

Vetores de Objetos e Collection Framework

AGENDA



- Vetores
- Diferenças entre vetores e Collections
- Descanso

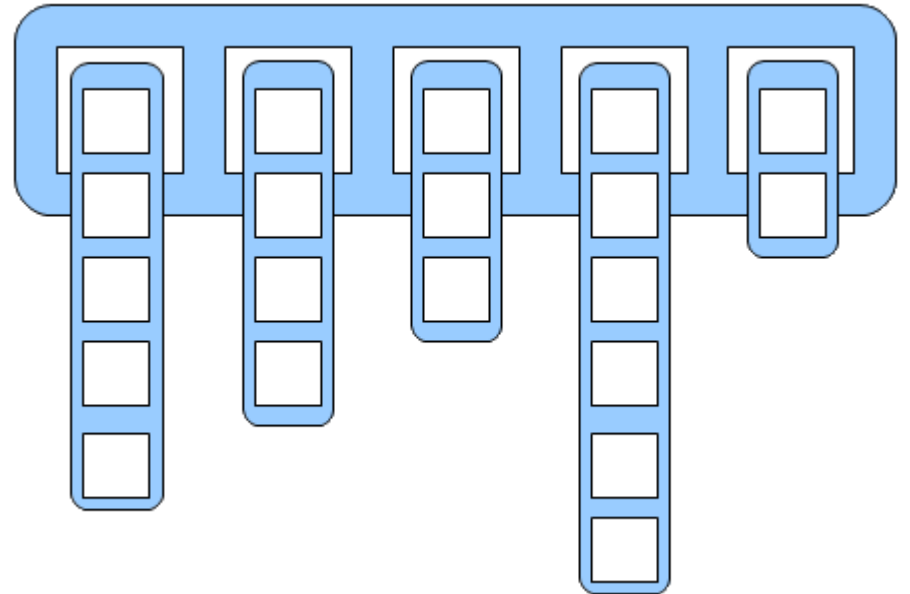
VETORES

- Vetor

```
String nome[] = new String[5];
```

```
double salario[] = new double[5];
```

```
Date nasc[] = new Date[5];
```

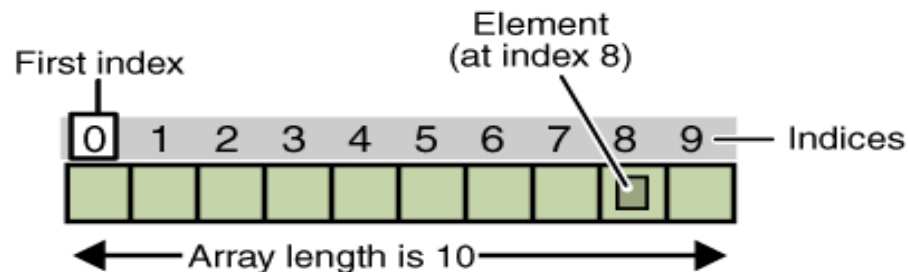


VETORES

Vetor (Array) é o nome de uma matriz unidimensional considerada a mais simples das estruturas de dados, ou seja, podemos dizer que é um conjunto de variáveis de mesmo tipo. Geralmente é constituída por dados do mesmo tipo (homogêneos) e tamanho que são agrupados continuamente na memória e acessados por sua posição (índice - geralmente um número inteiro – referência da localização dentro da estrutura) dentro do vetor. Na sua inicialização determina-se o seu tamanho que geralmente não se modifica mesmo que utilizemos menos elementos do que determinado à princípio.

