



# **LINGUAGEM DE PROGRAMAÇÃO ORIENTADA A OBJETOS**

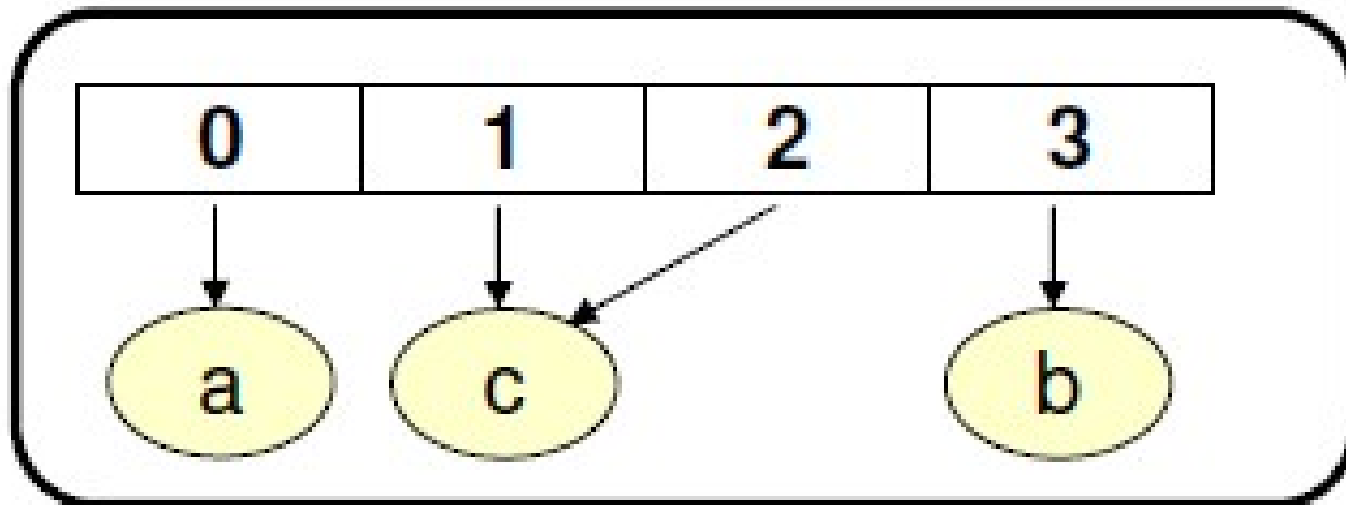
## **LISTA E COLEÇÕES**

**Dr<sup>a</sup>. Alana Moraes**

**[alanamm.prof@gmail.com](mailto:alanamm.prof@gmail.com)**

# PRINCIPAIS COLEÇÕES

- Array Tradicional
  - uma coleção de objetos *indexada por 0, 1, ..., N-1* com tamanho fixo.



# COLEÇÕES

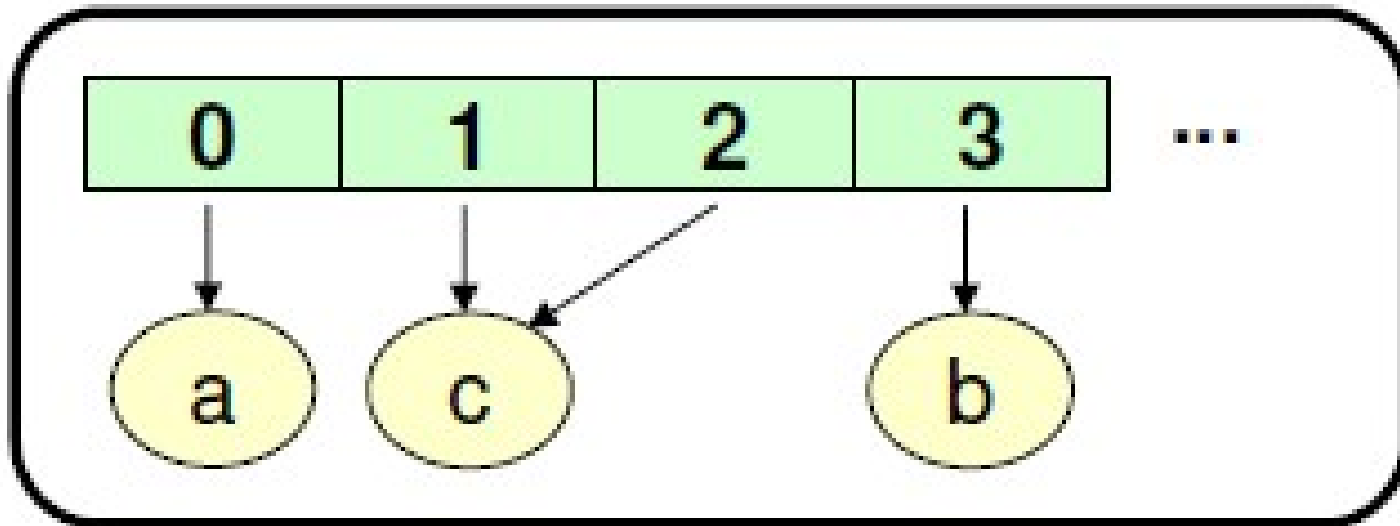
- Coleções permitem que um número arbitrário de objetos seja armazenado.
- Várias aplicações envolvem coleções de objetos:
  - Agendas Pessoais
  - Catálogos de bibliotecas
  - Sistema de registro de alunos
- O número de itens armazenados varia:
  - Itens podem ser adicionados.
  - Itens podem ser excluídos.



