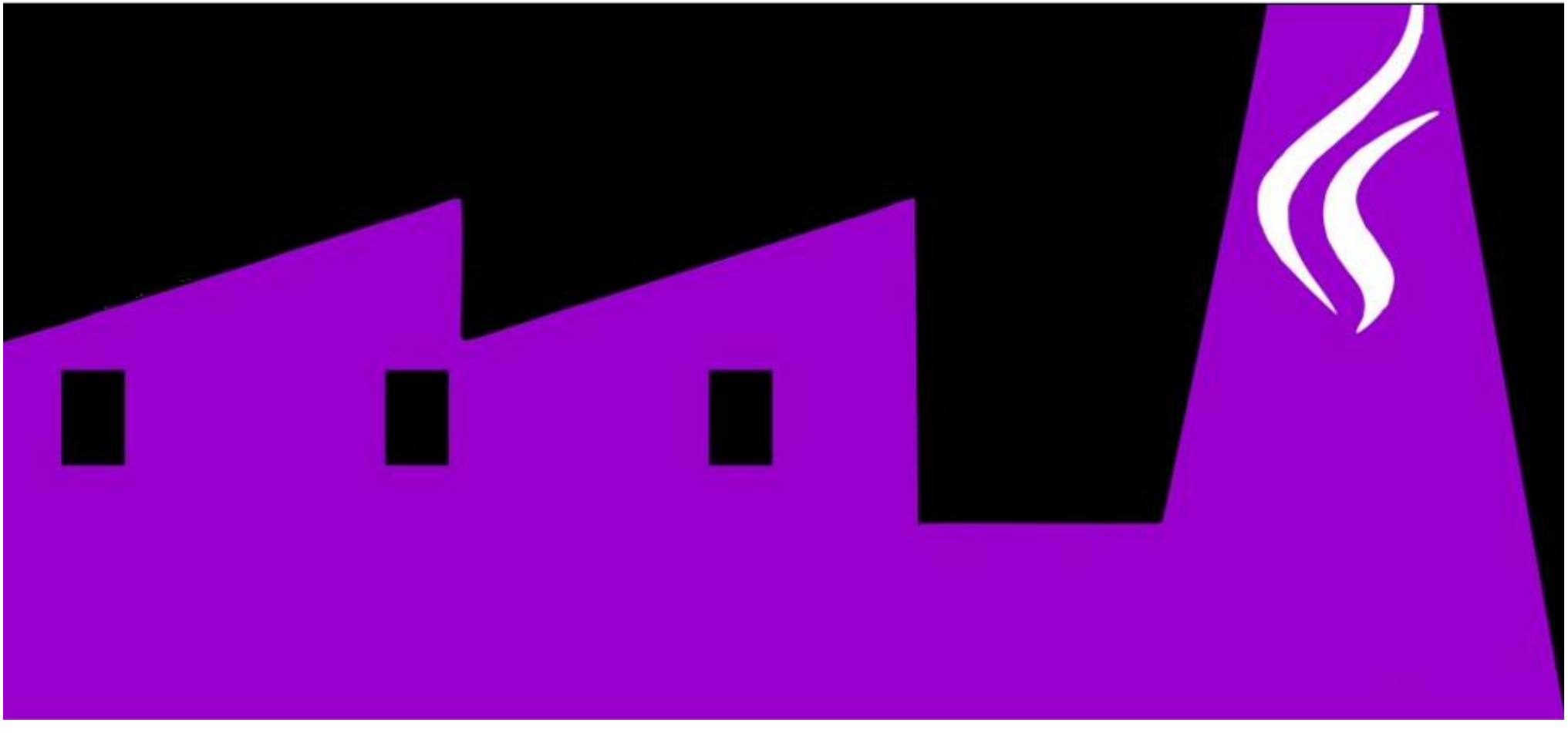
Fábrica de Software

Prof[es.] Ivan L. Süptitz e Evandro Franzen

Arrays e Listas



Arrays (ou vetores)

1 – Declarar
2 - Dimensionar

```
int  idades[ ];
```

Uma array é sempre um objeto, portanto, a variável *idades* é uma referência.

