Interfaces Comparable e Comparator

Instituto Federal do Espirito Santo – Ifes
Campus Colatina – Programação Orientada a Objetos I
Professor: Giovany Frossard Teixeira

Interface Comparable

- A interface Comparable define a ordem natural dos elementos de uma classe. Assim, se uma classe implementa essa interface, ela será capaz, pela ordem natural definida, de ordenar uma lista de objetos dessa classe.
- No próximo slide veremos um exemplo do uso dessa interface.

Exemplo

```
public class Pessoa implements Comparable{

    private String nome;

    private int idade;



    public Pessoa(String nome, int idade) {

        this.nome = nome;

        this.idade = idade;

    }



    @Override

    public String toString() {

        return "Pessoa{ " + "nome = " + nome + " idade = " + idade +
" }";

    }
```

A interface Comparable nos obrigou a implementar o método compareTo tendo os possíveis retornos:

- Maior que zero: o objeto corrente é “maior” que o passado como parâmetro.
- Igual a zero: os objetos tem o mesmo valor na comparação.
- Menor que zero: o objeto passado como parâmetro tem “maior” valor.

```
    public int compareTo(Object o) {

        Pessoa outro = (Pessoa)o;

        // O objeto corrente tem maior idade

        if(this.idade > outro.idade) return 1;

        // as idades são iguais

        else if(this.idade == outro.idade) return 0;

        // O objeto forneceido como parâmetro tem maior idade

        else return -1;

    }

}
```

Exemplo

```
public class Main {

    public static void main(String[] args) {

        ArrayList<Pessoa> pessoas = new ArrayList(10);

        Pessoa p;

        pessoas.add(new Pessoa("Pedro", 10));

        pessoas.add(new Pessoa("Medro", 5));

        pessoas.add(new Pessoa("Ane", 20));

        pessoas.add(new Pessoa("Zana", 30));

        pessoas.add(new Pessoa("Tina", 15));

        Collections.sort(pessoas);

        Iterator<Pessoa> it = pessoas.iterator();

        while (it.hasNext())  {

            p = it.next();

            System.out.println(p);

        }

    }

}
```

```
Pessoa{ nome = Medro idade = 5 }
Pessoa{ nome = Pedro idade = 10 }
Pessoa{ nome = Tina idade = 15 }
Pessoa{ nome = Ane idade = 20 }
Pessoa{ nome = Zana idade = 30 }
```

Interface Comparator

- Mas e se desejássemos fazer a comparação entre 2 objetos por uma ordem diferente da “natural”.
- A interface Comparator entra nesse contexto, ela define a comparação de 2 objetos por um objeto externo.
- No próximo slide vamos utilizar o exemplo anterior e ordenar por **nome** ao invés de **idade** (considerado no exemplo anterior a ordem natural).
- Acrescentaremos na classe Pessoa o método **getNome** para utilizar na classe ComparadorPessoaPorNome definido no próximo slide.
- A interface Comparable pode ser substituída pela interface Comparator, mas o contrário não é verdade.
- A melhor estratégia é definir uma ordem natural e utilizar a interface Comparable, caso seja necessário definir uma outra ordem qualquer utiliza-se a interface Comparator.

Exemplo

```
public class ComparadorPessoaPorNome implements
Comparator<Pessoa> {

    public int compare(Pessoa o1, Pessoa o2) {

        return o1.getNome().compareTo(o2.getNome());

    }

}

public class Main {

    public static void main(String[] args) {

        ArrayList<Pessoa> pessoas = new ArrayList(10);

        Pessoa p;

        pessoas.add(new Pessoa("Pedro", 10));

        pessoas.add(new Pessoa("Medro", 5));

        pessoas.add(new Pessoa("Ane", 20));

        pessoas.add(new Pessoa("Zana", 30));

        pessoas.add(new Pessoa("Tina", 15));

        Collections.sort(pessoas, new ComparadorPessoaPorNome());

        Iterator<Pessoa> it = pessoas.iterator();

        while (it.hasNext())  {

            p = it.next();

            System.out.println(p);

        }

    }

}
```

```
Pessoa{ nome = Ane idade = 20 }
Pessoa{ nome = Medro idade = 5 }
Pessoa{ nome = Pedro idade = 10 }
Pessoa{ nome = Tina idade = 15 }
Pessoa{ nome = Zana idade = 30 }
```

Fazendo o mesmo exemplo usando classe anônima

```
public class ComparatorLambda {
    public static void main(String[] args) {
       ArrayList<Pessoa> pessoas = new ArrayList(10);
        Pessoa p;
        pessoas.add(new Pessoa("Pedro", 10));
        pessoas.add(new Pessoa("Medro", 5));
        pessoas.add(new Pessoa("Ane", 20));
        pessoas.add(new Pessoa("Zana", 30));
        pessoas.add(new Pessoa("Tina", 15));
        Collections.sort(pessoas, new Comparator<Pessoa>(){
            @Override
            public int compare(Pessoa o1, Pessoa o2) {
                return o1.getNome().compareTo(o2.getNome());
            }
        });
        Iterator<Pessoa> it = pessoas.iterator();
        while (it.hasNext())  {
            p = it.next();
            System.out.println(p);
        }
    }
}
```

Fazendo o mesmo exemplo usando funções lambda

```
public class ComparatorLambda {
    public static void main(String[] args) {
       ArrayList<Pessoa> pessoas = new ArrayList(10);
        Pessoa p;
        pessoas.add(new Pessoa("Pedro", 10));
        pessoas.add(new Pessoa("Medro", 5));
        pessoas.add(new Pessoa("Ane", 20));
        pessoas.add(new Pessoa("Zana", 30));
        pessoas.add(new Pessoa("Tina", 15));
        Collections.sort(pessoas, (Pessoa o1, Pessoa o2) ->
              o1.getNome().compareTo(o2.getNome()));
        Iterator<Pessoa> it = pessoas.iterator();
        while (it.hasNext())  {
            p = it.next();
            System.out.println(p);
        }
    }
}
```

Dúvidas?