# PRINCIPAIS COLEÇÕES

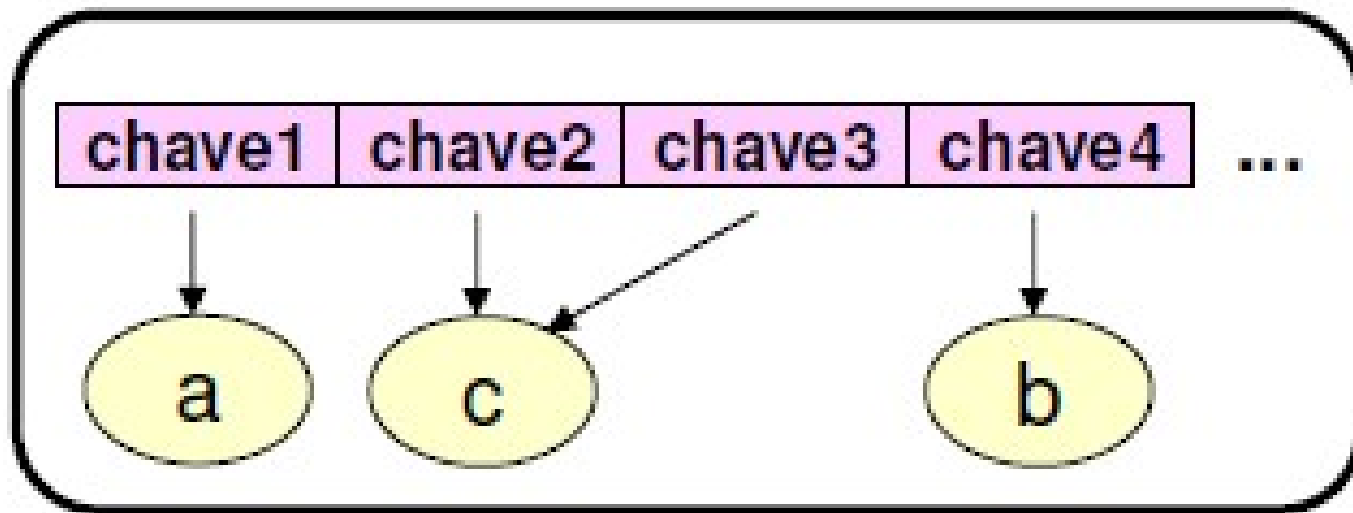
- Classe ArrayList

- uma coleção de objetos *indexada por 0, 1, ..., com tamanho* variável.
- Pode adicionar objetos de diversos tipos



# PRINCIPAIS COLEÇÕES

- Classe HashMap
  - uma coleção de objetos *indexada por <chave>*, com *tamanho* variável.



# ARRAYLIST

- Ações:
  - **Declaração**
  - **Adicionar valores**
  - **Operações**



# ARRAYLIST

- Declaração

- Coleções de objetos do tipo T
- T pode ser uma classe encapsulada (Integer, String, Double, Float, etc) ou uma classe própria (Aluno, Livro, etc).

- ArrayList

**ArrayList<T> nome = new ArrayList<T>(); (Java 1.5)**

Atual:

**ArrayList nome = new ArrayList();**



MAS ANTES PRECISAMOS IMPORTAR A CLASSE  
ARRAYLIST ... ADICIONE!

IMPORT JAVA.UTIL.ARRAYLIST;





# ARRAYLIST

- Ações:
  - **Declaração**
  - **Adicionar valores**
  - **Operações**



# ARRAYLIST

- Adicionar Valores
  - Método add( )

```
lista.add("alguma coisa");  
lista.add(new Integer(10));  
lista.add(new Aluno());
```

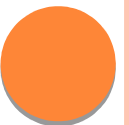
```
lista.add(1, "elemento");
```

**CLASSES ENCAPSULADORAS:**  
**Integer, Long, Double, Boolean, etc...**



# ARRAYLIST

- Ações:
  - **Declaração**
  - **Adicionar valores**
  - **Operações**



# ARRAYLIST

## ○ Operações

- **Incluir** um objeto numa posição específica ou em qualquer posição.
- **Obter** a quantidade de objetos.
- **Percorrer** (iterar) a coleção.
- **Localizar** um determinado objeto pela chave ou referência.
- **Recuperar** um objeto pela posição ou chave ou referência.
- **Atualizar** e **remover** um objeto pela posição ou referência.



