

- Uma **coleção** é um objeto que pode armazenar referências a outros objetos;
- Normalmente, coleções contêm referências a objetos que são todos do mesmo tipo;
- Pacote *java.util*.

Interface	Descrição
Collection	A interface-raiz na hierarquia de coleções a partir da qual as interfaces Set, Queue e List são derivadas.
Set	Uma coleção que não contém duplicatas.
List	Uma coleção ordenada que pode conter elementos duplicados.
Map	Associa chaves a valores e não pode conter chaves duplicadas.
Queue	Em geral, uma coleção primeiro a entrar, primeiro a sair que modela uma fila de espera; outras ordens podem ser especificadas.

- Uma **coleção** é um objeto que pode armazenar referências a outros objetos;
- Normalmente, coleções contêm referências a objetos que são todos do mesmo tipo;
- Pacote *java.util*.

Interface	Descrição
Collection	A interface-raiz na hierarquia de coleções a partir da qual as interfaces Set, Queue e List são derivadas.
Set	Uma coleção que não contém duplicatas.
List	Uma coleção ordenada que pode conter elementos duplicados.
Map	Associa chaves a valores e não pode conter chaves duplicadas.
Queue	Em geral, uma coleção primeiro a entrar, primeiro a sair que modela uma fila de espera; outras ordens podem ser especificadas.

- Permitem manipular valores de tipo primitivo como objetos;
- **Boolean, Byte, Character, Double, Float, Integer, Long** e **Short**;

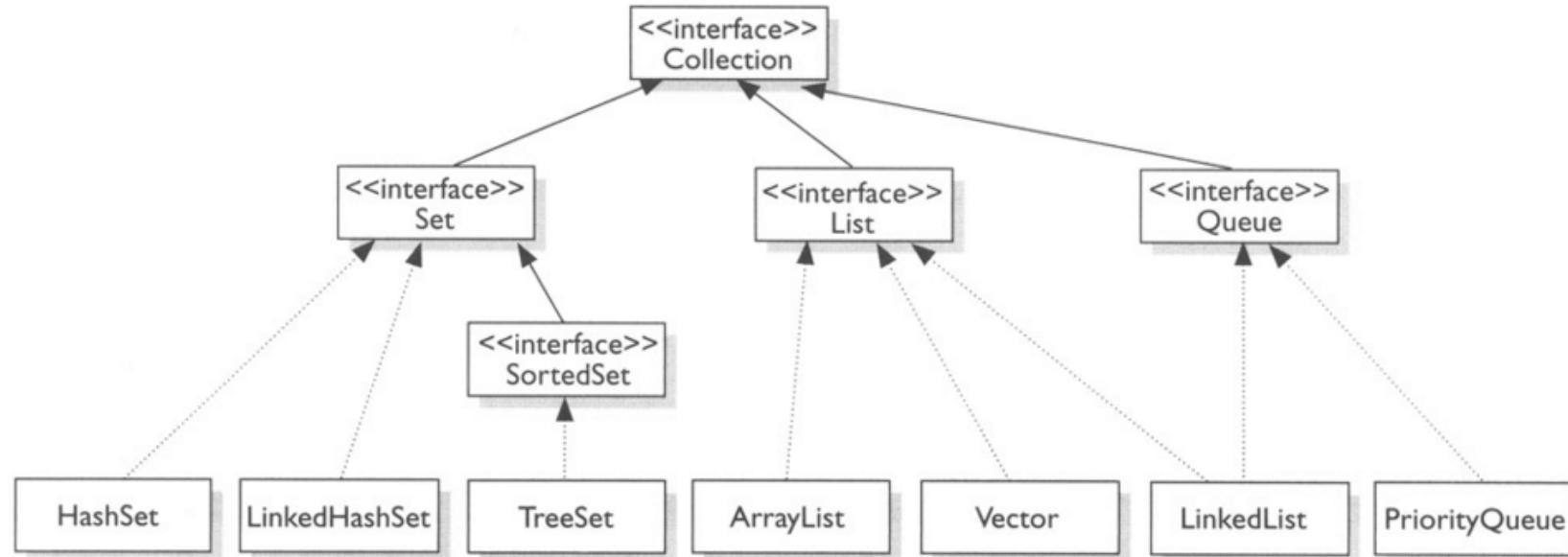
- Permitem manipular valores de tipo primitivo como objetos;
- **Boolean, Byte, Character, Double, Float, Integer, Long** e **Short**;
- Podem ser usados em coleções Java;

- Permitem manipular valores de tipo primitivo como objetos;
- **Boolean, Byte, Character, Double, Float, Integer, Long** e **Short**;
- Podem ser usados em coleções Java;
- Possuem alguns métodos que auxiliam na manipulação e conversão de dados;

