

Porém Sião diz: Já me desamparou o Senhor, e o meu Senhor se esqueceu de mim.

Porventura pode uma mulher esquecer-se tanto de seu filho que cria, que não se compadeça dele, do filho do seu ventre? Mas ainda que esta se esquecesse dele, contudo eu não me esquecerei de ti.

Isaías 49: 14–15

# Curso de Especialização em Tecnologia Java

## LINGUAGEM DE PROGRAMAÇÃO JAVA I

- ▶ Prof: José Antonio Gonçalves
- ▶ [zag655@gmail.com](mailto:zag655@gmail.com)
- ▶ Ao me enviar um e-Mail coloque o “Assunto” começando: “pós2013\_2+seu nome”

## Nestes Slides:

- **Orientação a Objetos em Java:**

Aplicação do conceito sobre Coleções de Objetos;

## Experimento:

### Aplicação do conceito de Collection: List/ArrayList

- Embora tenhamos visto na aula anterior sobre a existência de vários tipos de Collections, por uma questão prática associada ao tempo, neste experimento iremos utilizar List/ArrayList.

## Contextualização

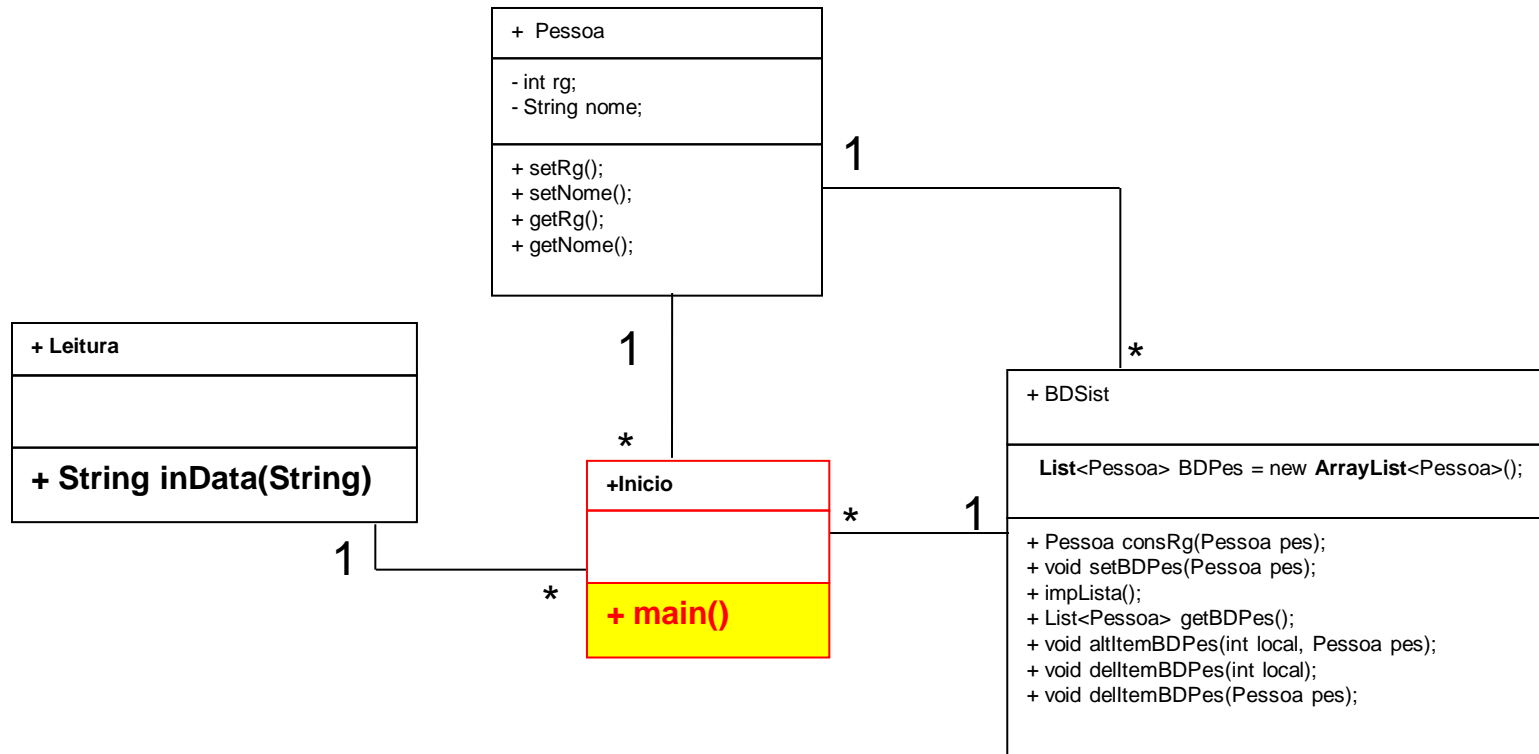
– De acordo com os exemplos anteriores que utilizam a gestão de pessoas como objeto de estudo e aplicação dos conceitos e técnicas, vamos imaginar que precise armazenar um conjunto de pessoas. Considere também que em algum momento deva manipular estes objetos (do tipo pessoa) podendo consultar, alterar os valores de seus atributos e ainda apagar estes objetos.

