

26

Relações de ordem

Relações de ordem entre elementos



27

- Para ordenar uma colecção de objetos de tipo E
 - Implementar interface **Comparable**<E>
 - Mecanismo de comparação natural
 - Método `compareTo`
 - Implementar interface **Comparator**<E>
 - Caso sejam necessários múltiplos critérios de ordenação especializados
 - Método `compare`

Interface Comparable<E>

28

- Quando uma classe necessita de definir uma relação de ordem para os seus elementos, temos a possibilidade de implementar a interface **Comparable**, ou seja, definir um comparador
 - Com a implementação do método **compareTo**, uma instância passa a dispor de um mecanismo de ordem natural relativamente a outro elemento
- Resultado do método **compareTo**
 - Se negativo, então a instância é menor do que o objecto em argumento
 - Se zero, a instância é igual ao objecto em argumento
 - Se positivo, a instância é maior do que o objecto em argumento
- Este mecanismo de comparação natural pode ser utilizado num método de ordenação
 - Utilizado implicitamente no método `sort(List<T> list)` da classe `java.util.Collections`
- Conceito muito útil em colecções

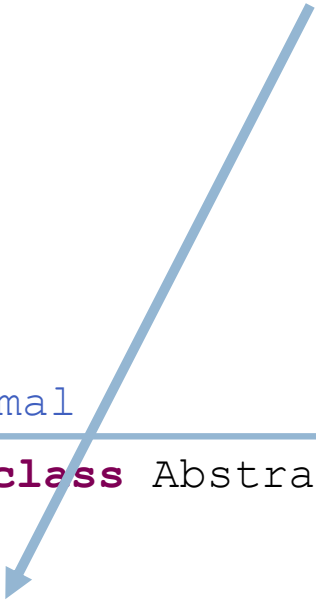
	<code>Comparable<T></code>
	<code>compareTo(o: T): int</code>

Ordenar a colecção de animais

29

```
public interface Animal extends Comparable<Animal>{  
    /**  
     * Devolve o nome do animal  
     * @return nome do animal  
     */  
    public String getName();  
  
    /**  
     * Devolve o "falar" do animal  
     * @return  
     */  
    public Str  
}
```

```
public abstract class AbstractAnimal implements Animal {  
    ...  
    public int compareTo(Animal other) {  
        //Ordenação por ordem alfabética de nome  
        return this.getName().compareTo(other.getName());  
    }  
}
```



Ordenar a colecção de animais

30

```
public class ZooClass implements Zoo {  
  
    private List<Animal> animals;  
  
    public ZooClass() {  
        animals = new LinkedList<>();  
    }  
  
    ...  
  
    public Iterator<Animal> listAnimalsByName() {  
        Collections.sort(animals);  
        return animals.iterator();  
    }  
}
```

ordenamos a lista usando o sort
da classe Collections

os elementos são ordenados com
base na ordem natural (definida
pelo método compareTo)

Interface Comparator<E>

31

- Tal como na interface **Comparable<T>**, a interface **Comparator<T>** de `java.util` permite definir uma função de comparação, especificando uma relação de ordem total entre os elementos de uma colecção
 - Suporta a implementação de múltiplos critérios de ordenação especializados
 - Existem dois métodos a implementar, **compare** e **equals**
- Resultado do método **compare**
 - Se negativo, então o primeiro é menor do que o segundo
 - Se zero, o primeiro é igual ao segundo
 - Se positivo, o primeiro é maior do que o segundo
- Analogamente, um comparador deste tipo pode ser utilizado num método de ordenação
 - Ex: método `sort` da classe `Collections`
- Note-se que a ordem imposta pelo **compare** deve ser consistente com o método **equals**

Comparator<T>	
compare(o1: T, o2: T): int	
equals(obj: Object): boolean	

Ordenar a colecção de animais por múltiplos critérios

32

```
public class ZooClass implements Zoo {  
  
    private List<Animal> animals;  
  
    public ZooClass() {  
        animals = new LinkedList<>();  
    }  
  
    ...  
  
    public Iterator<Animal> listAnimalsBySpecies() {  
        Collections.sort(animals, new ComparatorBySpecies());  
        return animals.iterator();  
    }  
}
```

os elementos são ordenados com base num comparador

Ordenar a colecção de animais por múltiplos critérios

33

```
public class ComparatorBySpecies implements Comparator<Animal> {

    @Override
    public int compare(Animal o1, Animal o2) {
        //primeiro cães, desempate usando ordem alfabética do nome

        boolean o1IsDog = o1 instanceof Dog;
        boolean o2IsDog = o2 instanceof Dog;
        if (o1IsDog && !o2IsDog) // o1; o2
            return -1;

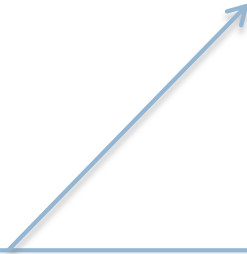
        if (!o1IsDog && o2IsDog) // o2; o1
            return 1;
        // (o1IsDog && o2IsDog) || são de outro tipo de animais
        return o1.getName().compareTo(o2.getName());
    }
}
```


