

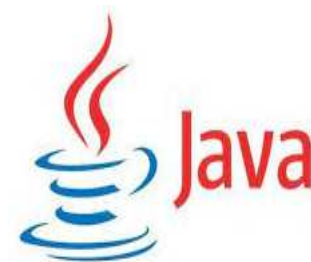


Residência em Software

Residência em Tecnologia da Informação e Comunicação Coleções – 3ª Partes

Professor:

Alvaro Degas Coelho



INSTITUIÇÃO EXECUTORA



COORDENADORA



APOIO



Algoritmos (muito) Úteis

- A classe Collections disponibiliza uma série de algoritmos
 - Todos implementados com desempenho otimizado
 - Todos polimórficos
 - Aplicáveis a diferentes tipos de objetos
 - Em alguns casos é necessário se cumprir pré-condições.
 - Sempre implementados de modo estático

Os Algoritmos

- Aplicáveis a objetos que implementam a Interface List
 - sort
 - binarySearch
 - reverse
 - shuffle
 - fill
 - copy

Os Algoritmos

- Aplicáveis a objetos que implementam a Interface Collections
 - min
 - max

sort

- Ordena elementos de uma lista
 - `Collections.sort(list)` ;
- A ordem é determinada pelo critério “natural” de ordenação do tipo do dado
 - Usa o método `compareTo` (retorna 0, negativo, positivo)
 - Você pode implementar em uma classe de sua autoria!
- Outra opção
- `Collections.sort(list, comparator)`
 - Onde `comparator` é um objeto do tipo `Comparator`
 - Pesquise!

Exemplo

```
1 // Sort1.java
2 // Usando o sort.
3 import java.util.*;
4
5 public class Sort1 {
6     private static final String suits[] =
7         { "Hearts", "Diamonds", "Clubs", "Spades" };
8
9     // mostrar os elementos
10    public void printElements()
11    {
12        // criando um arraylist
13        List list = new ArrayList( Arrays.asList( suits ) );
14
15        // gerando a lista ordenada
16        System.out.println( "Unsorted array elements:\n" + list );
17
18        Collections.sort( list ); // ordenando o ArrayList
19
20        // mostrando os elementos
21        System.out.println( "Sorted array elements:\n" + list );
22    }
23
```

Exemplo

```
1 // Sort1.java
2 // Usando o sort.
3 import java.util.*;
4
5 public class Sort1 {
6     private static final String suits[] =
7         { "Hearts", "Diamonds", "Clubs", "Spades" };
8
9     // mostrar os elementos
10    public void printElements()
11    {
12        // criando um arraylist
13        List list = new ArrayList( Arrays.asList( suits ) );
14
15        // gerando a lista ordenada
16        System.out.println( "Unsorted array elements:\n" + list );
17
18        Collections.sort( list ); // ordenando o ArrayList
19
20        // mostrando os elementos
21        System.out.println( "Sorted array elements:\n" + list );
22    }
23
```

Cria um ArrayList

Ordena o ArrayList

Exemplo (método main)

```
24     public static void main( String args[] )
25     {
26         new Sort1().printElements();
27     }
28
29 } // end class Sort1
```


Sort com comparator

- Um comparator é um objeto que permite que dois outros objetos sejam comparados
 - `a.compareTo(b)` ou `compare(a,b)`
 - Retorna: negativo se $a < b$, 0 se $a == b$ e positivo se $a > b$
 - Podemos criar comparators
 - Podemos implementar a interface `comparable`
 - O que nos força a implementar o método `compareTo()`

Exemplo

```
1 // Fig. 22.7: Sort2.java
2 // sort com comparator
3 import java.util.*;
4
5 public class Sort2 {
6     private static final String suits[] =
7         { "Hearts", "Diamonds", "Clubs", "Spades" };
8
9     // escrevendo a lista de elementos
10    public void printElements()
11    {
12        List list = Arrays.asList( suits ); // cria a lista
13
14        // escreve os elementos
15        System.out.println( "Unsorted array elements:\n" + list );
16
17        // ordena em ordem DECRESCENTE (usando um comparator)
18        Collections.sort( list, Collections.reverseOrder() );
19
20        // escreve os elementos
21        System.out.println( "Sorted list elements:\n" + list );
22    }
23
```