## Contextualização

- Para isso vamos construir um array dinâmico que nos ofereça as condições mínimas para manipular estes dados.
- Para esta situação vamos utilizar a classe **List**, porém vamos instanciá-la de forma que se comporte como um **ArrayList**.

# Problema

– Observe o diagrama de classes a seguir. Perceba que, embora fosse possível agregar o array dinâmico na própria classe Pessoa, optou-se por criar uma classe especialista para gerir os dados:



# Código da classe BDSist (1 / 3)

```
import java.util.ArrayList;  
import java.util.List;
```

```
public class BDSist { //inicio da classe
```

```
    private List<Pessoa> BDPes = new ArrayList<Pessoa>();
```



# Código da classe BDSist (2 / 3)

```
public int consRg(Pessoa pes, int endereco){
    endereco = -1;
    for(int i=0;i<BDPes.size();i++){
        if(BDPes.get(i).getRg()==pes.getRg()){
            return i;
        }
    }
    return endereco;
}

public Pessoa consRg(Pessoa pes){
    for(int i=0;i<BDPes.size();i++){
        if(BDPes.get(i).getRg()==pes.getRg()){
            return BDPes.get(i);
        }
    }
    return null;
}

public void setBDPes(Pessoa pes){
    if(consRg(pes)==null) {
        BDPes.add(pes);
    }
}

public void impLista(){
    for(int i=0;i<BDPes.size();i++){
        System.out.println("RG.: "+BDPes.get(i).getRg());
        System.out.println("Nome: "+BDPes.get(i).getNome());
    }
}
```

# Código da classe BDSist (3 / 3)

```
public List<Pessoa> getBDPes(){
    return BDPes; //retornando a lista
}

public void altItemBDPes(int local, Pessoa pes){
    BDPes.set(local, pes); //alterando obj da lista
}

public void delItemBDPes(int local){
    BDPes.remove(local); //apagando obj da lista pelo índice (inteiro)
}

public void delItemBDPes(Pessoa pes){
    BDPes.remove(pes); //apagando obj da lista pelo obj
}
} // fim da classe
```

# Definições dos métodos da classe BDSist

– Neste caso, a classe BDSist foi construída a fim de servir como um repositório de dados de pessoas. Ela conterá 7 métodos públicos, sendo:

1. Pessoa consRg(Pessoa pes);
2. void setBDPes(Pessoa pes);
3. impLista();
4. List<Pessoa> getBDPes();
5. void altItemBDPes(int local, Pessoa pes);
6. void delItemBDPes(int local);
7. void delItemBDPes(Pessoa pes);

## Classe BDSist – Método 1: Pessoa consRg(Pessoa pes)

**Funcionalidade:** permitirá consultar pessoas contidas no array por meio do valor do RG;

O método consRg recebe um objeto do tipo Pessoa por parâmetro, utiliza este objeto para fazer uma busca no array. COMPARA OS RG's da pessoa passada por parâmetro com todas as pessoas que estão armazenadas no array. CASO ENCONTRE alguma pessoa com o mesmo RG, retornará a pessoa encontrada, SE NÃO retornará NULL (um objeto pessoa "vazio").