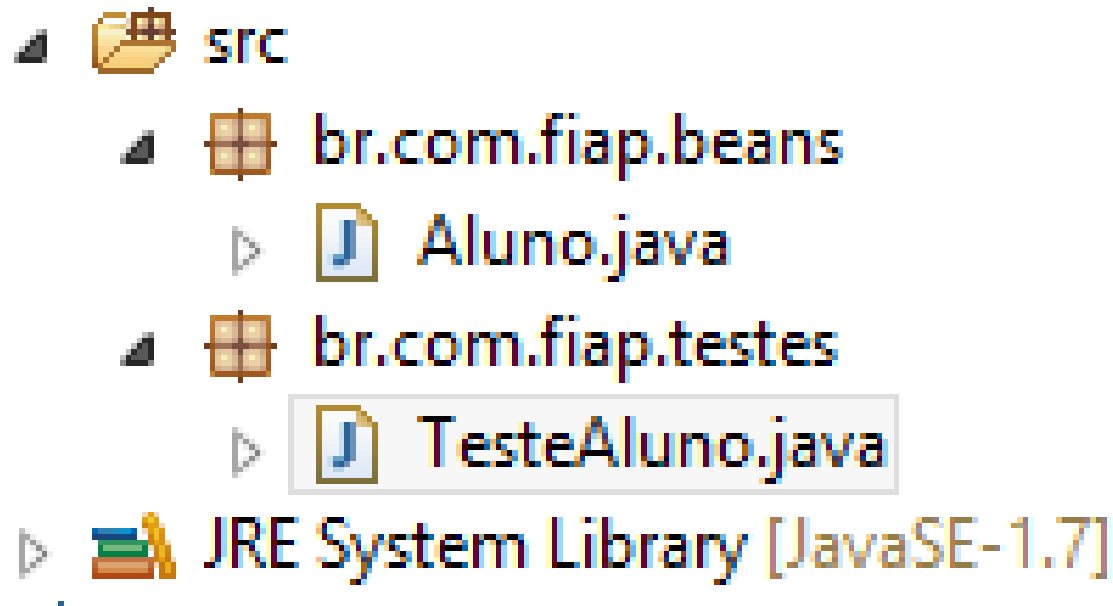
VETORES

Utilizamos um vetor para representar os dados em termos de conjuntos. Um vetor é uma coleção de variáveis de um mesmo tipo que compartilham o mesmo nome e que ocupam posições consecutivas de memória. Cada variável da coleção denomina-se elemento e é identificado por um índice.



VETORES DE OBJETOS - EXEMPLO

Inicie um novo projeto, chamado: VetorObjeto, e monte a estrutura abaixo:



```
package br.com.fiap.beans;
public class Aluno {
    private String nome;
    private double media;
    private int faltas;
    private int idade;
    public Aluno(String n, double m, int f, int i){
        nome = n;
        media = m;
        faltas = f;
        idade =i;
    }
    public Aluno() {
        super();
    }
    public String getNome() {
        return nome;
    }
}
```

VETORES DE OBJETOS - EXEMPLO

```
public static void main(String[] args) {  
    int resp =0, intFaltas=0,  
        intIdade=0, indice=0;  
    double dblMedia =0;  
    String strNome = null;  
    Aluno[] objeto = new Aluno[10];  
    while (resp==0){  
        strNome = JOptionPane.showInputDialog  
            ("Digite o nome do aluno");  
        dblMedia = Double.parseDouble  
            (JOptionPane.showInputDialog  
                ("Digite a média"));  
        intFaltas = Integer.parseInt  
            (JOptionPane.showInputDialog  
                ("Digite as faltas"));
```


VETORES DE OBJETOS - EXEMPLO

```
intIdade = Integer.parseInt
    (JOptionPane.showInputDialog
        ("Digite a idade"));
objeto[indice]= new Aluno();
objeto[indice].setIdade(intIdade);
objeto[indice].setFaltas(intFaltas);
objeto[indice].setMedia(dblMedia);
objeto[indice].setNome(strNome);
indice+=1;
resp = JOptionPane.showConfirmDialog
    (null,
        "Deseja Continuar?", "Camadas",
        JOptionPane.YES_NO_OPTION,
        JOptionPane.QUESTION_MESSAGE);
}
```