Exemplo

```
1 // Fig. 22.7: Sort2.java
2 // sort com comparator
3 import java.util.*;
4
5 public class Sort2 {
6     private static final String suits[] =
7         { "Hearts", "Diamonds", "Clubs", "Spades" };
8
9     // escrevendo a lista de elementos
10    public void printElements()
11    {
12        List list = Arrays.asList( suits ); // cria a lista
13
14        // escreve os elementos
15        System.out.println( "Unsorted array elements:\n" + list );
16
17        // ordena em ordem DECRESCENTE (usando um comparator)
18        Collections.sort( list, Collections.reverseOrder() );
19
20        // escreve os elementos
21        System.out.println( "Sorted list" );
22    }
23
```

O método `reverseOrder()` retorna um comparator que ordena a lista no caminho inverso

O método `sort` usa o comparator para ordenar a lista

Exemplo (método main)

```
• 24     public static void main( String args[] )  
• 25     {  
• 26         new Sort2().printElements();  
• 27     }  
• 28  
29 } // sort2
```

Outro Exemplo com Comparator

```
1  // Sort3.java
2  // Um objeto comparator
3  import java.util.*;
4
5  public class Sort3 {
6
7      public void printElements()
8      {
9          List list = new ArrayList(); // create List
10
11         list.add( new Time2( 6, 24, 34 ) );
12         list.add( new Time2( 18, 14, 05 ) );
13         list.add( new Time2( 8, 05, 00 ) );
14         list.add( new Time2( 12, 07, 58 ) );
15         list.add( new Time2( 6, 14, 22 ) );
16
17         //Listando elementos
18         System.out.println( "Unsorted array elements:\n" + list );
19
20         // ordenando com comparator
21         Collections.sort( list, new TimeComparator() );
22
23         // listando elementos
24         System.out.println( "Sorted list elements:\n" + list );
25     }
26
```

Exemplo

```
1 // Sort3.java
2 // Um objeto comparador
3 import java.util.*;
4
5 public class Sort3 {
6
7     public void printElements()
8     {
9         List list = new ArrayList(); // create List
10
11         list.add( new Time2( 6, 24, 34 ) );
12         list.add( new Time2( 18, 14, 05 ) );
13         list.add( new Time2( 8, 05, 00 ) );
14         list.add( new Time2( 12, 07, 58 ) );
15         list.add( new Time2( 6, 14, 22 ) );
16
17         //Listando elementos
18         System.out.println( "Unsorted array elements:" );
19
20         // ordenando com comparador
21         Collections.sort( list, new TimeComparator() );
22
23         // listando elementos
24         System.out.println( "Sorted list elements:\n" + list );
25     }
26 }
```

Ordena os elementos usando
um objeto do tipo
TimeComparator.

Exemplo

```
27  public static void main( String args[] )  
28  {  
29      new Sort2().printElements();  
30  }  
31
```

```
32 private class TimeComparator implements Comparator {
33     int hourCompare, minuteCompare, secondCompare;
34     Time2 time1, time2;
35
36     public int compare(Object object1, Object object2)
37     {
38         // faz o cast dos objetos como Time2
39         time1 = (Time2)object1;
40         time2 = (Time2)object2;
41
42         hourCompare = new Integer( time1.getHour() ).compareTo(
43             new Integer( time2.getHour() ) );
44
45         // se a hora for diferente, já decidiu
46         if ( hourCompare != 0 )
47             return hourCompare;
48
49         minuteCompare = new Integer( time1.getMinute() ).compareTo(
50             new Integer( time2.getMinute() ) );
51
52         // se não, testa os minutos
53         if ( minuteCompare != 0 )
54             return minuteCompare;
55
56         secondCompare = new Integer( time1.getSecond() ).compareTo(
57             new Integer( time2.getSecond() ) );
58
59         return secondCompare; // senão, testa os segundos
60     }
61
62     } // TimeComparator
63
64 } // Sort3
```



```
32 private class TimeComparator implements Comparator {
33     int hourCompare, minuteCompare, secondCompare;
34     Time2 time1, time2;
35
36     public int compare(Object object1, Object object2)
37     {
38         // faz o cast dos objetos como Time2
39         time1 = (Time2)object1;
40         time2 = (Time2)object2;
41
42         hourCompare = new Integer( time1.getHour() ).compareTo(
43             new Integer( time2.getHour() ) );
44
45         // se a hora for diferente, já decidiu
46         if ( hourCompare != 0 )
47             return hourCompare;
48
49         minuteCompare = new Integer( time1.getMinute() ).compareTo(
50             new Integer( time2.getMinute() ) );
51
52         // se não, testa os minutos
53         if ( minuteCompare != 0 )
54             return minuteCompare;
55
56         secondCompare = new Integer( time1.getSecond() ).compareTo(
57             new Integer( time2.getSecond() ) );
58
59         return secondCompare; // senão, testa os segundos
60     }
61 } // TimeComparator
62
63
64 } // Sort3
```