Métodos nativos e seus usos (BDPes é o nome do array):

BDPes.size() -> **size()**: retorna o total de endereços utilizados do array, logo o tamanho do array;

BDPes.get(i) -> **get(i)**: retorna o objeto existente no array na posição indicada por "i".

BDPes.get(i).getRg() -> **get(i).getRg()**: retorna o objeto existente no endereço "i" (**get(i)**) e deste retorna o RG (**getRg()**).

# Classe BDSist – Método 1: Pessoa consRg(Pessoa pes)

## Código:

```
public Pessoa consRg(Pessoa pes){  
    for(int i=0;i<BDPes.size();i++){  
        if(BDPes.get(i).getRg() == pes.getRg()){  
            return BDPes.get(i);  
        }  
    }  
    return null;  
}
```

Percorrer todo  
o array

Compara **SE**:  
- RG da pessoa do  
array

é igual

- **SE FOR IGUAL**  
retorna a pessoa do  
array

- Ao da pessoa  
passada por parâmetro

- se **NÃO** for igual  
retorna objeto vazio  
NULL

# Classe BDSist – Método 1: Pessoa consRg(Pessoa pes) usando **FOR** **extendido**

## Código:

```
public Pessoa consRg(Pessoa pes){  
    for(Pessoa p : BDPes){  
        if(p.equals(pes)){  
            return p;  
        }  
    }  
    return null;  
}
```

Percorrer todo o array: para cada objeto Pessoa existente no array BDPes gere um novo "p"

Compara **SE**:  
- O objeto "p"

é igual

- **SE FOR IGUAL**  
retorna a pessoa do array

- Ao objeto Pessoa passado por parâmetro

- se **NÃO** for igual  
retorna objeto vazio  
NULL

## Classe BDSist – Método 2: void setBDPes(Pessoa pes)

**Funcionalidade:** permitirá inserir uma pessoa no array;

O método setBDPes recebe um objeto do tipo Pessoa por parâmetro, passado pelo usuário. Porém antes de inserir o objeto no array, utiliza o método **consRg** para saber se já existe uma pessoa com o mesmo RG armazenada. Para isto também passa o objeto que recebeu por parâmetro para o método **consRg(Pessoa)**, somente se o retorno for nulo, isto é, não existe pessoa com o mesmo RG, o objeto enviado (pelo usuário) como parâmetro será adicionado no array.

Métodos nativos e seus usos (**BDPes** é o nome do array):

**BDPes.add(Pessoa) -> add(Pessoa):** adiciona um novo objeto ao array e com ele um novo endereço.

## Classe BDSist – Método 2: void setBDPes(Pessoa pes)

### Código:

```
public void setBDPes(Pessoa pes){  
    if(consRg(pes) == null){  
        BDPes.add(pes);  
    }  
}
```

Compara **SE**:  
- O retorno do método  
consRg(pes) com o parâmetro  
da Pessoa que se deseja  
inserir

é igual

- **SE FOR NULO**  
armazena o objeto no  
array

- a Nulo (objeto Pessoa  
vazio)

### OBS.:

- O método **ADD** adiciona um novo objeto ao array e com ele um novo endereço.
- A lógica neste caso, indica que para se inserir uma pessoa no array, não deve existir outra com o mesmo RG. Logo só conseguirá inserir se o retorno do consRg for nulo, caso contrário é sinal que **já existe** alguém com o mesmo RG armazenado no array.