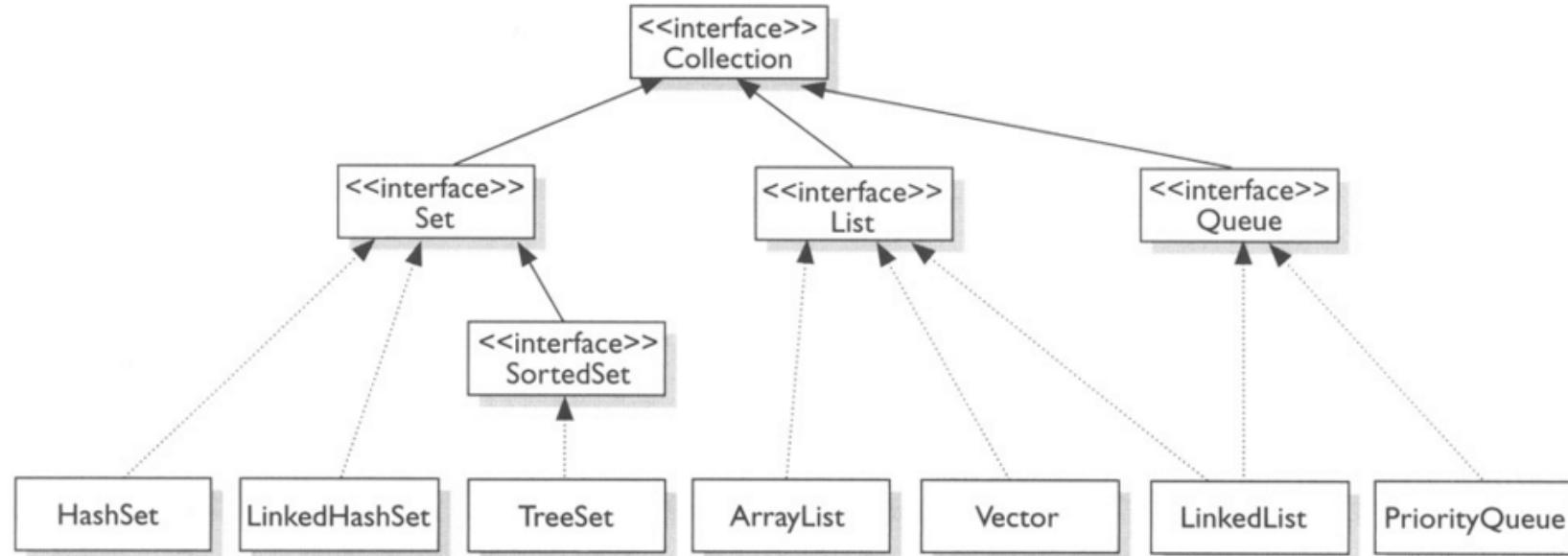
- Permitem manipular valores de tipo primitivo como objetos;
- **Boolean, Byte, Character, Double, Float, Integer, Long e Short;**
- Podem ser usados em coleções Java;
- Possuem alguns métodos que auxiliam na manipulação e conversão de dados;
- Conversão automática (*autoboxing*):

```
Integer[] integerArray = new Integer[ 5 ];  
integerArray[ 0 ] = 10;  
int value = integerArray [ 0 ];
```

Hierarquia das Coleções Java



Hierarquia das Coleções Java



Observação de engenharia de software 20.1

Collection é comumente utilizada como um tipo de parâmetro nos métodos para permitir processamento polimórfico de todos os objetos que implementam a interface Collection.

- **Collection** é a interface-raiz das interfaces *Set*, *Queue* e *List*;
- Contém operações de volume para adicionar, limpar e comparar objetos de uma coleção;
- Fornece um método que retorna um objeto **Iterator**, que permite percorrer uma coleção polimorficamente;

Resumo dos métodos da interface *Collection*

boolean	add(E e) Ensures that this collection contains the specified element (optional operation).
boolean	addAll(Collection<? extends E> c) Adds all of the elements in the specified collection to this collection (optional operation).
void	clear() Removes all of the elements from this collection (optional operation).
boolean	contains(Object o) Returns <code>true</code> if this collection contains the specified element.
boolean	containsAll(Collection<?> c) Returns <code>true</code> if this collection contains all of the elements in the specified collection.
boolean	equals(Object o) Compares the specified object with this collection for equality.
int	hashCode() Returns the hash code value for this collection.
boolean	isEmpty() Returns <code>true</code> if this collection contains no elements.
Iterator<E>	iterator() Returns an iterator over the elements in this collection.
boolean	remove(Object o) Removes a single instance of the specified element from this collection, if it is present (optional operation).
boolean	removeAll(Collection<?> c) Removes all of this collection's elements that are also contained in the specified collection (optional operation).
boolean	retainAll(Collection<?> c) Retains only the elements in this collection that are contained in the specified collection (optional operation).
int	size() Returns the number of elements in this collection.
Object[]	toArray() Returns an array containing all of the elements in this collection.
<T> T[]	toArray(T[] a) Returns an array containing all of the elements in this collection; the runtime type of the returned array is that of the specified array.