VETORES DE OBJETOS - EXEMPLO

```
// Varre somente os elementos utilizados
for(int i=0;i<indice;i++){
    System.out.println("\n"+objeto[i].getNome()
        +"\n"+objeto[i].getMedia());
}

// Varre todos os elementos
for (Aluno aluno : objeto) {
    System.out.println("\n"+aluno.getNome()
        +"\n"+aluno.getMedia());
}
```

DESCANSO - VETORES DE OBJETOS

Monte os métodos para:

- Exibir a média de todas notas dos alunos adicionados;
- Exibir o total das faltas de todos os alunos;
- Exibir a média de todas as idades dos alunos adicionados;
- Exibir o aluno mais velho; e
- Exibir o aluno mais novo.

```
public static double medias
    (Aluno a[], int ind){
    double total=0;
    for(int i=0;i<ind;i++){
        total+=a[i].getMedia();
    }
    return total/ind;
}

System.out.println("Média"+
    medias(objeto, indice));
```

- Dentro do método main():

DIFERENÇAS ENTRE VETORES E COLLECTION'S

- Coleções são estruturas de dados utilizadas para agrupar objetos que permitem de maneira eficiente e prática o armazenamento e organização de objetos
- De forma contrária ao vetor(*array*), elas permitem que um número arbitrário de objetos seja armazenado numa estrutura
- Coleções podem ser utilizadas para representar vetores, listas, pilhas, filas, mapas, conjuntos e outras estruturas de dados
 - A escolha de um tipo de estrutura depende dos requisitos do problema que se deseja resolver
- Várias aplicações necessitam utilizar as coleções de objetos, por exemplo: agendas pessoais, catálogos de bibliotecas, cadastro de funcionários
- São amplamente utilizadas no acesso a dados em bases de dados, principalmente no resultado de buscas

DIFERENÇAS ENTRE VETORES E COLLECTION'S

- O que é o collections framework?

É um conjunto de classes e interfaces, localizadas no pacote “java.util”.

- Qual a finalidade deste conjunto?

Suprir e permitir facilidades para a manipulação dos dados quando comparado ao uso dos arrays.

