

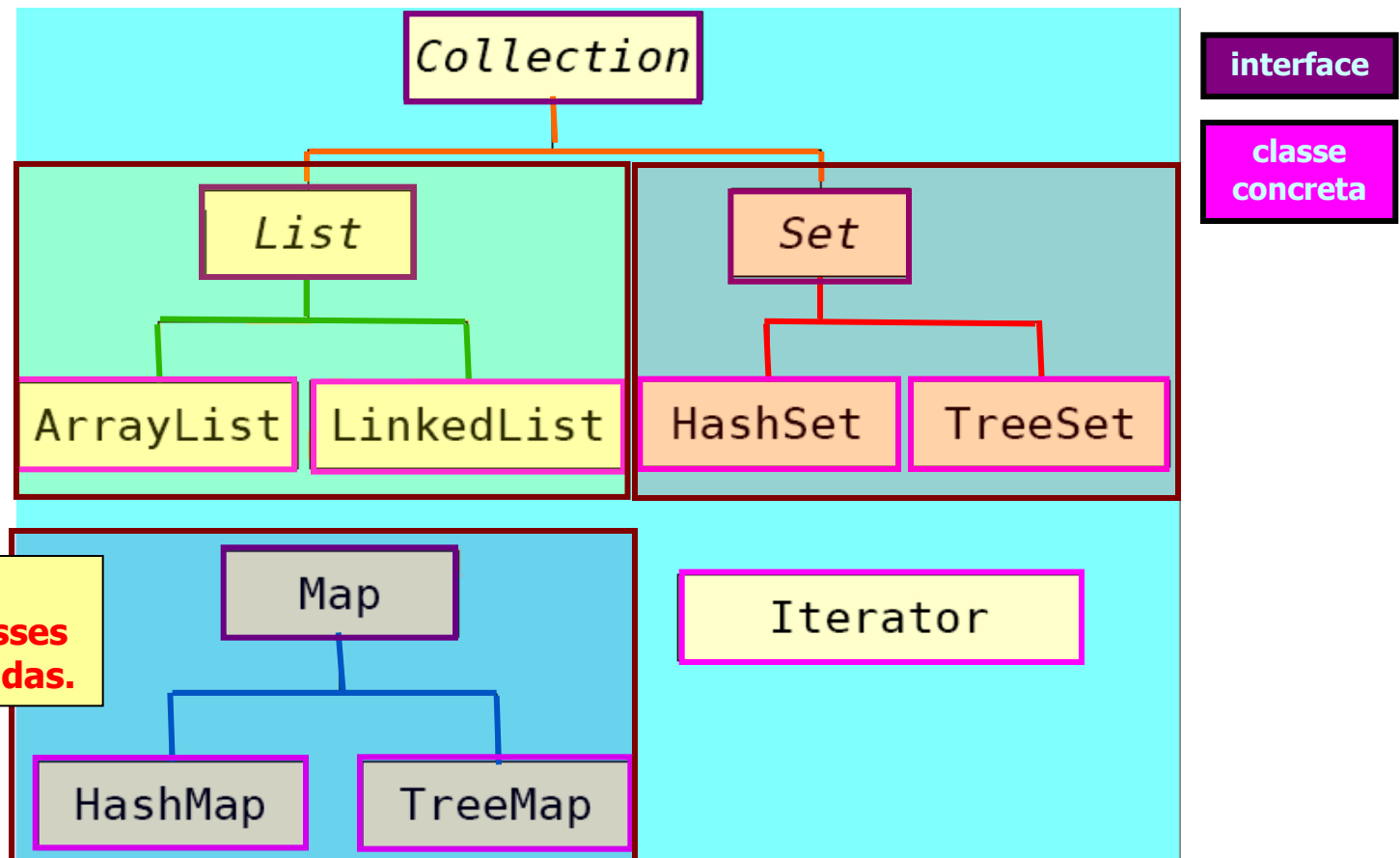
Ordenação de coleções

Prof. Vinicius Rosalen

Java Collection Framework

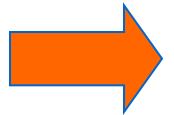
Hierarquia das Interfaces e suas Classes

Os elementos que compreendem a estrutura de coleções estão no pacote `java.util`.



Ordenação de coleções

- Blz,
 - Uma vez entendido todas essas coisas...
 - Vamos conversar um pouco sobre o nosso foco de hoje que é sobre a ordenação dessas coleções de objetos....



Ordenação de coleções

→ Java já implementa alguns algoritmo de ordenação:

- Coleções ordenadas: TreeSet, TreeMap;
- Collections.sort() para coleções;
- Arrays.sort() para vetores.