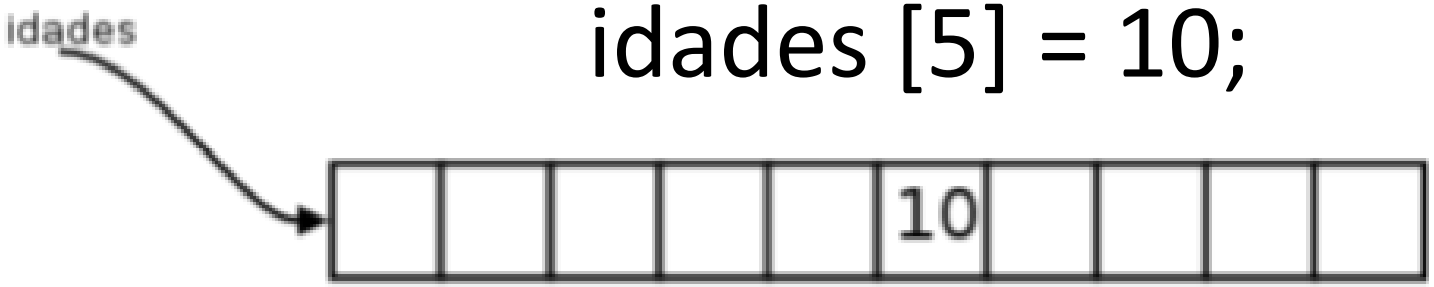
Vamos precisar criar um objeto para poder usar a array.

Como criamos o objeto-array?

```
idades = new int [10];
```

Arrays

idades [5] = 10;



No **Java**, os **índices** do array vão de **0 a n-1**,onde **n** é o **tamanho** dado no momento em que você criou o **array**.

OBS: Se você tentar acessar uma posição fora desse alcance, um erro ocorrerá durante a execução.

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 10
        at ArrayIndexOutOfBoundsExceptionTeste.main(ArrayIndexOutOfBoundsExcepti
onTeste.java:5)
```

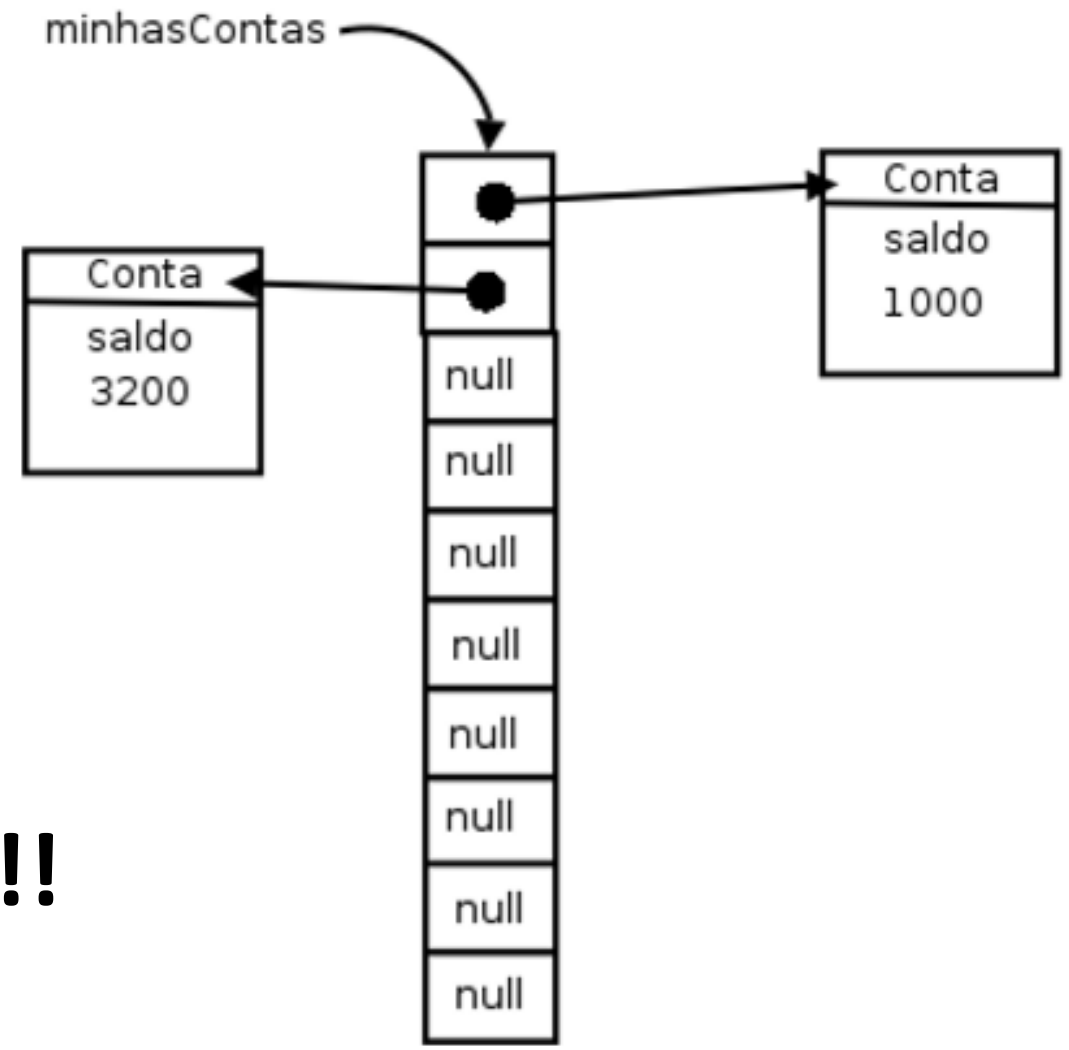
Arrays