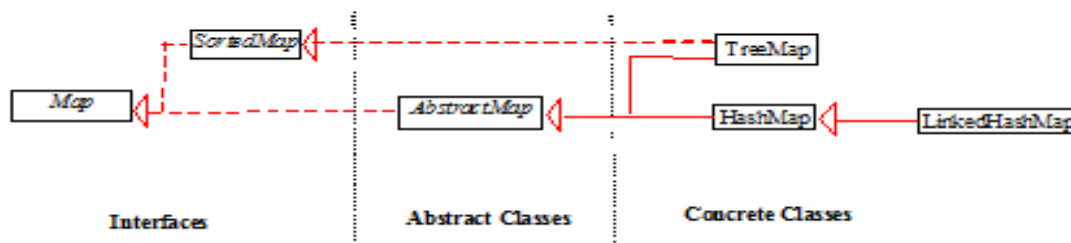
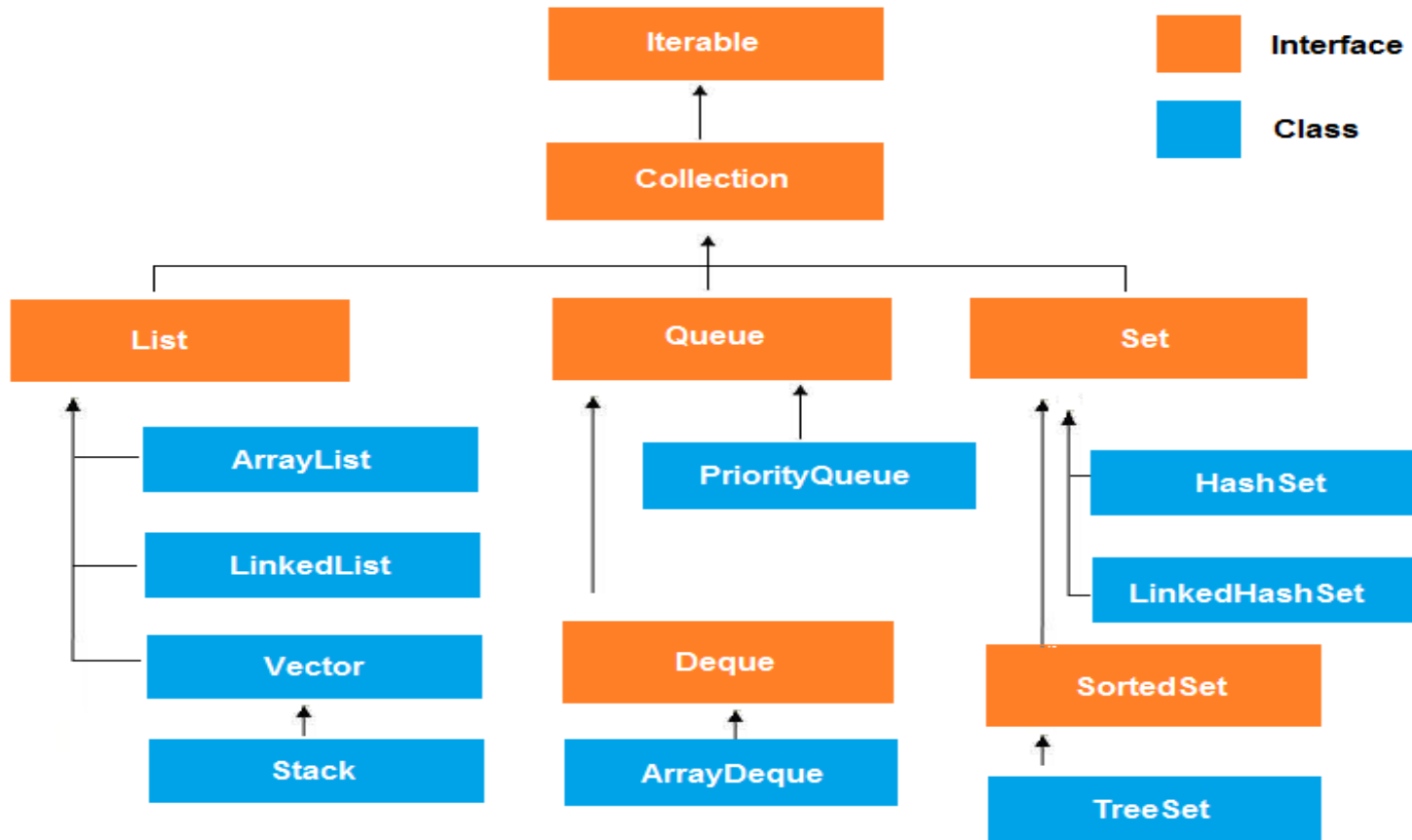
DIFERENÇAS ENTRE VETORES E COLLECTION'S

- Por que utilizar estruturas de dados no lugar dos arrays?

Alguns motivos consideráveis:

- É impossível buscar diretamente por um determinado elemento cujo índice seja desconhecido.
- É impossível saber a quantidade de posições já utilizadas, sem a utilização de métodos ou recursos paralelos.

HIERARQUIA DAS COLLECTION'S



HIERARQUIA DAS COLLECTION'S

Todas as estruturas das collections contêm três características:

- Interfaces
Tipo de dados abstratos representando Collections. Interfaces permitem que as Collections sejam manipuladas independentemente dos detalhes de sua representação. Em linguagens orientadas a objeto com o Java, estas interfaces geralmente formam uma hierarquia
- Implementações
Implementações concretas das interfaces das Collections. Em essência, são estruturas de dados reusáveis
- Algoritmos
Métodos que desenvolvem computações úteis, como procura e armazenamento, sobre objetos que implementam as interfaces das Collections. Estes algoritmos são chamados de polimórficos, pois o mesmo método pode ser usado em diferentes implementações de apropriadas interfaces de Collections. Em essência, algoritmos são funcionalidades reusáveis

