

# Aula 9

# Interfaces

Vamos aprender hoje:

- Array
- Collections Framework
  - List
  - ArrayList
  - Generics
  - Set
- Map

# Arrays

Podemos declarar diversas variáveis e usá-las:

```
double var1 = 1.0;
```

```
double var2 = 2.0;
```

```
double var3 = 3.0;
```

```
double var4 = 4.0;
```

# Arrays

Podemos declarar um vetor (array) de double;

```
int tamanho = 10;
double[] arrayVar;
arrayVar = new double[tamanho];

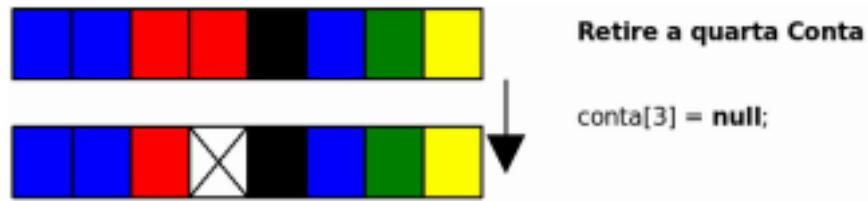
for(int i = 0; i < tamanho; i++)
    arrayVar[i] = i * tamanho;

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

# Array

## Manipulação de arrays

- Manipular é bastante trabalhoso.
- Essa dificuldade aparece em diversos momentos:
  - Não podemos redimensionar um array em Java;
  - É impossível buscar diretamente por um determinado elemento cujo índice não se sabe;
  - Não conseguimos saber quantas posições do array já foram populadas sem criar, para isso, métodos auxiliares



# Collections

- Foi inserida no Java na versão 2;
- A API de Collections é robusta;
- Diversas classes com estruturas de dados avançadas.
- Oferece diversas estruturas para utilização:
  - List
  - ArrayList
  - Generics
  - Set

# Collections - List

## List

- É uma coleção que permite elementos duplicados ;
- Mantém uma ordenação específica entre os elementos;
- Trabalha com um array interno para gerar uma lista;
- A implementação mais utilizada da interface List é a ArrayList;
- Sim, List é uma interface;

# Collections - ArrayList

## ArrayList

```
List lista = new ArrayList();  
lista.add("Manoel");  
lista.add("Joaquim");  
lista.add("Maria");  
lista.remove(1);  
lista.size();
```



# Collections - ArrayList

## ArrayList

- ArrayList não é um array;
- Internamente usa um array como estrutura para armazenar os dados;
- Porém este atributo está propriamente encapsulado;
- Não podemos acessá-lo,
- Não podemos usar [ ]
- Não podemos acessar o atributo length;

# Collections - ArrayList

## ArrayList

- Operacionais - add(), remove(), clear(), get();
- Informativos - size(), isEmpty(), sort();
- Uma lista é uma excelente alternativa a um array comum;
- Todos os benefícios de arrays, sem cuidado com remoções e espaço;
- Trabalha do modo mais genérico possível;
- Não há uma ArrayList específica para Strings , outra para Números, outra para Datas etc;
- Todos os métodos trabalham com Object;

# Collections - ArrayList

## ArrayList

- Isso mesmo! Em uma lista, é possível colocar qualquer Object .
- Com isso, é possível misturar objetos:
- E na hora de recuperar esses objetos?
- Como o método get devolve um Object. Precisamos fazer o cast.
- Mas com uma lista com vários objetos de tipos diferentes, isso pode não ser tão simples...
- Geralmente usamos listas como mesmo tipo de dado.

# Collections - ArrayList

## ArrayList

```
Funcionario func = new Funcionario()  
ContaCorrente cc = new ContaCorrente()  
List lista = new ArrayList();  
lista.add("Manoel");  
lista.add("Joaquim");  
lista.add(func);  
lista.add(cc);  
Funcionario func2 = (Funcionario) lista.get(3);  
lista.size();
```

# Collections - Generics

## Generics

- No Java 5.0, foi adicionado o recurso para restringir as listas a um determinado tipo de objetos (e não qualquer Object );

```
List<ContaCorrente> contas = new ArrayList<ContaCorrente>();  
contas.add(c1);  
contas.add(c3);  
contas.add(c2);  
contas.add("uma string qualquer"); // isso não compila mais!!
```

- O uso de Generics também elimina a necessidade de casting

# Collections - Generics

## Generics

- A partir do Java 7 houve uma simplificação na sintaxe do generics;
- Se instanciarmos na mesma linha da declaração, não é necessário passar os tipos novamente, basta usar `new ArrayList<>()`.
- É conhecido como operador diamante:

```
List<ContaCorrente> contas = new ArrayList<>();
```

# Collections - Generics

## Ordenação de Listas

- A classe Collections traz um método estático sort que recebe um List como argumento e o ordena por ordem crescente. Por exemplo:

```
List<String> lista = new ArrayList<>();  
lista.add("Sérgio");  
lista.add("Paulo");  
lista.add("Guilherme");  
// repare que o toString de ArrayList foi sobrescrito:  
System.out.println(lista);  
Collections.sort(lista);  
System.out.println(lista);
```

## Ordenação de Listas

- Para ordenar objetos precisamos determinar um critério;
- Esse critério irá determinar qual elemento vem antes de qual.
- É necessário instruir o sort sobre como comparar um objeto
- Para isto, o método sort necessita que todos os objetos da lista sejam **comparáveis**;
- Isso se dará através de um método que fará tal comparação com outro objeto;
- Como é que o método sort terá a garantia de que a sua classe possui esse método?
- Isso será feito, novamente, através de um contrato, de uma interface!



## Ordenação de Listas

- Vamos fazer com que os elementos da nossa coleção implementem a interface `java.lang.Comparable`, que define o método `int compareTo(Object)`.
- Este método deve retornar zero, se o objeto comparado for igual a este objeto
- Um número negativo, se este objeto for menor que o objeto dado
- E um número positivo, se este objeto for maior que o objeto dado.
- Para ordenar os Funcionários por salário, basta implementar o `Comparable`

# Collections - Generics

- Ordenação de Listas

```
public class DPFuncionario implements Comparable<DPFuncionario>{
    //Omissão de demais atributos e métodos
    private double salario;
    @Override
    public int compareTo(DPFuncionario f) {
        if(this.getSalario()<f.getSalario()) {
            return -1;
        }
        if(this.getSalario()>f.getSalario()) {
            return 1;
        }
        return 0;
    }
}
```

# Collections - Generics

## • Ordenação de Listas

```
Funcionario f1 = new Funcionario();  
f1.setSalario(5000);  
Funcionario f2 = new Funcionario();  
f2.setSalario(1500);  
Funcionario f3 = new Funcionario();  
f3.setSalario(2000);  
List<Funcionario> listaF = new ArrayList<>();  
listaF.add(f1);  
listaF.add(f3);  
listaF.add(f2);  
Collections.sort(listaF); // qual seria o critério para esta ordenação?
```

# Collections - Lambdas

- Novidades que não vamos abordar mas é bom saber:
- No Java 8 foram incluídas outras funcionalidades nas Collections;
- O uso dos chamados Streams;
- São utilizados em funções Lambda;
- Possuem uma sintaxe bem diferente do que costumamos trabalhar;
- Isso fica um pouco fora do escopo de um curso inicial de Java.
- Mas que fique no radar pois esse assunto é muito importante;

# Collections - Set

- É uma coleção que não permite elementos duplicados;
- Um conjunto é representado pela interface Set:
  - HashSet , LinkedHashSet e TreeSet .

```
Set<String> cargos = new HashSet<>();  
cargos.add("Gerente");  
cargos.add("Diretor");  
cargos.add("Presidente");  
cargos.add("Secretária");  
cargos.add("Funcionário");  
cargos.add("Diretor"); // repetido!  
// imprime na tela todos os elementos  
System.out.println(cargos);
```

# Collections - Map

## Map

- Muitas vezes queremos buscar rapidamente um objeto dado alguma informação sobre ele.
- Como exemplo, dada a placa do carro, obter todos os dados do carro.
- Poderíamos utilizar uma lista e percorrer todos os seus elementos;
- Mas a performance é péssima, mesmo em listas não muito grandes.
- Aqui entra o map.

# Collections - Map

## Map

- Um map é composto por um conjunto de associações entre um objeto chave a um objeto valor.
- O método put(Object Chave, Object Valor) da interface Map recebe a chave e o valor de uma nova associação.
- O método get(Object Chave) retorna o Object Valor associado a ele;
- O método keySet() retorna um Set com as chaves daquele mapa;
- O método values() retorna a Collection com todos os valores que foram associados a alguma das chaves.

# Map

```
ContaCorrente c1 = new ContaCorrente();
c1.deposita(10000);
ContaCorrente c2 = new ContaCorrente();
c2.deposita(3000);
// cria o mapa
Map<String, ContaCorrente> 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!)
ContaCorrente contaDoDiretor = mapaDeContas.get("diretor");
System.out.println(contaDoDiretor.getSaldo());
```