```
int x[  ] = new int [6];
int[ ] x = new int [6]; //tanto faz onde vão os colchetes
int y[  ] = new int [  ] { 1, 8, 23, 28, 44, 58};
int z[  ] = { 1, 8, 23, 28, 44, 58};

double m[  ] = new double[6];
float  n[  ] = new float[6];
char  o[  ] = new char[6];
String p[  ] = new String[6];

Conta minhasContas[ ] = new Conta [10];
```

Como ficaria com outros tipos?

Arrays

```
Conta minhasContas [ ] = new Conta [10];
Conta c = new Conta();
c.setSaldo(1000);
minhasContas [0] = c;
c = new Conta();
c.setSaldo(3200);
minhasContas [1] = c;
```



Arrays não podem mudar de tamanho!!

Arrays

```
for (int i = 0; i < idades.length; i++) {
    System.out.println(idades[ i ]);
}

for (int i = 0; i < minhasContas.length; i++) {
    System.out.println(minhasContas[ i ].getSaldo());
}
```

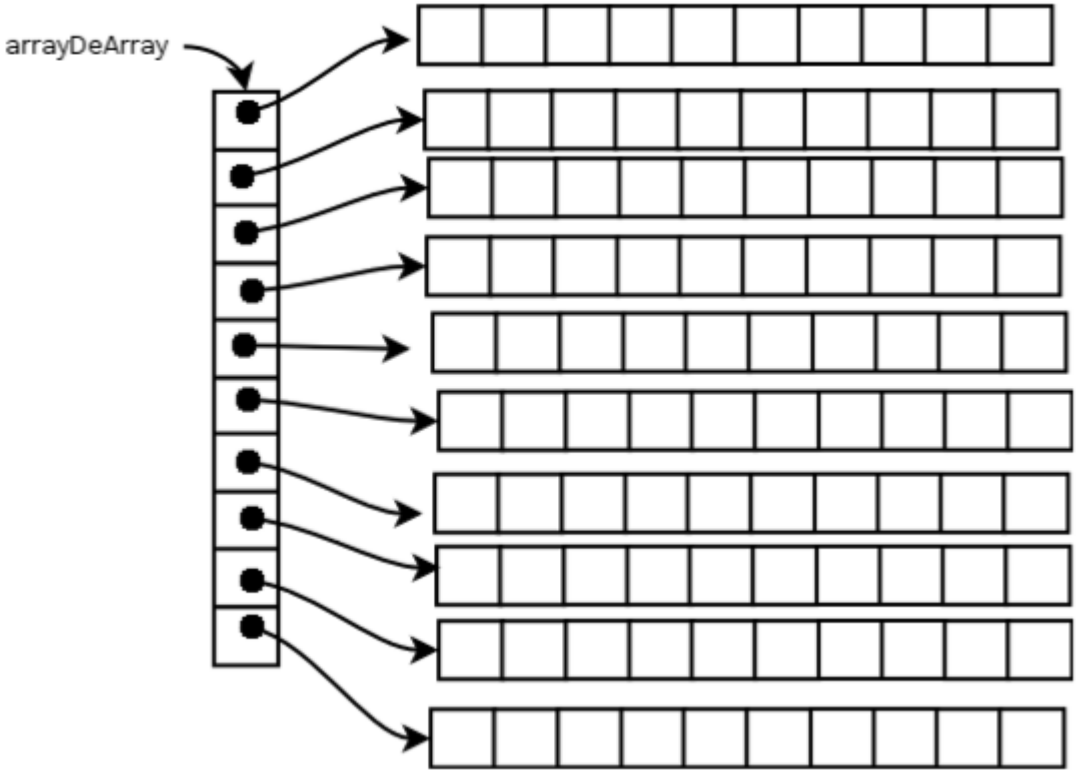
Arrays Bidimensionais

Arrays podem ter **mais** de uma **dimensão** ou seja, podemos criar arrays de arrays!

```
Contas  minhasContas [ ] [ ] = new Contas[10] [10];
```

Em vez de termos uma array de 10 contas, podemos ter uma array de 10 por 10 contas e você pode acessar a conta na posição da linha x e coluna y.

Pode ser chamada também de matriz.



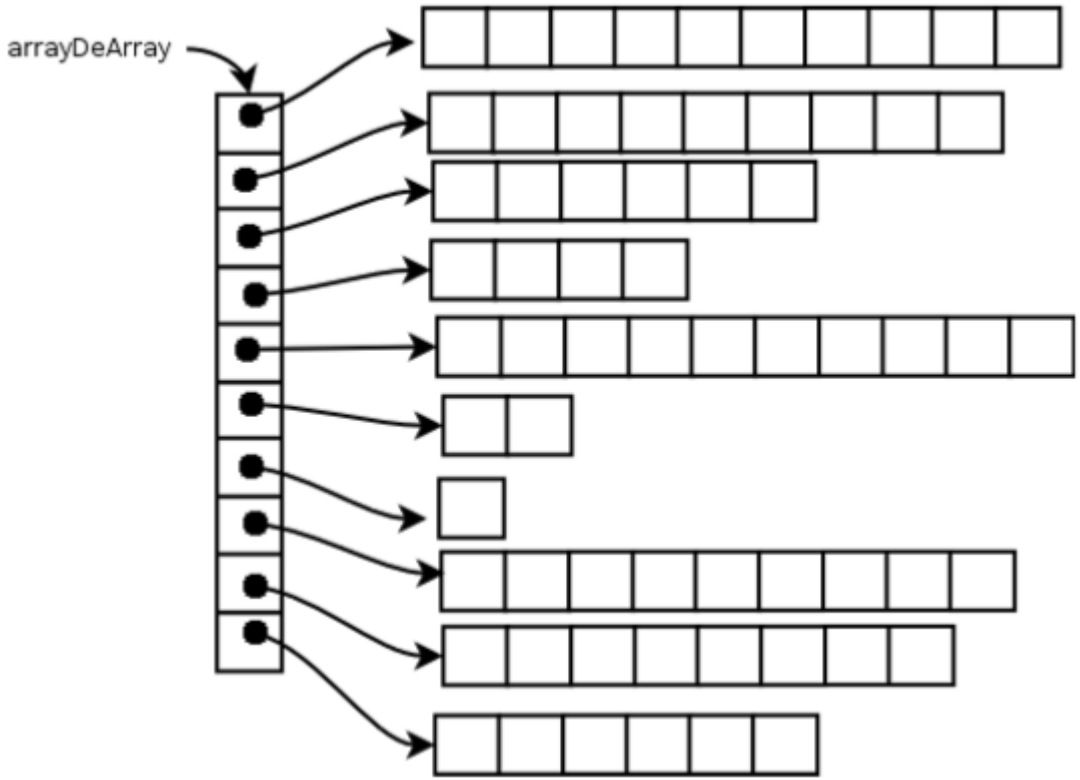
Arrays Multidimensionais