LIST - ARRAYLIST

- Interface - `java.util.List`

Esta interface especifica o que uma classe deve fazer para ser uma lista. A implementação **ArrayList**, é a mais utilizada da interface List, suas principais características são:

- Representa uma sequência de elementos - *ordered collection* -
- pode conter elementos duplicados
- Permite controlar a posição de inserção de um elemento e acessar elementos por sua posição
- Permite procurar por um objeto específico na lista e retornar sua posição numérica
- Ótimo desempenho para acesso em listas sem muitas modificações no início e meio



UMA ARRAYLIST NÃO É UM ARRAY!!!

LIST

- Interface - `java.util.List`

LinkedList

- possui métodos adicionais para remover o primeiro ou o último item da lista;
- pode obter uma melhor performance que o `ArrayList`

Vector

- trabalha com processos em paralelo, e pode consumir mais recursos que as outras duas implementações.

LIST - ARRAYLIST

- Seus principais métodos:

Método	Descrição
add ¹	Adiciona um objeto numa determinada posição
get	Retorna o objeto localizado numa determinada posição
remove ¹	Remove um objeto localizado numa determinada posição
set	Coloca um objeto numa determinada posição (substitui objetos)
indexOf	Retorna a posição de um objeto na lista
lastIndexOf	Retorna a última posição de um objeto na lista
subList	Retorna parte de uma lista

¹método sobrecarregado

SET - HASHSET

- Uma coleção que não pode conter elementos duplicados
 - corresponde à abstração de um conjunto
 - contem somente os métodos herdados da interface **Collection**:

Método	Descrição
add	Adiciona um objeto no Set
clear	Remove todos objetos do Set
contains	Verifica se o Set possui um objeto determinado
isEmpty	Verifica se o Set está vazio
remove	Remove um objeto do Set
size	Retorna o quantidade de objetos no Set
toArray	Retorna uma array contendo os objetos do Set

- A classe **HashSet** implementa esta interface

MAP - HASHMAP

- Representa um objeto que mapeia chaves em valores
- Um objeto **Map** não pode conter chaves duplicadas
 - cada chave é mapeada para um único valor
- A classe **HashMap** implementa esta interface

Método	Descrição
clear	Remove todos os mapeamentos
containsKey	Verifica se uma chave já está presente no mapeamento
containsValue	Verifica se um valor já está presente no mapeamento
get	Retorna o valor associado a uma chave determinada
isEmpty	Verifica se o mapeamento está vazio
keySet	Retorna um Set contendo as chaves
put	Adiciona um mapeamento
remove	Remove um mapeamento
size	Retorna o número de mapeamentos
values	Retorna uma Coleção contendo os valores dos mapeamentos

LIST - ARRAYLIST

- Interface - java.util.List

Declaração de uma ArrayList pode ser:

```
List nossaLista = new ArrayList();
```

Array

Recurso denominado **Generics**

```
ArrayList<String> nossaLista = new  
    ArrayList<String>();
```

LIST – ARRAYLIST - EXEMPLO

- Crie um novo projeto, chamado **Lista** e acrescente somente uma classe para teste, chamada **TesteLista**, no método **main()** digite:

```
List cargos = new ArrayList();  
cargos.add("DBA");  
cargos.add("Analista");  
cargos.add("Desenvolvedor");  
System.out.println(cargos);  
Collections.sort(cargos);  
System.out.println(cargos);
```

Entretanto, a lista anterior é uma lista de qualquer objeto, o melhor é definirmos o tipo de objetos que estarão na lista.

```
public static void main(String[] args) {  
    List<String> cargos = new ArrayList<String>();  
    ...  
}
```