→ Por exemplo

```
1  import java.util.*;
2
3  public class Ordem1 {
4      public static void main(String args[]) {
5
6          List lista = new ArrayList();
7          lista.add("Sergio");
8          lista.add("Vinicius");
9          lista.add("Paulo");
10         lista.add("Guilherme");
11
12         System.out.println(lista);
13         Collections.sort(lista);
14         System.out.println(lista);
15
16     }
17 }
```

[Sergio, Vinicius, Paulo, Guilherme]
[Guilherme, Paulo, Sergio, Vinicius]

→ Observe que primeiro a lista é impressa na ordem de inserção e, depois de chamar o sort, ela é impressa em ordem alfabética

Ordenação de coleções

➡ Mas toda coleção em Java pode ser de qualquer tipo de objeto

– Por exemplo: ContaCorrente.

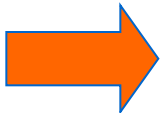
- ➡ E se quisermos ordenar uma lista de ContaCorrente?
- ➡ Em que ordem a classe Collections ordenará?
- ➡ Pelo saldo? Pelo nome do correntista?

➡ Sempre que falamos em ordenação, precisamos pensar em um **critério de ordenação**,

- Uma forma de determinar qual elemento vem antes de qual.

➡ É necessário instruir o Java sobre como **comparar** nossas Objetos

- A fim de determinar uma ordem na coleção.



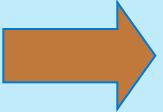
Ordenação de coleções - Comparable

➡ Para que esse tipo de ordenação funcione,
– É preciso que os objetos implementem a interface Comparable

➡ Esta interface define a operação de comparação do próprio objeto com outro,
– Usado para definir a ordem natural dos elementos de uma coleção.

➡ Define o método `compareTo(Object obj)`:
➡ Compara o objeto atual (this) com o objeto informado (obj);

- Retorna 0 se `this = obj`;
- Retorna um número negativo se `this < obj`;
- Retorna um número positivo se `this > obj`.



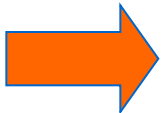
Ordenação de coleções – Comparable -

➡ O que o código está fazendo?

➡ E se quisermos ordenar pela idade?

**Analisar e
Implementar
juntos**

```
1  import java.util.*;
2
3  class Pessoa implements Comparable {
4      private String nome;
5      private int idade;
6
7      public Pessoa(String n, int i) {
8          nome = n; idade = i;
9      }
10
11     public String toString() {
12         return nome + ", " + idade + " ano(s)";
13     }
14
15     public int compareTo(Object o) {
16         return nome.compareTo(((Pessoa)o).nome);
17     }
18 }
19
20 public class Ordem2 {
21     public static void main(String[] args) {
22
23         List pessoas = new ArrayList();
24         pessoas.add(new Pessoa("Fulano", 20));
25         pessoas.add(new Pessoa("Beltrano", 18));
26         pessoas.add(new Pessoa("Cicrano", 23));
27
28         Collections.sort(pessoas);
29
30         for (Object o : pessoas)
31             System.out.println(o);
32     }
33 }
34
```



Ordenação de coleções - Comparable

O que o código está fazendo?

```
1  import java.util.*;
2
3  class Pessoa implements Comparable {
4      private String nome;
5      private int idade;
6
7      public Pessoa(String n, int i) {
8          nome = n; idade = i;
9      }
10
11     public String toString() {
12         return nome + ", " + idade + " ano(s)";
13     }
14
15     public int compareTo(Object o) {
16         // return nome.compareTo(((Pessoa)o).nome);
17         if(this.idade < ((Pessoa)o).idade) {
18             return -1;
19         }
20         if(this.idade > ((Pessoa)o).idade) {
21             return 1;
22         }
23         return 0;
24     }
25 }
26
27 public class Ordem2 {
28     public static void main(String[] args) {
29
30         List pessoas = new ArrayList();
31         pessoas.add(new Pessoa("Fulano", 20));
32         pessoas.add(new Pessoa("Beltrano", 18));
33         pessoas.add(new Pessoa("Cicrano", 23));
34
35         Collections.sort(pessoas);
36
37         for (Object o : pessoas)
38             System.out.println(o);
39     }
40 }
41
```


Implemente esse caso



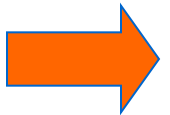
Ordenação de coleções - Comparable

Resumindo

 Com o código anterior, nossa classe tornou-se “**comparável**”


 Dados dois objetos da classe, conseguimos dizer se um objeto é maior, menor ou igual ao outro, segundo algum critério por nós definido.


- No nosso caso, a comparação foi feita baseando no nome e depois na idade.