```
double notas [ ][ ] = { {8.0, 7.5, 8.5 }, {8.9, 9.0, 8.6 }, {7.1, 7.0, 7.6 } };
```

No caso acima é uma matriz 3x3!

Uma **array bidimensional** **não** precisa ser **retangular**, isto é, cada linha pode ter um número diferente de colunas

```
double notas [ ][ ] = {
        { 8.0, 7.5, 8.5 },
        { 8.9 },
        { 7.1, 7.6 } };
```

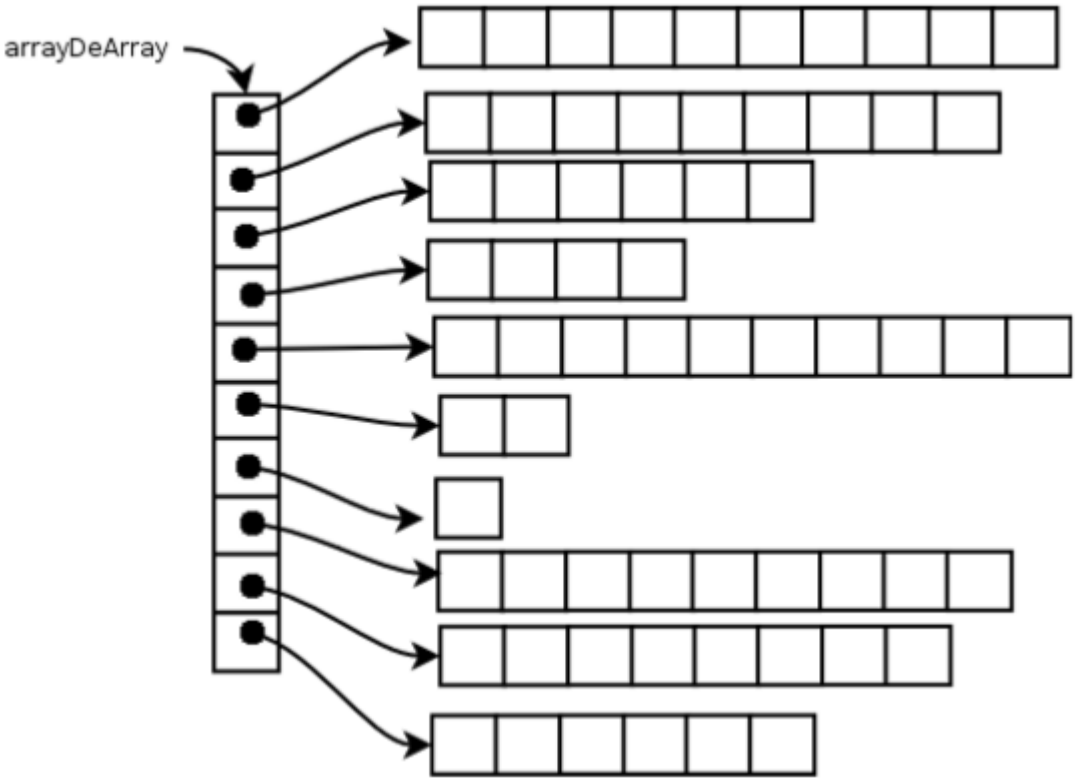


Arrays Multidimensionais