LIST – ARRAYLIST - EXEMPLO

- Aperfeiçoando um pouco nosso exemplo:

```
List<String> cargos = new ArrayList<String>();  
cargos.add("DBA");  
cargos.add("Analista");  
cargos.add("Desenvolvedor");  
System.out.println(cargos);  
// método sort() ordena os objetos  
Collections.sort(cargos);  
System.out.println(cargos);  
// método get() retorna o objeto com base na posição  
for(int i =0;i<2;i++){  
    System.out.println(cargos.get(i).toUpperCase());  
}  
String cargo = JOptionPane.showInputDialog("O que deseja pesquisar?");  
// método indexOf procura o objeto e retorna sua posição  
if(cargos.indexOf(cargo)>=0){  
    System.out.println("Existe");  
}else{  
    System.out.println("Não Existe");  
}
```

LIST – ARRAYLIST - EXEMPLO

- Como seria a ordenação de objetos e não de dados?

Crie um novo projeto chamado OrdemObjetos

Crie os pacotes:

br.com.fiap.beans

br.com.fiap.testes

Dentro do “beans” monte a classe **Cargo** com:


- os atributos cargo (String), nível (String) e salário (double)
- Construtores, getter's e setter's
- Um método get que exiba os três atributos como no exemplo abaixo:

DBA – Nível: Júnior – Salário: 3500

LIST – ARRAYLIST - EXEMPLO

- Na classe de Teste, dentro do método main(), digite:

```
public static void main(String[] args) {  
    Cargo c1 = new Cargo("DBA", "Junior", 3500);  
    Cargo c2 = new Cargo("Estagiário", "Pleno", 5500);  
    Cargo c3 = new Cargo("Analista", "Junior", 3000);  
    List<Cargo> cargos= new ArrayList<Cargo>();  
    cargos.add(c1);  
    cargos.add(c2);  
    cargos.add(c3);  
    Collections.sort(cargos);  
    System.out.println(cargos);  
}
```



**DEVEMOS IMPLEMENTAR O MÉTODO
compareTo()**

LIST – ARRAYLIST - EXEMPLO

- Na classe de **Cargo**, acrescente o seguinte método:

```
public class Cargo implements Comparable<Cargo>{  
    private String cargo, nivel;  
    private double salario;
```

```
    public int compareTo(Cargo outro) {  
        if (this.salario < outro.salario) {  
            return -1;  
        }  
        if (this.salario > outro.salario) {  
            return 1;  
        }  
        return 0;  
    }
```

```
    public Cargo(String c, String n, double s) {
```

O “sysout” está retornando a mensagem desejada???

LIST – ARRAYLIST - EXEMPLO

- Na classe de **Teste**, acrescente o seguinte código:

```
public static void main(String[] args) {  
    Cargo c1 = new Cargo("DBA", "Junior", 3500);  
    Cargo c2 = new Cargo("Estagiário", "Pleno", 5500);  
    Cargo c3 = new Cargo("Analista", "Junior", 3000);  
    List<Cargo> cargos= new ArrayList<Cargo>();  
    cargos.add(c1);  
    cargos.add(c2);  
    cargos.add(c3);  
    Collections.sort(cargos);  
    //System.out.println(cargos);  
    for (Cargo c : cargos){  
        System.out.println(c.getCargo());  
    }  
}
```

DESCANSO

1º-) Faça com que a ordem não seja mais por salário e sim por cargo em ordem crescente.

```
public int compareTo(Cargo outro) {  
    return this.cargo.compareTo(outro.cargo);  
}
```

DESCANSO

2º-) Faça com que a ordem agora seja decrescente.

```
public int compareTo(Cargos outro) {  
    return this.cargo.compareTo(outro.cargo) * -1;  
}
```

DÚVIDAS...



REFERÊNCIAS



Java: Como Programar, 8ª Edição
Capítulo 7 – Arrays e ArrayLists
7.14 Introdução a coleções e classe
ArrayList

Classe Collections
[http://docs.oracle.com/javase/7/docs/api/
java/util/Collections.html](http://docs.oracle.com/javase/7/docs/api/java/util/Collections.html)