value>, também chamado de entrada.
<b>Object get(Object key);</b>	Retorna o valor para o qual a chave (key) é mapeada ou null se nenhum mapeamento for encontrado.
<b>Object remove(Object key);</b>	Remove um objeto do mapeamento
<b>boolean containsKey(Object o);</b>	Verifica se a chave existe.
<b>boolean containsValue(Object o);</b>	Verifica se o valor existe.
<b>void clear();</b>	Remove todos os elementos do mapeamento.



Prof. Marcos Vinicius – UFC/Russas - POO

29/35

## INSERINDO ELEMENTOS NO *HASHMAP*

```
import java.util.*;
public class TesteHashMap {
    public static void main( String args[] )
    {
        HashMap hm = new HashMap();
        Estudante e = new Estudante( "Maria" );
        EstudanteMonitor em =
        new EstudanteMonitor( "João" );
        hm.put( "1", e );
        hm.put( "2", em );
        ...
    }
}
```

Prof. Marcos Vinicius – UFC/Russas - POO

30/35

## BUSCANDO ELEMENTO NO HASHMAP

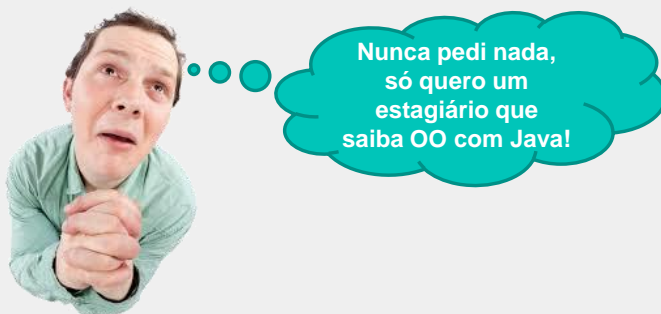
```
...  
    // Buscando o estudante "1"  
    Estudante e1 = (Estudante)hm.get( "1" );  
    // Buscando o estudante "2"  
    EstudanteMonitor e2 =  
        (EstudanteMonitor)hm.get( "2" );  
    e1.exibir();  
    e2.exibir();  
}  
}
```

Prof. Marcos Vinicius – UFC/Russas - POO

31/35

## CUIDADO COM A VOADORA!

Se o **estagiário aprontar** é possível ter um problema de *casting* após receber um objeto do mapeamento!



Prof. Marcos Vinicius – UFC/Russas - POO

32/35



## HASHMAP DE TIPO ESPECÍFICO

- A partir da versão 1.5 do Java também temos tipos específicos para HashMap:

```
public class TesteArrayList {  
    public void static main( String args [] ){  
        HashMap<Integer, String> hm =  
            new HashMap<Integer, String>();  
        hm.put( 2, "Maria" );  
        hm.put( 7, "Pedro" );  
        hm.put( 5, "Madalena" );  
        String s = hm.get(7);  
        System.out.println( s );  
    }  
}
```

Prof. Marcos Vinicius – UFC/Russas - POO

33/35





# Obrigado!

## Mais alguma dúvida?

Acesse o **AME** para mais informações e treinamento do **NERDS!**

<http://ame2.russas.ufc.br>