Classe TimeComparator
implementa a interface
Comparator

Implementa o método compare
para determinar a ordem entre dois
objetos

shuffle

- Ordena elementos aleatoriamente
 - Pseudo aleatoriamente
 - Porque? Pesquise!

Exemplo

```
1 // Cards.java
2 // Usando shuffle.
3 import java.util.*;
4
5 // classe para representar uma carta num baralho
6 class Card {
7     private String face;
8     private String suit;
9
10    // construtor
11    public Card( String initialface, String initialSuit )
12    {
13        face = initialface;
14        suit = initialSuit;
15    }
16
17    // get da face
18    public String getFace()
19    {
20        return face;
21    }
22
23    // get do naipe
24    public String getSuit()
25    {
26        return suit;
27    }
```

Exemplo

```
28
29 // String descrevendo a carta (3 de paus, 4 de espadas...)
30 public String toString()
31 {
32     StringBuffer buffer = new StringBuffer( face + " de " + suit );
33     buffer.setLength( 20 );
34
35     return buffer.toString();
36 }
37
38 } Card
39
40 // Classe Cards
41 public class Cards {
42     private static final String suits[] =
43         { "Copas", "Paus", "Ouros", "Espadas" };
44     private static final String faces[] = { "As", "Duque", "Terno",
45         "Quadra", "Quina", "Sena", "Sete", "Oito", "Nove", "Dez",
46         "Valete", "Dama", "Rei" };
47     private List list;
48
49     // construtor para criar e embaralhar um baralho
50     public Cards()
51     {
52         Card deck[] = new Card[ 52 ];
53
```

Exemplo

```
54     for ( int count = 0; count < deck.length; count++ )
55         deck[ count ] = new Card( faces[ count % 13 ],
56             suits[ count / 13 ] ); //13 valores e 4 naipes
57
58     list = Arrays.asList( deck ); // cria a lista
59     Collections.shuffle( list ); // shuffle deck
60 }
61
62 // mostrar o baralho "cortado" em dois maços
63 public void printCards()
64 {
65     int half = list.size() / 2 - 1;
66     int j = half+1;
67     for ( int i = 0; i <= half; i++)
68         System.out.println( list.get( i ).toString() + list.get( j ) );
69     j++;
70 }
71 public static void main( String args[] )
72 {
73     new Cards().printCards();
74 }
75
76 } // Cards
```

Exemplo

```
54     for ( int count = 0; count < deck.length; count++ )
55         deck[ count ] = new Card( faces[ count % 13 ],
56             suits[ count / 13 ] ); //13 valores e 4 naipes
57
58     list = Arrays.asList( deck ); // cria a lista
59     Collections.shuffle( list ); // shuffle deck
60 }
61
62 // mostrar o baralho "cortado" em dois maços
63 public void printCards()
64 {
65     int half = list.size() / 2 - 1;
66     int j = half+1;
67     for ( int i = 0; i <= half; i++)
68         System.out.println( list.get( i ).toString() + list.get( j ) );
69     j++;
70 }
71 public static void main( String args[] )
72 {
73     new Cards().printCards();
74 }
75
76 } // Cards
```

