Ficha de trabalho #9

Interface Comparator
Interface Comparable

Como visto anteriormente, o Java permite que as classes implementem interfaces. As interfaces Comparator e Comparable permitem a ordenação de Collections (por exemplo, ArrayLists).

A **interface Comparable** permite ordenar coleções por um critério. Por exemplo, se necessitarmos de ordenar pessoas pela ordem alfabética do nome, podemos ter o seguinte na classe Pessoa:

```
public class Pessoa implements Comparable<Pessoa>{
    private String nome;
    private String nif;
    private int idade;

    public Pessoa(String nome, String nif, int idade) {
        this.nome = nome;
        this.nif = nif;
        this.idade = idade;
    }

    // (...)

    @Override
    public int compareTo(Pessoa o) {
        return this.nome.compareTo(o.nome);
    }
}
```

A classe Pessoa implementa a interface Comparable, o que irá obrigar a implementar o método compareTo.

O método compareTo irá comparar duas pessoas pelo seu nome, neste caso utilizando o método compareTo implementado para Strings.

```
import java.util.ArrayList;
import java.util.Collections;

public class TesteOrdenacao {

    public static void main(String[] args) {
        Pessoa p1 = new Pessoa("Zé","111111111",33);
        Pessoa p2 = new Pessoa("Carla","222222222",21);
        Pessoa p3 = new Pessoa("Daniel","333333333",47);
        Pessoa p4 = new Pessoa("Xavier","444444444",18);
        Pessoa p5 = new Pessoa("Edgar","555555555",22);
        Pessoa p6 = new Pessoa("Paulo","666666666",28);

        ArrayList<Pessoa> pessoas = new ArrayList<>();
        pessoas.add(p1);
        pessoas.add(p2);
        pessoas.add(p3);
        pessoas.add(p4);
        pessoas.add(p5);
        pessoas.add(p6);

        System.out.println("Pessoas (ordem de inserção):");
        for(Pessoa p: pessoas) {
            System.out.println(p);
        }

        System.out.println("\nPessoas (ordem alfabética):");
        Collections.sort(pessoas);
        for(Pessoa p: pessoas) {
            System.out.println(p);
        }

        System.out.println("\nPessoas (ordem alfabética inversa):");
        Collections.sort(pessoas,Collections.reverseOrder());
        for(Pessoa p: pessoas) {
            System.out.println(p);
        }
    }
}
```

No main podemos então criar pessoas e um ArrayList de pessoas e adicionar as pessoas ao ArrayList. Posteriormente podemos mostrar a listagem de pessoas:
Pela ordem que foram **inseridas** no ArrayList
Por ordem **alfabética** do nome, usando:
`Collections.sort(pessoas);`
Por ordem **alfabética inversa** do nome, usando:
`Collections.sort(pessoas, Collections.reverseOrder());`

O output deste Código será:

```
Pessoas (ordem de inserção):
Pessoa [nome=Zé, nif=111111111, idade=33]
Pessoa [nome=Carla, nif=222222222, idade=21]
Pessoa [nome=Daniel, nif=333333333, idade=47]
Pessoa [nome=Xavier, nif=444444444, idade=18]
Pessoa [nome=Edgar, nif=555555555, idade=22]
Pessoa [nome=Paulo, nif=666666666, idade=28]

Pessoas (ordem alfabética):
Pessoa [nome=Carla, nif=222222222, idade=21]
Pessoa [nome=Daniel, nif=333333333, idade=47]
Pessoa [nome=Edgar, nif=555555555, idade=22]
Pessoa [nome=Paulo, nif=666666666, idade=28]
Pessoa [nome=Xavier, nif=444444444, idade=18]
Pessoa [nome=Zé, nif=111111111, idade=33]

Pessoas (ordem alfabética inversa):
Pessoa [nome=Zé, nif=111111111, idade=33]
Pessoa [nome=Xavier, nif=444444444, idade=18]
Pessoa [nome=Paulo, nif=666666666, idade=28]
Pessoa [nome=Edgar, nif=555555555, idade=22]
Pessoa [nome=Daniel, nif=333333333, idade=47]
Pessoa [nome=Carla, nif=222222222, idade=21]
```

