

INSTITUTO MAUÁ DE TECNOLOGIA



# Linguagens I

Wrapper e ArrayList

Profº. Tiago Sanches da Silva

# **Wrapper, Autoboxing e Unboxing**

# Wrapper, Autoboxing e Unboxing

Wrappers vem do verbo inglês “wrap” que significa envolver.

Tem como principal função “envolver coisas” adicionando funcionalidades à ela.

O Java possui oito wrappers para tipos primitivos que adicionam a funcionalidade de tratar tipos primitivos como classes.

**Integer, Double, Float, Byte, Character, Boolean, Short e Long**

# Wrapper, Autoboxing e Unboxing

Vamos olhar um pouco de código!

# Wrapper, Autoboxing e Unboxing

Lista dos Wrappers mais comuns no Java:

	Tipo primitivo	Classe Wrapper	Subclasse
Lógico	Boolean	Boolean	Object
Caractere	Char	Character	
Integral	byte	Byte	Number
	short	Short	
	int	Integer	
	long	Long	
Ponto Flutuante	float	Float	
	Double	Double	

# Wrapper, Autoboxing e Unboxing

O Java (a partir da versão 5) é inteligente o suficiente para criar ou desfazer wrappers de tipo primitivo automaticamente (Autoboxing).

## Autoboxing:

```
Integer inum = 3; //Assigning int to Integer: Autoboxing  
Long lnum = 32L; //Assigning long to Long: Autoboxing
```

```
ArrayList<Integer> arrayList = new ArrayList<Integer>();  
arrayList.add(11); //Autoboxing - int primitive to Integer  
arrayList.add(22); //Autoboxing
```

# Wrapper, Autoboxing e Unboxing

## Unboxing:

```
Integer inum = new Integer(5);  
int num = inum; //unboxing object to primitive conversion
```


```
class UnboxingExample1  
{  
    public static void myMethod(int num){  
        System.out.println(num);  
    }  
    public static void main(String[] args) {  
  
        Integer inum = new Integer(100);  
  
        /* passed Integer wrapper class object, it  
         * would be converted to int primitive type  
         * at Runtime  
         */  
        myMethod(inum);  
    }  
}
```

# Quando Autoboxing e Unboxing são usados?

**Autoboxing** é aplicada pelo compilador do Java nas seguintes condições:

- Quando um valor primitivo é passado como um parâmetro para um método que espera um objeto da classe Wrapper correspondente.
- Quando um valor primitivo é atribuído a uma variável da classe Wrapper correspondente.

**Unboxing** é aplicada pelo compilador do Java nas seguintes condições:

- Quando um objeto é passado como um parâmetro para um método que espera um valor primitivo correspondente.
  - Quando um objeto é atribuído a uma variável do tipo primitivo correspondente.
- 



# Arrays

# Arrays

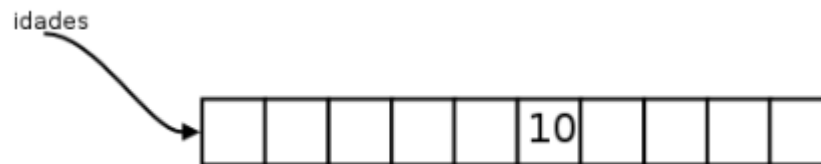
O `int[]` é um tipo. Um **array** é sempre um objeto, portanto, a variável `idades` é uma referência. Vamos precisar criar um objeto para poder usar a array. Como criamos o objeto-array?

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

O que fizemos foi criar um array de `int` de 10 posições e atribuir o endereço no qual ela foi criada. Podemos ainda acessar as posições do array:

```
idades[5] = 10;
```

Resultando em:



# Arrays

Trecho completo:

```
int[] idades;  
  
idades = new int[10];  
  
idades[5] = 10;
```

O código acima altera a sexta posição do array. 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. Se você tentar acessar uma posição fora desse alcance, um erro ocorrerá durante a execução.