Usando o método shuffle para
“misturar” a lista

Reverse, fill, copy, max e min

- **reverse**
 - Reverte a ordem de elementos de uma lista
- **fill**
 - Preenche uma lista com algum valor
- **copy (dest, source)**
 - Cria uma cópia (delimitada) de uma lista
- **max**
 - Retorna o maior elemento de uma Collection
- **min**
 - Retorna o menor elemento de uma Collection
- **max e min** podem ser chamados com um comparador passado como segundo parâmetro

Binary Search

- Busca binária
- Collections ordenadas
- Locates **objetos** em listas
 - Retorna o índice do objeto, se ele for encontrado
 - Retorna valor negativo se não for
- Opções de uso (sobrecarga)
 - `Collections.binarySearch(list, key)`
 - `Collections.binarySearch(list, key, comparator)`

Exemplo

```
// BinarySearchTest.java
2 // Uso de binarySearch.
3 import java.util.*;
4
5 public class BinarySearchTest {
6     private static final String colors[] = { "vermelho", "branco",
7         "azul", "preto", "amarelo", "rosa", "verde", "marrom" };
8     private List list;    // lista de referência
9
10    // criar e ordenar a lista
11    public BinarySearchTest()
12    {
13        list = new ArrayList( Arrays.asList( colors ) );
14        Collections.sort( list ); // ordena o ArrayList
15        System.out.println( "ArrayList Ordenado: " + list );
16    }
```

Exemplo

```
// BinarySearchTest.java
2 // Uso de binarySearch.
3 import java.util.*;
4
5 public class BinarySearchTest {
6     private static final String colors[] = { "vermelho", "branco",
7         "azul", "preto", "amarelo", "rosa", "verde", "marrom" };
8     private List list;    // lista de referência
9
10    // criar e ordenar a lista
11    public BinarySearchTest()
12    {
13        list = new ArrayList( Arrays.asList( colors ) );
14        Collections.sort( list ); // ordena o ArrayList
15        System.out.println( "ArrayList Ordenado: " + list );
16    }
```

Ordena em ordem crescente



Exemplo

```
17
18 // procurando valores
19 private void printSearchResults()
20 {
21     printSearchResultsHelper( colors[ 4 ] ); // primeiro: amarelo
22     printSearchResultsHelper( colors[ 0 ] ); // vermelho: último
23     printSearchResultsHelper( colors[ 7 ] ); // marom (posição 3)
24     printSearchResultsHelper( "aardvark" ); // antes do primeiro
25     printSearchResultsHelper( "goat" ); // não existe
26     printSearchResultsHelper( "zebra" ); // não existe
27 }
28
```

Exemplo

```
29 // o método que busca elementos
30 private void printSearchResultsHelper( String key )
31 {
32     int result = 0;
33
34     System.out.println( "\nBuscando pelo item: " + key );
35     result = Collections.binarySearch( list, key );
36     System.out.println( ( result >= 0 ? "Encontrado na posicao " + result :
37         "Nao encontrado (" + result + ")" ) );
38 }
39
40 public static void main( String args[] )
41 {
42     new BinarySearchTest().printSearchResults();
43 }
44
45 } // BinarySearchTest
```

Exemplo

```
29 // o método que busca elementos
30 private void printSearchResultsHelper( String key )
31 {
32     int result = 0;
33
34     System.out.println( "\nBuscando pelo item: " + key );
35     result = Collections.binarySearch( list, key );
36     System.out.println( ( result >= 0 ? "Encontrado na posicao " + result :
37         "Nao encontrado (" + result + ")" ) );
38 }
39
40 public static void main( String args[] )
41 {
42     new BinarySearchTest().printSearchResults();
43 }
44
45 } // BinarySearchTest
```