Ordenar a colecção de animais por múltiplos critérios

34

```
public class ZooClass implements Zoo {  
  
    private List<Animal> animals;  
  
    public ZooClass() {  
        animals = new LinkedList<>();  
    }  
  
    ...  
  
    public Iterator<Animal> listAnimalsBySpecies(Comparator<Animal> c)  
    {  
        Collections.sort(animals, c);  
        return animals.iterator();  
    }  
}
```

comparador recebido como
parâmetro



Interface SortedMap<K, V>

35

- SortedMap é um Map que mantém as suas entradas em ordem ascendente, e de acordo com a ordem natural das chaves, ou então de acordo com o comparador que tenha sido fornecido aquando da sua criação
- Bastante utilizado em colecções ordenadas de pares chave-valor, como é o caso de dicionários e listas telefónicas
- Para além das operações de Map, permite ainda
 - Aplicar operações a zonas delimitadas do Map
 - Devolver a primeira e a última chave do mapa ordenado
 - Devolver o iterador que percorre os elementos em ordem ascendente
 - Devolver o comparador utilizado na ordenação, se este existir

```
public interface SortedMap<K, V> extends Map<K, V>
```

Interface SortedMap<K, V>

36

Métodos

K firstKey()	Devolve a primeira chave (mais pequena) do mapa
K lastKey()	Devolve a última chave (maior) do mapa
SortedMap<K,V> headMap(K toKey)	Devolve uma vista para o mapa só com pares cuja chave é menor que toKey
SortedMap<K,V> subMap (K fromKey, K toKey)	Devolve uma vista para o mapa só com pares cuja chave é maior ou igual a fromKey e menor que toKey
SortedMap<K,V> tailMap(K fromKey)	Devolve uma vista para o mapa só com pares cuja chave é maior ou igual a fromKey
Comparator<? super K> comparator()	Devolve o comparador usado na ordenação (se existir)

Classe TreeMap<K, V>

37

- A classe TreeMap é uma implementação da interface SortedMap baseada numa árvore binária de pesquisa
 - Permite também null como valor e como chave
 - Também suporta acesso eficiente
 - Garante ordem

```
public class TreeMap<K,V> extends AbstractMap<K,V>  
                                implements NavigableMap<K,V>, ...
```

K, o tipo das chaves **V**, o tipo dos valores mapeados

Conjuntos



Interface Set<E>

39

- Um set modela o conceito matemático de conjunto
 - Ex: baralho de cartas
- A interface Set estende Collection com as seguintes restrições
 - Não são admitidos elementos duplicados
 - Não estabelece ordem entre elementos
- A interface não adiciona métodos para além dos que são herdados de Collection



```
public interface Set<E> extends Collection<E>
```

Interface Set<E>

40

Métodos

<code>boolean add(E e)</code>	Adiciona o elemento indicado ao conjunto se ainda não existir no conjunto
<code>boolean remove(Object o)</code>	Remove o elemento indicado do conjunto, se este existir no conjunto
<code>void clear()</code>	Remove todos os elementos do conjunto
<code>boolean contains (Object o)</code>	Devolve true se o conjunto contém o elemento indicado
<code>boolean isEmpty()</code>	Devolve true se o conjunto não contém elementos
<code>int size()</code>	Devolve o número de elementos do conjunto
<code>Iterator<E> iterator()</code>	Devolve um iterador sobre os elementos do conjunto

Classe HashSet<E>

41

- A classe HashSet é uma implementação geral e eficiente da interface Set, baseada numa tabela de dispersão
 - Permitem também null como valor e como chave
 - Não garante a ordem de iteração ao longo do tempo

```
public class HashSet<E> extends AbstractSet<E>  
    implements Set<E>, ...
```


Interface SortedSet<E>

42

- SortedSet é um conjunto que permite estabelecer o conceito de ordem total entre os seus elementos
- Os elementos são ordenados de acordo com a sua ordem natural, através da implementação da interface Comparable, ou então de acordo com o comparador que tenha sido fornecido aquando da sua criação
- As novas operações que utilizam a ordenação são as seguintes:
 - Aplicar operações a zonas delimitadas do Set
 - Devolver o primeiro e o último elemento do conjunto ordenado
 - Devolver o iterador que percorre o conjunto por ordem ascendente
 - Devolver o comparador utilizado na ordenação, se este existir

```
public interface SortedSet<E> extends Set<E>
```

Interface SortedSet<E>

43

Métodos

E first()

Devolve o primeiro elemento (mais pequeno) do conjunto

E last ()

Devolve o último elemento (maior) do conjunto

SortedSet<E> headSet(E toElem)

Devolve uma vista para o conjunto só com elementos menores que toElem

SortedSet<E> subSet(E fromElem, E toElem)

Devolve uma vista para o conjunto só com elementos maiores ou iguais a fromElem e menores que toElem

SortedSet<E> tailSet(E fromElem)

Devolve uma vista para o conjunto só com elementos maiores ou iguais a fromElem

Comparator<? super E> comparator()

Devolve o comparador usado na ordenação (se existir)

Classe TreeSet<E>

44

- A classe TreeSet é uma implementação geral e eficiente da interface SortedSet, baseada numa árvore binária de pesquisa
 - Permitem também null como valor e como chave
 - Garante a ordem de iteração ao longo do tempo

```
public class TreeSet<E> extends AbstractSet<E>  
    implements SortedSet<E>, ...
```

Java Framework

45

			Implementações			
			Resizable array	Linked list	Balanced tree	Hash table
Interfaces	Collection	Set				HashSet
		SortedSet			TreeSet	
		List	ArrayList	LinkedList		
		Map				HashMap
		SortedMap			TreeMap	

DIRECT URL

Java Framework

46