```
double notas [ ][ ] = { { 8.0, 7.5, 8.5 }, { 8.9 }, { 7.1, 7.6 } };
```

Ou:

```
double notas[ ][ ] = new double[3][ ];
notas[0] = new double[3];
notas[1] = new double[1];
notas[2] = new double[2];
```



Arrays Multidimensionais

```
double notas [ ] [ ] [ ]
double [ ] [ ] notas [ ]
double [ ] notas [ ] [ ]
```

```
public class Desafio
{
   public static void main(String[ ] args)
   {
      int [ ] a = new int [2];
      int [ ] b [ ]= new int [2][2];
      int [ ][ ] c [ ]= new int [2][2][2];
      a [0] = 10;
      a [1] = 15;
      b [0] = a;
      c [0] = b;
      System.out.println(c[0][0][1]);
   }
}
```

Listas: java.util.List

Um primeiro recurso que a API de Collections traz são listas. Uma lista é uma coleção que permite elementos duplicados e mantém uma ordenação específica entre os elementos.

A implementação mais utilizada da interface List é a ArrayList, que trabalha com um array interno para gerar uma lista.

Listas: java.util.List

Para criar um ArrayList, basta chamar o construtor:

```
ArrayList lista = new ArrayList();
```

Para criar uma lista de nomes(String), podemos fazer:

```
ArrayList lista = new ArrayList();
lista.add("Manoel");
lista.add("Joaquim");
lista.add("Maria");
```

Listas genéricas de objetos

```
Conta c1 = new Conta();
c1.setSaldo(100);
Conta c2 = new Conta();
c2.setSaldo(200);
Conta c3 = new Conta();
c3.setSaldo(300);

ArrayList minhasContas = new ArrayList();
minhasContas.add(c1);
minhasContas.add(c3);
minhasContas.add(c2);

for (int i = 0; i < minhasContas.size(); i++) {
    Conta cc = (Conta) minhasContas.get(i);
    System.out.println(cc.getSaldo());
}
```

É possível?? SIM!

```
minhasContas.add("oi");
```

Lista de tipos específicos

As listas de um determinado tipo de objetos !

```
ArrayList<Conta> minhasContas = new ArrayList<Conta>();
minhasContas.add(c1);
minhasContas.add(c3);
minhasContas.add(c2);
```

Repare no uso de um parâmetro ao lado de List e ArrayList: ele indica que nossa lista foi criada para trabalhar exclusivamente com objetos do tipo Conta.

Isso nos traz uma segurança em tempo de compilação:

```
minhasContas.add("oi"); // isso não compila mais!!
```

```
for (int i = 0; i < minhasContas.size(); i++) {
    System.out.println(minhasContas[ i ].getSaldo());  //sem casting
}
```

Alguns métodos

```
ArrayList<Integer> val = new ArrayList<Integer>();
val.add(9);
val.add(5);
val.add(44);
val.add(3);

val.remove(2);

Collections.sort(val);
Collections.reverse(val);

int max = Collections.max(val);
int min = Collections.min(val);
System.out.println(max - min);
```

```
for (int i = 0; i < val.size(); i++)
   System.out.println(val.get(i));

