Aula 8

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 Funcionario implements Comparable<Funcionario>{
    //Omissão de demais atributos e métodos
    private double salario;
    @Override
    public int compareTo(Funcionario 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());
```