# ARRAYLIST - API

- Principais métodos:

`boolean add(Object o)`

`boolean add(int index, Object o)`

`Object get(int index)`

`Object set(int index, Object element)`

`boolean remove(int index)`

`Object remove(Object o)`



# ARRAYLIST - API

- Percorrendo usando o tamanho físico  
**for (int i=0; i<alunos.size(); i++) {  
    System.out.println(alunos.get(i).toString());  
}**
- Percorrendo usando o FOR (Java 1.5):  
**for (Aluno a: alunos) {  
    System.out.println(a.imprima());  
}**



# ARRAYLIST - API

- **boolean add(Object o):**
  - Insere o objeto no final da lista

- **Exemplo:**

```
alunos.add(a1); // a1
```

```
alunos.add(a2); // a1, a2
```

```
alunos.add(a3); // a1, a2, a3
```

```
nomes.add("joao");
```

```
nomes.add("maria"); //joao, maria
```

```
numeros.add(new Double(100) );
```

```
numeros.add(new Integer(200)); //100, 200
```



# ARRAYLIST -API

- **boolean add(int index, Object o):**
  - Insere o objeto na posição (0,...,n-1), deslocando os demais objetos para direita.
- **Exemplo:**
  - `alunos.add(a1); // a1`
  - `alunos.add(0, a2); // a2, a1`
  - `alunos.add(0, a3); // a3, a2, a1`





# ARRAYLIST - API

- **Object get(int index):**

- Retorna a referência do objeto da posição indicada

- **Exemplo:**

`alunos.add(a1); // a1`

`alunos.add(a2); // a1, a2`

`alunos.add(a3); // a1, a2, a3`

`Aluno a = alunos.get(2); // retorna referencia a3`



# ARRAYLIST - API

- **Object set(int index, Object o):**

- Atualiza o objeto da posição indicada

- **Exemplo:**

```
alunos.add(a1); // a1
```

```
alunos.add(a2); // a1, a2
```

```
alunos.set(0,a3); // a3, a2
```

```
numeros.add(100);
```

```
numeros.set(0,101);
```



# ARRAYLIST - API

- **boolean remove(int index):**

- Remove o objeto, fechando o “buraco” automaticamente.

- **Exemplo:**

```
alunos.add(a1);  
alunos.add(a2); // a1, a2  
alunos.remove(0); //a2
```

```
numeros.add(new Integer(100));  
numeros.add(new Integer(200)); //100, 200  
numeros.remove(1); //100
```



# ARRAYLIST - API

- **Object remove(Object o):**

- Remove o objeto fechando o “buraco” automaticamente.

- **Exemplo:**

```
alunos.add(a1);  
alunos.remove(a1);
```

```
Integer i = new Integer(100) ;  
numeros.add(i);  
numeros.remove(i);
```



# ARRAYLIST - API

- **int indexOf( Object o )**

- Localiza um objeto e retorna índice i, tal que:
  - `o.equals(get(i) ) == true`

- **Exemplo:**

`i = alunos.indexOf( umaluno ) ;`

`i = nomes.indexOf( "maria" ) ;`

`i = numeros.indexOf( a1 ) ;`



# ARRAYLIST - API

- **boolean contains( Object o )**
  - If (nomes.contains ("joao")) ...
- Outros métodos
  - **boolean isEmpty( )**  
if (nomes.isEmpty( )) ...
  - **int size( )**  
i = alunos.size( )
  - **void clear( )**
  - **int lastIndexOf (Object elem)**



# EXERCÍCIO RESOLVIDO

- Faça um programa que catalogue uma coleção de Livros e mostre suas informações ao final. Saiba que os livros devem ter nome(String), autor(String) e preço (double).