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException:
```

# Array de objetos

# Array de objetos

Mesmo que seja comum ouvirmos a expressão “**array de objetos**”, temos que ter em mente que na verdade possuímos um **array** de referências para os objetos, e portanto eles precisam ser criados individualmente.

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

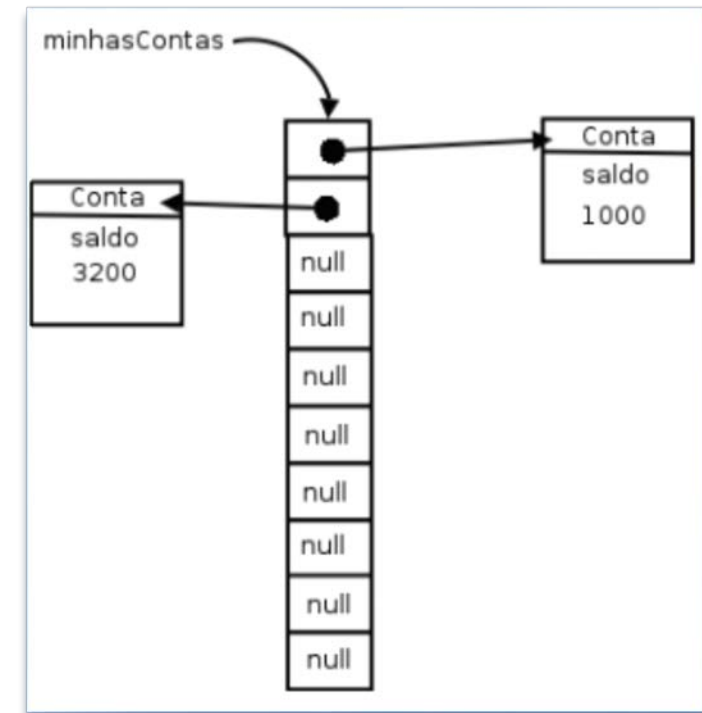
Quantas objetos contas foram criadas aqui?

Na verdade, nenhuma. Foram criados 10 espaços que você pode utilizar para guardar uma referência a uma Conta. Por enquanto, eles se referenciam para lugar nenhum (null).

# Array de objetos

Você deve popular seu array antes.

```
Conta contaNova = new Conta();  
contaNova.saldo = 1000.0;  
minhasContas[0] = contaNova;
```

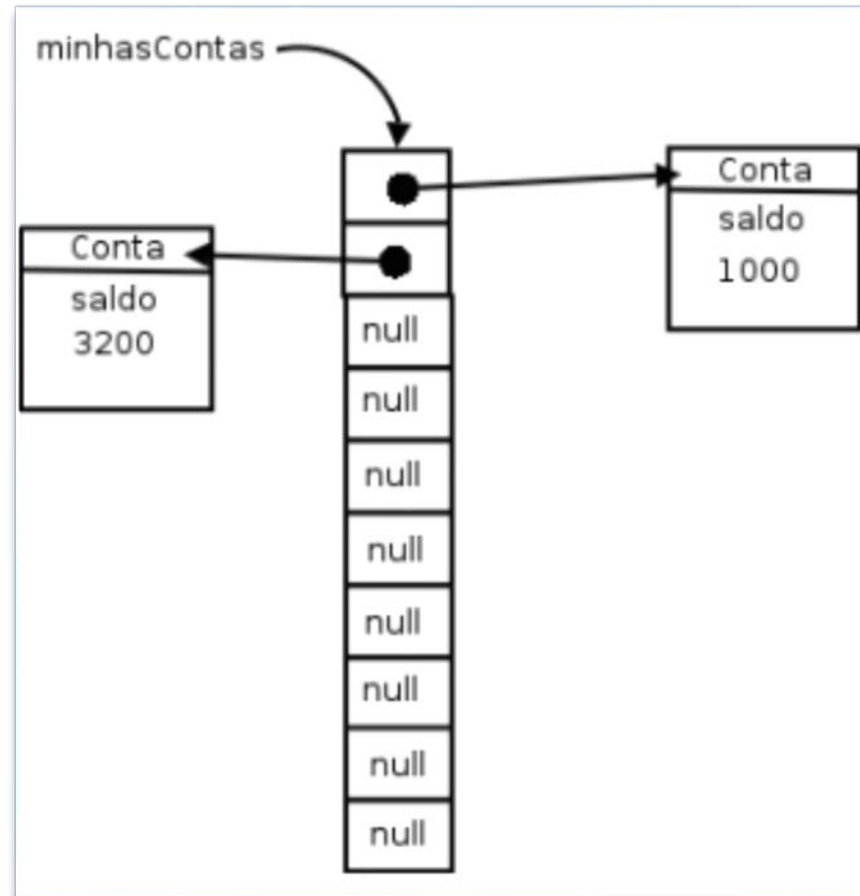


Ou você pode fazer assim:

```
minhasContas[1] = new Conta();  
minhasContas[1].saldo = 3200.0;
```

# Array de objetos

Uma **array** de tipos primitivos guarda valores, uma array de objetos guarda **referências**.



# Percorrendo um array

Percorrer um **array** é muito simples, basta utilizar alguma estrutura de repetição. Sugestões?

```
public static void main(String args[]) {  
    int[] idades = new int[10];  
    for (int i = 0; i < 10; i++) {  
        idades[i] = i * 10;  
    }  
    for (int i = 0; i < 10; i++) {  
        System.out.println(idades[i]);  
    }  
}
```



# **Um pouco mais de arrays**

# Array como parâmetros

```
private static void printArray(int[] arr)
{
    for(int i = 0; i < arr.length; i++)
        System.out.print(arr[i] + " ");
    System.out.println();
}
```

# Array como retorno de função

```
public static int[] getIntegers(int number) {  
    System.out.println("Enter " + number + " integer values.\r");  
    int[] values = new int[number];  
  
    for(int i=0; i<values.length; i++) {  
        values[i] = scanner.nextInt();  
    }  
  
    return values;  
}
```

# Redimensionando Array

E se o número de objetos que preciso armazenar aumentar?

Código:  
RedimensionandoArray.java

# **Classe ArrayList**

# ArrayList

O Java, por padrão, possui uma série de recursos prontos (APIs) para que possamos tratar de estrutura de dados, também chamados de coleções (collections).

Podemos dizer que ArrayList é uma classe para coleções.

Você pode criar seus objetos - através de uma classe - e agrupá-los através de ArrayList e realizar, nessa coleção, várias operações, como: **adicionar** e **retirar** elementos, ordená-los, procurar por um elemento específico, apagar um elemento específico, limpar o ArrayList dentre outras possibilidades.

# ArrayList

O **List** é uma interface e o **ArrayList** é a classe que a implementa.


<https://docs.oracle.com/javase/8/docs/api/java/util/List.html>

<https://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html>

# Exercício 1

Crie uma classe `ListaDeCompras` em que seja possível, adicionar itens, remover por nome, listar, procurar, modificar (através do index).

Crie na classe principal um menu para gerenciar a lista de comprar criada na classe `ListaDeComprar`. Ou utilize o JavaFX (Apenas para exibição e entrada de dados).





# Exercício 2

Sua classe principal é um celular.

Crie uma classe ListaDeContatos, em que cada contato possui minimamente nome e número de telefone (crie uma classe para isso).

Crie menu para gerenciar a lista de contatos na classe principal.



Perguntas?

# Referências

- Deitel
  - DevMedia
  - JavatPoint: [www.javatpoint.com](http://www.javatpoint.com)
  - GUJ
  - Oracle
  - Tim Buchalka
- 