Também é possível fazer ordenação por um atributo numérico. Por exemplo, para ordenar pela idade, o método compareTo passaria a ser:

```java
public int compareTo(Pessoa o) {
    if(this.idade > o.idade)
        return 1;
    if(this.idade < o.idade)
        return -1;
    return 0;
}
```

Neste caso, o output de um código main semelhante ao anterior seria o apresentado à direita

```
Pessoas (ordem de inserção):
Pessoa [nome=Zé, nif=111111111, idade=33]
Pessoa [nome=Carla, nif=222222222, idade=21]
Pessoa [nome=Daniel, nif=333333333, idade=47]
Pessoa [nome=Xavier, nif=444444444, idade=18]
Pessoa [nome=Edgar, nif=555555555, idade=22]
Pessoa [nome=Paulo, nif=666666666, idade=28]

Pessoas (ordem de idade crescente):
Pessoa [nome=Xavier, nif=444444444, idade=18]
Pessoa [nome=Carla, nif=222222222, idade=21]
Pessoa [nome=Edgar, nif=555555555, idade=22]
Pessoa [nome=Paulo, nif=666666666, idade=28]
Pessoa [nome=Zé, nif=111111111, idade=33]
Pessoa [nome=Daniel, nif=333333333, idade=47]

Pessoas (ordem de idade decrescente):
Pessoa [nome=Daniel, nif=333333333, idade=47]
Pessoa [nome=Zé, nif=111111111, idade=33]
Pessoa [nome=Paulo, nif=666666666, idade=28]
Pessoa [nome=Edgar, nif=555555555, idade=22]
Pessoa [nome=Carla, nif=222222222, idade=21]
Pessoa [nome=Xavier, nif=444444444, idade=18]
```

Note-se que a interface Comparable apenas permite fazer a comparação/ordenação por um critério de cada vez. Para utilizar um critério diferente, com essa interface, teríamos de estar constantemente a alterar código. Para prevenir essa situação, podemos usar a **interface Comparator**, que permite ordenar coleções por vários critérios.

Se por exemplo quisermos ter a possibilidade de ordenar pessoas por nome (já implementado com a interface Comparator), mas também por idade e por NIF, podemos recorrer à interface Comparator e, neste caso, não temos de fazer nenhuma alteração à classe pessoa. Teremos, no entanto, de criar duas classes:

```java
import java.util.Comparator;

public class ComparadorIdade implements Comparator<Pessoa>{

    @Override
    public int compare(Pessoa o1, Pessoa o2) {
        if(o1.getIdade() > o2.getIdade())
            return 1;
        if(o1.getIdade() < o2.getIdade())
            return -1;
        return 0;
    }
}
```

Uma classe ComparadorIdade que implementa um comparador de pessoas.

Aqui definimos o método compare, que compara dois objetos do tipo Pessoa a partir da sua idade.

```java
import java.util.Comparator;

public class ComparadorNif implements Comparator<Pessoa>{

    @Override
    public int compare(Pessoa o1, Pessoa o2) {
        return o1.getNif().compareTo(o1.getNif());
    }
}
```

Uma classe ComparadorNif que também implementa um comparador de pessoas.