# EXERCÍCIO

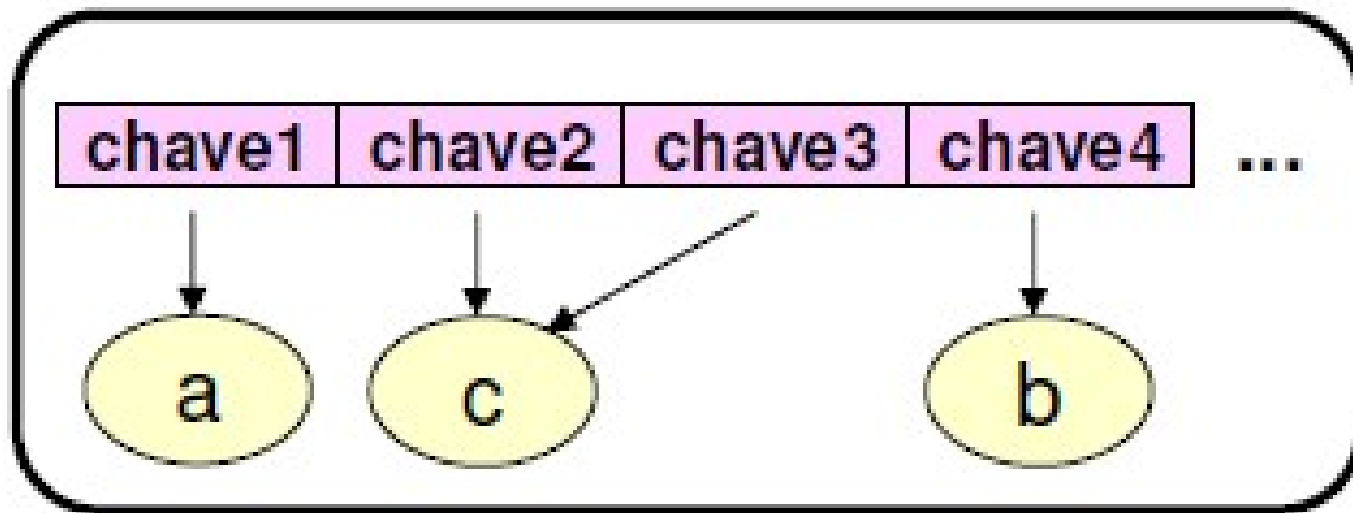
- Crie um projeto que utilize um ArrayList de objetos da **classe Conta** e determine o número da conta de maior saldo. Apresente o número e saldo da Conta com o maior saldo.
- Conta deve ter os atributos: numero (int) e saldo (double), além de seu construtores, gets e sets.
- Para casa: faça o mesmo processo utilizando Array no projeto.





# PRINCIPAIS COLEÇÕES

- Classe HashMap
  - uma coleção de objetos *indexada por <chave>*, com *tamanho* variável.



# HASHMAP

- Ações:
  - **Declaração**
  - **Adicionar valores**
  - **Operações**



# HASHMAP

- Declaração

- Coleções de objetos do tipo T
- T pode ser uma classe encapsulada (Integer, String, Double, Float, etc) ou uma classe própria (Aluno, Livro, etc).

- Java 1.5

```
HashMap<K,V> c = new HashMap<K,V>(  
);
```

- Java 1.8 (Atual)

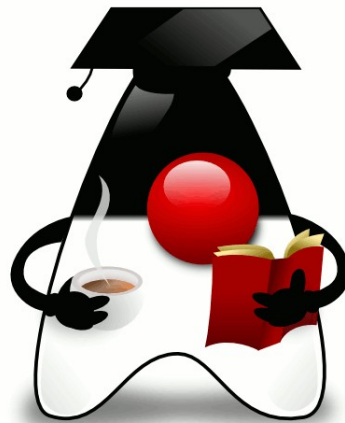
```
HashMap c = new HashMap();
```



NOVAMENTE NÃO ESQUEÇA!

ADICIONE A CLASSE ....

**IMPORT JAVA.UTIL.HASHMAP;**



# HASHMAP

- Ações:
  - **Declaração**
  - **Adicionar valores**
  - **Operações**



# HASHMAP

- Adicionar Valores
  - Método **put()**

```
c.put("chave1", new Integer(1));  
c.put("chave2", new Double(11));
```

```
c.put(new Aluno, new Integer(1));  
c.put(new Float(33), new Double(11));
```



# HASHMAP

- Ações:
  - **Declaração**
  - **Adicionar valores**
  - **Operações**



# HASHMAP

## ○ Operações

- **Incluir** um objeto numa posição específica ou em qualquer posição.
- **Obter** a quantidade de objetos.
- **Percorrer** (iterar) a coleção.
- **Localizar** um determinado objeto pela chave ou referência.
- **Recuperar** um objeto pela posição ou chave ou referência.
- **Atualizar** e **remover** um objeto pela posição ou referência.





# HASHMAP

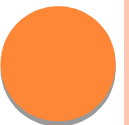
- Object get(Object key)
  - Object valor = listaHash.get("chave1");
- void replace(K key, V value)
- int size()
- boolean containsValue(Object value)



# HASHMAP

- Exercício

- 1) Faça um exemplo de lista em HashMap usando como chave-conteúdo a estrutura nome-data de nascimento dos usuários.



# HASHMAP

## ○ Exercício

- Faça um sistema de matrícula de Alunos.
- Crie um HashMap composto por chave String e conteúdo String para representar a matrícula e nome do aluno.
- Insira, remova e consulte elementos desta lista em uma classe teste.



DÚVIDAS?