## Classe BDSist – Método 3: void impLista()

**Funcionalidade:** imprime todos os valores de todos objetos do array;

# Classe BDSist – Método 3: void impLista()

## Código:

```
public void impLista(){  
  
    for(int i=0;i<BDPes.size();i++){  
        System.out.println("RG...: "+BDPes.get(i).getRg());  
        System.out.println("Nome..: "+BDPes.get(i).getNome());  
    }  
}
```

Percorrer todo  
o array

Recupera o objeto da  
posição "i"

Recupera o RG/NOME do  
objeto retornado por "get(i)"

**OBS.:** Imprime todos os valores de cada objeto do array

## Classe BDSist – Método 4: List<Pessoa> getBDPes();

**Funcionalidade:** retorna todo o array;

## Classe BDSist – Método 4: List<Pessoa> getBDPes()

Aqui, na assinatura do método, definiu-se que este iria retornar um **List de Pessoas**

**Código:**

```
public List<Pessoa> getBDPes(){  
    return BDPes;  
}
```

Retorna todo o List BDPes

**OBS.:** Imprime todos os valores de cada objeto do array

## Classe BDSist – Método 5: void altItemBDPes(int local, Pessoa pes);

**Funcionalidade:** altera objeto do array de pessoas. Neste caso dever o ser passados dois parâmetros:

**int local** = endereço do array onde está o objeto a ser alterado;

**Pessoa pes** = novo objeto, já com novos valores, que irá substituir o objeto existente no "local" ;

BDPes.set -> **set(local, pes)**: método que atualiza o array no endereço definido (local) pelo novo objeto Pessoa (pes).

# Classe BDSist – Método 5: void altItemBDPes(int local, Pessoa pes);

## Código:

```
public void altItemBDPes(int local, Pessoa pes){  
    BDPes.set(local, pes);  
}
```

altera o objeto do array que foi indicado pelo "local" (endereço do array) pelo novo objeto "pes", informado por parâmetro.

## Classe BDSist – Método 6: void delItemBDPes(int local);

**Funcionalidade:** permite apagar um objeto do array informando o endereço do array onde se encontra:

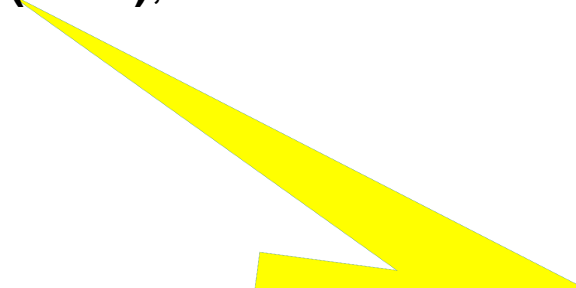
**int local** = endereço do array onde está o objeto a ser apagado;

BDPes.remove -> **remove(local)**: método que remove (apaga) um objeto do array no endereço definido (local) .

## Classe BDSist – Método 6: void delItemBDPes(int local);

### Código:

```
public void delItemBDPes(int local){  
    BDPes.remove(local);  
}
```



apaga objeto do array que foi indicado pelo "local" (endereço do array).



## Classe BDSist – Método 7: void delItemBDPes(Pessoa pes);

**Funcionalidade:** permite apagar um objeto na sua primeira ocorrência no array de acordo com o objeto informando com parâmetro:

Pessoa pes = objeto do array a ser apagado;

BDPes.remove -> **remove(pes)**: método que remove (apaga) um objeto do array de acordo com o objeto definido como parâmetro (pes) .

## Classe BDSist – Método 7: void delItemBDPes(Pessoa pes);

### Código:

```
public void delItemBDPes(int local){  
    BDPes.remove(pes);  
}
```

apaga a primeira ocorrência do objeto no array que foi indicado pelo parâmetro "pes".

## Classes: Leitura e Pessoa

As classes **Leitura** e **Pessoa** já forma estudadas em outras oportunidades, logo, aqui só serão apresentadas mas **não explicadas**.