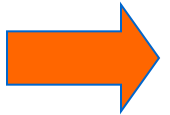


Ordenação de coleções - Comparable

Resumindo

-  **IMPORTANTE:** Repare que o critério de ordenação é **totalmente aberto, definido pelo programador.**
- Se quisermos ordenar por outro atributo (ou até por uma combinação de atributos), basta modificar a implementação do método `compareTo` na classe.

-  Agora sim, quando chamarmos o método `sort` de `Collections` ele saberá como fazer a ordenação da lista;
- Ele usará o critério que **definimos no método `compareTo`**



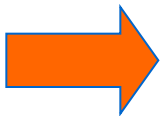
Ordenação de coleções - Comparator

➡ E o que acontece quando não podemos criar comparadores pois não posso mexer no código??

➡ Podemos utilizar a **interface Comparable** quando os objetos a serem adicionados não podem ser modificados....

– biblioteca de terceiros, por exemplo

➡ Importante: Esta interface define a operação de comparação entre dois objetos por um objeto externo.



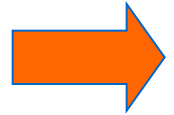
Ordenação de coleções - Comparator

➡ Para trabalhar com esse interface....

- É necessário implementar `java.util.Comparator`;

➡ Esta interface define o método `compare(Object a, Object b)` e retorna:

- Número negativo, se o primeiro $a < b$;
- Zero, se $a = b$;
- Número positivo se $a > b$.



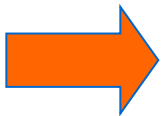
Ordenação de coleções - Comparator

➡ O que o código está fazendo?

➡ E se quisermos ordenar pelo nome?

Analisar e Implementar juntos

```
1  import java.util.*;
2
3  class Pessoa {
4      public String nome;
5      public int idade;
6
7      public Pessoa() {}
8
9      public Pessoa(String n, int i) {
10         nome = n; idade = i;
11     }
12
13     public String toString() {
14         return nome + ", " + idade + " ano(s)";
15     }
16 }
17
18 class PessoaComparator implements Comparator {
19
20     //Número negativo, se o1 < o2;
21     //Zero, se o1 = o2;
22     //Número positivo se o1 > o2.
23     ➡ public int compare(Object o1, Object o2) {
24         return (((Pessoa)o1).idade - ((Pessoa)o2).idade);
25     }
26 }
27
28 public class Ordem3 {
29     ➡ public static void main(String[] args) {
30
31         List pessoas = new ArrayList();
32         pessoas.add(new Pessoa("Fulano", 20));
33         pessoas.add(new Pessoa("Beltrano", 18));
34         pessoas.add(new Pessoa("Cicrano", 23));
35
36         ➡ Collections.sort(pessoas, new PessoaComparator());
37
38         for (Object o : pessoas)
39             System.out.println(o);
40     }
41 }
```



Ordenação de coleções - Comparator

➡ O que o código está fazendo?

```
1  import java.util.*;
2
3  class Pessoa {
4      public String nome;
5      public int idade;
6
7      public Pessoa() {}
8
9      public Pessoa(String n, int i) {
10         nome = n; idade = i;
11     }
12
13     public String toString() {
14         return nome + ", " + idade + " ano(s)";
15     }
16 }
17
18 class PessoaComparator implements Comparator {
19     //Número negativo, se o1 < o2;
20     //Zero, se o1 = o2;
21     //Número positivo se o1 > o2.
22     public int compare(Object o1, Object o2) {
23         // return (((Pessoa)o1).idade - ((Pessoa)o2).idade);
24         ➡ return ((Pessoa)o1).nome.compareTo(((Pessoa)o2).nome);
25     }
26 }
27
28
29 public class Ordem3 {
30     public static void main(String[] args) {
31
32         List pessoas = new ArrayList();
33         pessoas.add(new Pessoa("Fulano", 20));
34         pessoas.add(new Pessoa("Beltrano", 18));
35         pessoas.add(new Pessoa("Cicrano", 23));
36
37         ➡ Collections.sort(pessoas, new PessoaComparator());
38         for (Object o : pessoas)
39             System.out.println(o);
40     }
41 }
42
43
```

Implemente esse caso

Tipos Genéricos e ordenação

➡ **Dica...** Outra coisa legal que podemos fazer é:

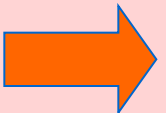
➡ Definir a ordenação dos objetos para o tipo de objetos que estamos trabalhando.