Aqui o método compare irá comparar dois objetos do tipo Pessoa pelos seus NIFs.

```java
import java.util.ArrayList;
import java.util.Collections;

public class TesteOrdenacao {

    public static void main(String[] args) {
        Pessoa p1 = new Pessoa("Zé","111111111",33);
        Pessoa p2 = new Pessoa("Carla","222222222",21);
        Pessoa p3 = new Pessoa("Daniel","333333333",47);
        Pessoa p4 = new Pessoa("Xavier","444444444",18);
        Pessoa p5 = new Pessoa("Edgar","555555555",22);
        Pessoa p6 = new Pessoa("Paulo","666666666",28);

        ArrayList<Pessoa> pessoas = new ArrayList<>();
        pessoas.add(p1); pessoas.add(p2); pessoas.add(p3);
        pessoas.add(p4); pessoas.add(p5); pessoas.add(p6);

        System.out.println("Pessoas (ordem de inserção):");
        for(Pessoa p: pessoas) {
            System.out.println(p);
        }

        System.out.println("\nPessoas (ordem alfabética):");
        Collections.sort(pessoas);
        for(Pessoa p: pessoas) {
            System.out.println(p);
        }

        System.out.println("\nPessoas (ordem alfabética inversa):");
        Collections.sort(pessoas,Collections.reverseOrder());
        for(Pessoa p: pessoas) {
            System.out.println(p);
        }

        ComparadorIdade ordemIdade = new ComparadorIdade();
        System.out.println("\nPessoas (ordem de idade crescente):");
        Collections.sort(pessoas,ordemIdade);
        for(Pessoa p: pessoas) {
            System.out.println(p);
        }

        System.out.println("\nPessoas (ordem de idade decrescente):");
        Collections.sort(pessoas,Collections.reverseOrder(ordemIdade));
        for(Pessoa p: pessoas) {
            System.out.println(p);
        }

        ComparadorNif ordemNIF = new ComparadorNif();
        System.out.println("\nPessoas (ordem de NIF crescente):");
        Collections.sort(pessoas,ordemNIF);
        for(Pessoa p: pessoas) {
            System.out.println(p);
        }

        System.out.println("\nPessoas (ordem de NIF decrescente):");
        Collections.sort(pessoas,Collections.reverseOrder(ordemNIF));
        for(Pessoa p: pessoas) {
            System.out.println(p);
        }
    }
}
```

Podemos agora usar todos os métodos de ordenação que implementámos no main, com ordenações crescentes e decrescentes pelos vários critérios.

O código à esquerda terá o seguinte output:

```
Pessoas (ordem de inserção):
Pessoa [nome=Zé, nif=111111111, idade=33]
Pessoa [nome=Carla, nif=222222222, idade=21]
Pessoa [nome=Daniel, nif=333333333, idade=47]
Pessoa [nome=Xavier, nif=444444444, idade=18]
Pessoa [nome=Edgar, nif=555555555, idade=22]
Pessoa [nome=Paulo, nif=666666666, idade=28]

Pessoas (ordem alfabética):
Pessoa [nome=Xavier, nif=444444444, idade=18]
Pessoa [nome=Carla, nif=222222222, idade=21]
Pessoa [nome=Edgar, nif=555555555, idade=22]
Pessoa [nome=Paulo, nif=666666666, idade=28]
Pessoa [nome=Zé, nif=111111111, idade=33]
Pessoa [nome=Daniel, nif=333333333, idade=47]

Pessoas (ordem alfabética inversa):
Pessoa [nome=Daniel, nif=333333333, idade=47]
Pessoa [nome=Zé, nif=111111111, idade=33]
Pessoa [nome=Paulo, nif=666666666, idade=28]
Pessoa [nome=Edgar, nif=555555555, idade=22]
Pessoa [nome=Carla, nif=222222222, idade=21]
Pessoa [nome=Xavier, nif=444444444, idade=18]

Pessoas (ordem de idade crescente):
Pessoa [nome=Xavier, nif=444444444, idade=18]
Pessoa [nome=Carla, nif=222222222, idade=21]
Pessoa [nome=Edgar, nif=555555555, idade=22]
Pessoa [nome=Paulo, nif=666666666, idade=28]
Pessoa [nome=Zé, nif=111111111, idade=33]
Pessoa [nome=Daniel, nif=333333333, idade=47]

Pessoas (ordem de idade decrescente):
Pessoa [nome=Daniel, nif=333333333, idade=47]
Pessoa [nome=Zé, nif=111111111, idade=33]
Pessoa [nome=Paulo, nif=666666666, idade=28]
Pessoa [nome=Edgar, nif=555555555, idade=22]
Pessoa [nome=Carla, nif=222222222, idade=21]
Pessoa [nome=Xavier, nif=444444444, idade=18]

Pessoas (ordem de NIF crescente):
Pessoa [nome=Daniel, nif=333333333, idade=47]
Pessoa [nome=Zé, nif=111111111, idade=33]
Pessoa [nome=Paulo, nif=666666666, idade=28]
Pessoa [nome=Edgar, nif=555555555, idade=22]
Pessoa [nome=Carla, nif=222222222, idade=21]
Pessoa [nome=Xavier, nif=444444444, idade=18]

Pessoas (ordem de NIF decrescente):
Pessoa [nome=Daniel, nif=333333333, idade=47]
Pessoa [nome=Zé, nif=111111111, idade=33]
Pessoa [nome=Paulo, nif=666666666, idade=28]
Pessoa [nome=Edgar, nif=555555555, idade=22]
Pessoa [nome=Carla, nif=222222222, idade=21]
Pessoa [nome=Xavier, nif=444444444, idade=18]
```

Note-se ainda que a implementação de interfaces também é herdada: se tivermos uma classe Funcionario que herda de Pessoa, iria também herdar os métodos de ordenação.