# Classe Pessoa

## Código:

```
public class Pessoa{
    private int rg;
    private String nome;

    public int getRg(){
        return rg;
    }

    public void setRg(int rg){
        this.rg = rg;
    }

    public String getNome(){
        return nome;
    }

    public void setNome(String nome){
        this.nome = nome;
    }
}
```

# Classe Leitura

## Código:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class Leitura{
    public String inData(String label){
        InputStreamReader teclado = new InputStreamReader(System.in);
        BufferedReader memoria = new BufferedReader(teclado);
        System.out.println(label);
        String s = "";
        try{
            s = memoria.readLine();
        }
        catch(IOException e){
            System.out.println("Erro de entrada");
        }
        return s;
    }
}
```

## Classes: Início

A classe **Início** foi construída a fim de testar todo o mecanismo do array dinâmico.

Para isso fez-se menus que permitisse executar a mesma ação várias vezes, como a possibilidade de se cadastrar várias “pessoas”, “imprimir” (na tela) várias vezes o mesmo relatório de pessoas (para averiguar as alterações no array dinâmico) e etc.

# Classe Inicio (1 / 4)

```
public class Inicio{  
    public static void main(String args[]){  
  
        Pessoa p;  
        BDSist bd = new BDSist();  
        Leitura in = new Leitura();  
        int opcao = 0;  
        boolean vai = true; //variável "vai" para condição de parada do sistema  
  
        while(vai){  
            System.out.println("\n( 1 ) - Cadastrar uma Pessoa");  
            System.out.println("\n( 2 ) - Ver lista de Pessoas");  
            System.out.println("\n( 3 ) - Exibir uma Pessoa pelo RG");  
            System.out.println("\n( 4 ) - Alterar dados de uma Pessoa pelo RG");  
            System.out.println("\n( 5 ) - Excluir a primeira ocorrencia de uma Pessoa RG");  
            System.out.println("\n( 6 ) - Sair");  
  
            opcao=Integer.parseInt(in.inData((" \n\n Digite o NUMERO da opcao: ")));
```

## Classe Inicio (2 / 4)

```
switch(opcao){
```

**case 1:**

```
    p = new Pessoa();  
    System.out.println("\n Cadastro de Pessoas\n");  
    p.setRg(Integer.parseInt(in.inData("\nRG...: ")));  
    p.setNome(in.inData("\nNome...: "));  
    bd.setBDPes(p);  
    break;
```

**case 2:**

```
    System.out.println("\n Lista de Pessoas\n");  
    bd.impLista();  
    break;
```



# Classe Inicio (3 / 4)

## case 3:

```
System.out.println("\n Consultar Pessoa pelo RG ");
p = new Pessoa();
p.setRg(Integer.parseInt(in.inData("\n Informe o RG: ")));
p = bd.consRg(p);
if(p==null){
    System.out.println("\n Pessoa não existente");
}
else{
    System.out.println("\n RG..: "+p.getRg());
    System.out.println("\n Nome: "+p.getNome());
}
break;
```

## case 4:

```
System.out.println("\n Alterar dados da Pessoa pelo RG");
int local = 0;
p = new Pessoa();
p.setRg(Integer.parseInt(in.inData("\n Informe o RG: ")));
local = bd.consRg(p,local);
if(local < 0){
    System.out.println("\n Pessoa não existente");
}
else{
    p.setRg(Integer.parseInt(in.inData("\n NOVO rg...: ")));
    p.setNome(in.inData("\n NOVO nome...: "));
    bd.altItemBDPes(local, p);
}
break;
```

# Classe Inicio (4 / 4)

**case 5:**

```
System.out.println("\n Excluir a primeira ocorrencia da Pessoa");  
p = new Pessoa();  
p.setRg(Integer.parseInt(in.inData("\n Informe o RG: ")));  
p.setNome(in.inData("\nNome...: "));  
bd.delltemBDPes(p);  
break;
```

**case 6:**

```
System.exit(0);  
break;
```

**default:**

```
System.out.println("Outras Op es (Valor fora da escala do menu)...: "+opcao);  
break;
```

```
} //fim do switch
```

```
} //fim do laço (while)
```

```
} // fim do main
```

```
}//final da classe
```

**Obs.: O case 5, deve ser feito. Não está funcionando do modo desejado**