Usa o método `binarySearch`
para localizar o elemento na
lista

sets

- Conjuntos
- `Collection` that contem elementos não duplicados
- `HashSet` (implementa `Set`)
 - Armazena os elementos numa tabela hash
 - A tabela “determina” uma ordem usando um algoritmo de hash
- `TreeSet` (implementa `SortedSet`)
 - Armazena os elementos numa árvore (ordenados)
 - `headSet(x)` retorna o subconjunto de todos os elementos antes de x
 - `tailSet(x)` retorna o subconjunto de todos os elementos depois de x (inclusive)

Exemplo

```
1 // SetTest.java
2 // Usar HashSet pra remover duplicatas.
3 import java.util.*;
4
5 public class SetTest {
6     private static final String colors[] = { "vermelho", "branco", "azul",
7         "verde", "cinza", "tangerina", "preto", "branco ", "rosa",
8         "amarelo", "cinza ", "tangerina" };
9
10    // criando um ArrayList
11    public SetTest()
12    {
13        List list = new ArrayList( Arrays.asList( colors ) );
14        System.out.println( "ArrayList: " + list );
15        printNonDuplicates( list );
16    }
17
18    // criando um conjunto para eliminar duplicatas
19    private void printNonDuplicates( Collection collection )
20    {
21        // criando um HashSet para poder usar um iterator
22        Set set = new HashSet( collection );
23        Iterator iterator = set.iterator();
24
25        System.out.println( "\nLista sem duplicacoes: " );
26
```

Exemplo

```
1 // SetTest.java
2 // Usar HashSet pra remover duplicatas.
3 import java.util.*;
4
5 public class SetTest {
6     private static final String colors[] = { "vermelho", "branco", "azul",
7         "verde", "cinza", "tangerina", "preto", "branco ", "rosa",
8         "amarelo", "cinza ", "tangerina" };
9
10    // criando um ArrayList
11    public SetTest()
12    {
13        List list = new ArrayList( Arrays.asList( colors ) );
14        System.out.println( "ArrayList: " + list );
15        printNonDuplicates( list );
16    }
17
18    // criando um conjunto para eliminar duplicatas
19    private void printNonDuplicates( Collection collection )
20    {
21        // criando um HashSet para poder usar um iterator
22        Set set = new HashSet( collection );
23        Iterator iterator = set.iterator();
24
25        System.out.println( "\nLista sem duplicacoes: " );
26
```

Cria um hashset a
partir do objeto
Collection

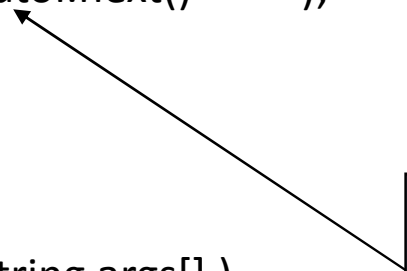
Exemplo

```
27     while ( iterator.hasNext() )
28         System.out.print( iterator.next() + " " );
29
30     System.out.println();
31 }
32
33 public static void main( String args[] )
34 {
35     new SetTest();
36 }
37
38 } // SetTest
```

Exemplo

```
27     while ( iterator.hasNext() )
28         System.out.print( iterator.next() + " " );
29
30     System.out.println();
31 }
32
33 public static void main( String args[] )
34 {
35     new SetTest();
36 }
37
38 } // SetTest
```

Usa o iterator para
percorrer a lista sem
duplicatas



Outro Exemplo

```
1 // SortedSetTest.java
2 // Usando TreeSet e SortedSet.
3 import java.util.*;
4
5 public class SortedSetTest {
6     private static final String names[] = { "vermelho", "branco", "azul",
7         "verde", "cinza", "tangerina", "preto", "branco ", "rosa",
8         "amarelo", "cinza ", "tangerina" };
9     // criando e usando um conjunto ordenado com TreeSet
10    public SortedSetTest()
11    {
12        SortedSet tree = new TreeSet( Arrays.asList( names ) );
13
14        System.out.println( "set: " );
15        printSet( tree );
16
17        // pegando o headSet anterior a "tangerina"
18        System.out.print( "\nheadSet (\" tangerina \"): " );
19        printSet( tree.headSet( "tangerina" ) );
20
21        // pegando tailSet a partir de "tangerina"
22        System.out.print( "tailSet (\"tangerina\"): " );
23        printSet( tree.tailSet( " tangerina" ) );
24
25        // primeiro e últimos elementos
26        System.out.println( "primeiro: " + tree.first() );
27        System.out.println( "ultimo : " + tree.last() );
28    }
29 }
```