1. No exercício 1 da ficha 6 fizemos um projeto para representar turmas de alunos. Pretende-se agora fazer as adaptações necessárias e criar uma nova classe para o mais em que seja possível listar (com ordem crescente e decrescente):
   - Turmas ordenadas por ano de escolaridade e por ordem alfabética do identificador
   - Alunos ordenados por número de aluno e por ordem alfabética do nome

2. No exercício 2 da ficha 6 fizemos um projeto para representar livrarias com livros. Pretende-se agora fazer as adaptações necessárias e criar uma nova classe para o mais em que seja possível listar (com ordem crescente e decrescente):
   - Livrarias ordenadas por ordem alfabética do nome
   - Livros ordenados pelo preço, ordem alfabética do título e por ISBN

3. No exercício 3 da ficha 6 fizemos um projeto para representar mercearias com produtos. Pretende-se agora fazer as adaptações necessárias e criar uma nova classe para o mais em que seja possível listar (com ordem crescente e decrescente):
   - Mercearias ordenadas por ordem alfabética do nome
   - Produtos ordenados por preço, quantidade em stock e ordem alfabética do nome

4. No exercício 1 da ficha 7 fizemos um projeto para representar colaboradores de uma empresa. Pretende-se agora fazer as adaptações necessárias e criar uma nova classe para o mais em que seja possível listar (com ordem crescente e decrescente):
   - Colaboradores ordenados pelo vencimento, nome e NIF
   - Colaboradores à comissão ordenados pelo salário base
   - Colaboradores à hora ordenados pelo número de horas realizadas

5. No exercício 2 da ficha 7 fizemos um projeto para representar automóveis num stand. Pretende-se agora fazer as adaptações necessárias e criar uma nova classe para o mais em que seja possível listar (com ordem crescente e decrescente):
   - Automóveis ordenados por marca, modelo, o preço e autonomia
   - Automóveis a combustível ordenados pelo consumo
   - Automóveis a bateria ordenados por capacidade da bateria

6. No exercício 3 da ficha 7 fizemos um projeto para representar produtos numa mercearia. Pretende-se agora fazer as adaptações necessárias e criar uma nova classe para o mais em que seja possível listar (com ordem crescente e decrescente):
   - Produtos ordenados por nome e preço final
   - Produtos à unidade ordenados pelo preço unitário
   - Produtos a quilo ordenados pelo preço por quilo
   - Produtos por volume ordenados pelo preço por litro

7. No exercício 1 da ficha 8 fizemos um projeto para representar uma escola com professores e alunos, alguns dos quais são bolseiros. Pretende-se agora fazer as adaptações necessárias e criar uma nova classe para o mais em que seja possível listar (com ordem crescente e decrescente):
   - Pessoas ordenadas por nome e NIF
   - Professores ordenados por salário base, número de horas extra realizadas e vencimento
   - Alunos ordenados por ano de escolaridade
   - Alunos bolseiros ordenados pelo valor da bolsa

8. No exercício 2 da ficha 8 fizemos um projeto para representar atrações turísticas. Pretende-se agora fazer as adaptações necessárias e criar uma nova classe para o mais em que seja possível listar (com ordem crescente e decrescente):
   - Atração ordenadas por nome
   - Museus ordenados por preço do bilhete
   - Parques ordenados por área
   - Parques ordenados de diversões por número de diversões

9. No exercício 3 da ficha 8 fizemos um projeto para representar veículos. Pretende-se agora fazer as adaptações necessárias e criar uma nova classe para o mais em que seja possível listar (com ordem crescente e decrescente):
   - Veículos por ordenados por marca e modelo
   - Veículos motorizados ordenados por potência do motor e por consumo
   - Veículos não motorizados ordenados pelo peso do veículo e pela carga máxima