➡ Antes

```
➡ class Pessoa implements Comparable {  
    private String nome;  
    ➡ public int compareTo(Object o) {  
        ➡ Pessoa p = (Pessoa)o;  
        return nome.compareTo(p.nome);  
    }  
}
```

➡ Depois

```
// Com generics:  
➡ class Pessoa implements Comparable<Pessoa> {  
    private String nome;  
    ➡ public int compareTo(Pessoa o) {  
        return nome.compareTo(o.nome);  
    }  
}
```



Tipos Genéricos e ordenação

➡ Outro exemplo de Genérico usando Comparable

➡ Observe que o método compareTo recebe um objeto ContaCorrente e não um Object

```
➡ public class ContaCorrente extends Conta implements Comparable<ContaCorrente>  
{
```

```
➡ public int compareTo(ContaCorrente outra) {
```

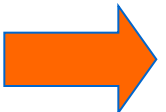
```
    if(this.saldo < outra.saldo) {  
        return -1;  
    }
```

```
    if(this.saldo > outra.saldo) {  
        return 1;  
    }
```

```
    return 0;
```

```
}
```

```
}
```



Tipos Genéricos e ordenação

➡ Modifique os códigos anteriores para que seja definido o tipo de objetos que estamos trabalhando.

Implemente os casos

```
|class Pessoa implements Comparable {  
    private String nome;  
    private int idade;
```

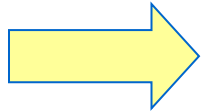
```
|class Pessoa {  
    public String nome;  
    public int idade;
```

```
|class PessoaComparator implements Comparator {
```

**Analisar e
Implementar
juntos**

Exercícios

- Blz... Agora é hora de exercitar.....
- Tente resolver os seguintes problemas...
 - Individual ou em grupo
 - Apresentar ao professor no final da aula



Exercício

- Modifique a classe implementada em sala de aula com a solução “Comparator” para que seja possível escolher a ordenação por idade ou por nome
 - Como podemos implementar essa solução?

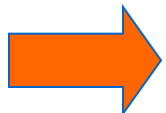
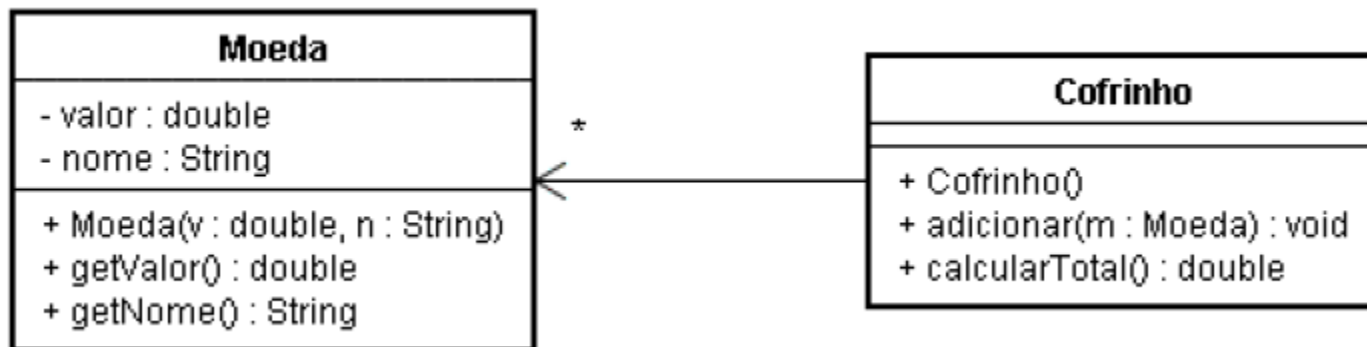
```
Lista ordenada por idade
Nome : Ana Beatriz Idade : 0 Sexo : Feminino
Nome : Vitor Hugo Idade : 25 Sexo : Masculino
Nome : Sirley Idade : 31 Sexo : Feminino

Lista ordenada por nome
Nome : Ana Beatriz Idade : 0 Sexo : Feminino
Nome : Sirley Idade : 31 Sexo : Feminino
Nome : Vitor Hugo Idade : 25 Sexo : Masculino
```

Exercício

- Implementar um cofrinho “fintech” de moedas com a capacidade de receber moedas e calcular o total depositado no cofrinho.
 - Implementa uma coleção de Moeda como uma lista.
 - Use o ArrayList
 - Faça um classe de teste

```
public class Cofrinho {  
    private List<Moeda> moedas;  
    public Cofrinho() {}  
    public .... get/setMoeda(...) {...}  
}
```



Exercício

- Altere a classe Cofrinho de modo que ela implemente métodos para:
 - Contar o número de moedas armazenadas
 - Contar o número de moedas de um determinado valor
 - Imprimir todo o conteúdo do Cofre
 - Ordenar e Informar qual a moeda de menor e maior valor
 - Imprima uma listagem por ordem crescente e outra em ordem decrescente de valor