Outro Exemplo

```
1 // SortedSetTest.java
2 // Usando TreeSet e SortedSet.
3 import java.util.*;
4
5 public class SortedSetTest {
6     private static final String names[] = { "vermelho", "branco", "azul",
7         "verde", "cinza", "tangerina", "preto", "branco", "rosa",
8         "amarelo", "cinza", "tangerina" };
9     // criando e usando um conjunto ordenado com TreeSet
10    public SortedSetTest()
11    {
12        SortedSet tree = new TreeSet( Arrays.asList( names ) );
13
14        System.out.println( "set: " );
15        printSet( tree );
16
17        // pegando o headSet anterior a "tangerina"
18        System.out.print( "\nheadSet (\" tangerina \"): " );
19        printSet( tree.headSet( "tangerina" ) );
20
21        // pegando tailSet a partir de "tangerina"
22        System.out.print( "\ntailSet (\"tangerina\"): " );
23        printSet( tree.tailSet( " tangerina " ) );
24
25        // primeiro e últimos elementos
26        System.out.println( "primeiro: " + tree.first() );
27        System.out.println( "ultimo : " + tree.last() );
28    }
29 }
```

Cria o TreeSet a partir do array

Usando headSet para listar os anteriores a "tangerina"

Usando tailSet pra listar os elementos a partir de "tangerina"

Métodos first e last para encontrar o primeiro e o último elementos

Exemplo

```
30 // o conjunto
31 private void printSet( SortedSet set )
32 {
33     Iterator iterator = set.iterator();
34
35     while ( iterator.hasNext() )
36         System.out.print( iterator.next() + " " );
37
38     System.out.println();
39 }
40
41 public static void main( String args[] )
42 {
43     new SortedSetTest();
44 }
45
46 } // SortedSetTest
```

Exemplo

Usando iterator para
percorrer os
elementos

```
30 // o conjunto
31 private void printSet( SortedSet set )
32 {
33     Iterator iterator = set.iterator();
34
35     while ( iterator.hasNext() )
36         System.out.print( iterator.next() + " " );
37
38     System.out.println();
39 }
40
41 public static void main( String args[] )
42 {
43     new SortedSetTest();
44 }
45
46 } // SortedSetTest
```

Exercício

- Crie uma lista (List) para armazenar instâncias de Itens de uma lista de compra.
 - Adicione alguns itens à lista, cada uma com um nome e um preço (float).
 - Implemente um método para exibir todas os itens da lista.
 - Implemente um método para encontrar e exibir todas os itens entre uma determinada faixa de valor.
 - Pense em uma sobrecarga para assumir 0 como limite inferior se apenas um valor for passado como parâmetro
 - Implemente um método que remova um item da lista com base em seu nome.
- Certifique-se de utilizar métodos apropriados da interface List para realizar essas operações.
- Uma classe ListaDeCompras deve conter a lista de itens e os métodos para manipulá-la.



Residência
em Software

Exercício

- Você está desenvolvendo um sistema de controle de membros de uma comunidade.

- Cada membro possui um nome e um número de identificação único.
- Implemente uma classe Membro com atributos para o nome e o número de identificação.
- Crie um conjunto (Set) para armazenar instâncias de Membro.
 - Adicione alguns membros ao conjunto, cada um com um nome e um número de identificação.
- Implemente um método para exibir todos os membros do conjunto.
 - Garanta que membros com o mesmo número de identificação não possam ser adicionados ao conjunto.
- Remova um membro do conjunto com base no número de identificação.
- Certifique-se de utilizar métodos apropriados da interface Set para realizar essas operações.
- A classe Comunidade deve conter o conjunto de membros e os métodos para manipulá-lo.
- Utilize a classe HashSet ou outra implementação de Set para armazenar os membros.