//ou

for (Integer i: val)
     System.out.println(i);
```

Exercícios - Arrays

6.1. Crie um programa que possua uma classe Funcionário com dois atributos: nome e salário.
Faça um sistema que guarde e controle uma listagem de Funcionários. O sistema deve permitir Adicionar, Mostrar, Editar e Excluir registros. Vamos testar com Array normal, ArrayList, Vector e LinkedList

Exercícios - Arrays

6.2. Crie um programa que faz a leitura de uma array bidimensional (matriz 4x6) de inteiros. Após, multiplicar seu elementos pelo maior número do array. Por fim, imprimir o array original e o array resultado.

Exercícios – ArrayList

6.3. (ENTREGAR NO VIRTUAL) Utilizando ArrayList, crie um projeto Agenda que possa conter entradas de objetos tipo Registro de Agenda.

- Nome: nome da pessoa
- Endereço: endereço da pessoa
- Telefone: número de telefone da pessoa
- Email: endereço eletrônico da pessoa

Devem ser oferecidos os seguintes métodos para a agenda:

- Adicionar registro; Excluir registro (pelo nome); Visualizar registros; Modificar registro.

- Dica, se você tiver problema para excluir um registro pelo nome (pois precisa percorrer a lista e pode aparecer 2 vezes) utilize o seguinte código para percorrer:

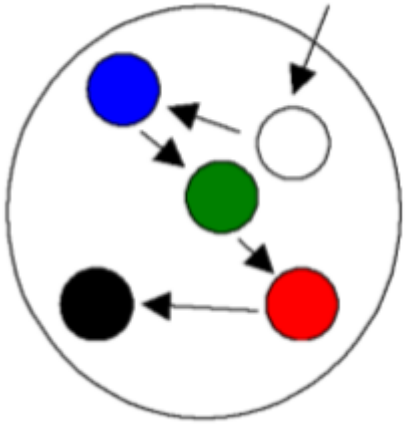
```
Iterator iter1 = al.iterator();
        while(iter1.hasNext()){}
```

CURIOSIDADE: Outros tipos de Collections - **Set**

Um Conjunto (Set) funciona de forma análoga aos conjuntos da matemática, ele é uma coleção que não permite elementos duplicado

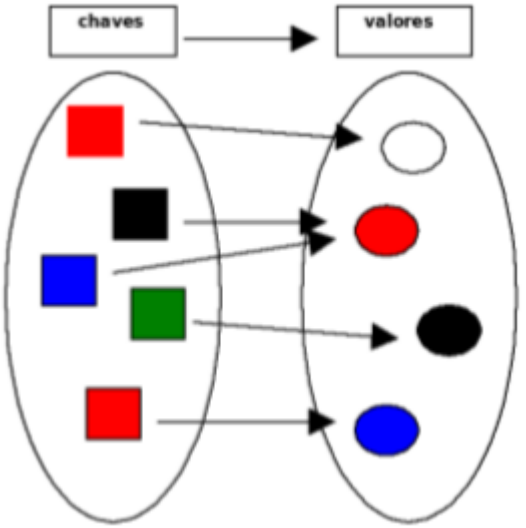
A ordem em que os elementos são armazenados pode não ser a ordem na qual eles foram inseridos no conjunto.

```
Set<String> cargos = new HashSet<>();
cargos.add("Gerente");
cargos.add("Diretor");
cargos.add("Presidente");
cargos.add("Secretária");
cargos.add("Funcionário");
cargos.add("Diretor"); // repetido!

Iterator<String> i = cargos.iterator();
while (i.hasNext()) { // recebe a palavra
    String palavra = i.next();
    System.out.println(palavra);
 }
```



CURIOSIDADE: Outros tipos de Collections - **Map**

Um Mapa é composto por um conjunto de associações entre um objeto chave a um objeto valor




```
Map<String, Conta> mapaDeContas = new HashMap<>();
// adiciona duas chaves e seus respectivos valores
mapaDeContas.put("diretor", c1);
mapaDeContas.put("gerente", c2);

// qual a conta do diretor? (sem casting!)
Conta contaDoDiretor = mapaDeContas.get("diretor");
System.out.println(contaDoDiretor.getSaldo());
```

Um mapa é muito usado para “indexar” objetos de acordo com determinado critério, para podermos buscar objetos rapidamente.

```
for (Entry<String, Conta> t : mapaDeContas.entrySet()) {
        String key = t.getKey();
        Conta value = mapaDeContas.get(key);
        System.out.println(key + ": " + value.getSaldo());
 }
```