- Uma **List** é uma *Collection* que simula o comportamento de um vetor;
- Pode conter elementos duplicados;
- Fornece métodos para manipular elementos via seus índices, manipular um intervalo especificado de elementos, procurar elementos e obter um *ListIterator* para acessar os elementos.
- Implementada principalmente por três classes: *ArrayList*, *LinkedList* e *Vector*.

Métodos da Interface *List*

boolean	add(E e) Appends the specified element to the end of this list (optional operation).
void	add(int index, E element) Inserts the specified element at the specified position in this list (optional operation).
boolean	addAll(Collection<? extends E> c) Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator (optional operation).
boolean	addAll(int index, Collection<? extends E> c) Inserts all of the elements in the specified collection into this list at the specified position (optional operation).
void	clear() Removes all of the elements from this list (optional operation).
boolean	contains(Object o) Returns true if this list contains the specified element.
boolean	containsAll(Collection<?> c) Returns true if this list contains all of the elements of the specified collection.
boolean	equals(Object o) Compares the specified object with this list for equality.
E	get(int index) Returns the element at the specified position in this list.
int	hashCode() Returns the hash code value for this list.
int	indexOf(Object o) Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element.
boolean	isEmpty() Returns true if this list contains no elements.
Iterator<E>	iterator() Returns an iterator over the elements in this list in proper sequence.
int	lastIndexOf(Object o) Returns the index of the last occurrence of the specified element in this list, or -1 if this list does not contain the element.
ListIterator<E>	listIterator() Returns a list iterator over the elements in this list (in proper sequence).
ListIterator<E>	listIterator(int index) Returns a list iterator over the elements in this list (in proper sequence), starting at the specified position in the list.

Métodos da Interface *List*

E	remove(int index) Removes the element at the specified position in this list (optional operation).
boolean	remove(Object o) Removes the first occurrence of the specified element from this list, if it is present (optional operation).
boolean	removeAll(Collection<?> c) Removes from this list all of its elements that are contained in the specified collection (optional operation).
boolean	retainAll(Collection<?> c) Retains only the elements in this list that are contained in the specified collection (optional operation).
E	set(int index, E element) Replaces the element at the specified position in this list with the specified element (optional operation).
int	size() Returns the number of elements in this list.
List<E>	subList(int fromIndex, int toIndex) Returns a view of the portion of this list between the specified <code>fromIndex</code> , inclusive, and <code>toIndex</code> , exclusive.
Object[]	toArray() Returns an array containing all of the elements in this list in proper sequence (from first to last element).
<T> T[]	toArray(T[] a) Returns an array containing all of the elements in this list in proper sequence (from first to last element); the runtime type of the returned array is that of the specified array.

ArrayList, LinkedList e Vector

- *ArrayList* e *Vector* são implementações de vetores redimensionáveis de *List*;
- Inserir um elemento entre elementos existentes de *ArrayList* ou *Vector* é uma operação ineficiente;
- Uma *LinkedList* permite inserção (ou remoção) eficiente de elementos no meio de uma coleção;
- A principal diferença entre *ArrayList* e *Vector* é que *Vectors* são sincronizados por padrão.

Construtores

`ArrayList()`

Constructs an empty list with an initial capacity of ten.

`ArrayList(Collection<? extends E> c)`

Constructs a list containing the elements of the specified collection, in the order they are returned by the collection's iterator.

`ArrayList(int initialCapacity)`

Constructs an empty list with the specified initial capacity.

Métodos da Classe *ArrayList*

Métodos

boolean	add(E e) Appends the specified element to the end of this list.
void	add(int index, E element) Inserts the specified element at the specified position in this list.
boolean	addAll(Collection<? extends E> c) Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator.
boolean	addAll(int index, Collection<? extends E> c) Inserts all of the elements in the specified collection into this list, starting at the specified position.
void	clear() Removes all of the elements from this list.
Object	clone() Returns a shallow copy of this <i>ArrayList</i> instance.
boolean	contains(Object o) Returns <code>true</code> if this list contains the specified element.
void	ensureCapacity(int minCapacity) Increases the capacity of this <i>ArrayList</i> instance, if necessary, to ensure that it can hold at least the number of elements specified by the minimum capacity argument.
E	get(int index) Returns the element at the specified position in this list.
int	indexOf(Object o) Returns the index of the first occurrence of the specified element in this list, or <code>-1</code> if this list does not contain the element.
boolean	isEmpty() Returns <code>true</code> if this list contains no elements.
Iterator<E>	iterator() Returns an iterator over the elements in this list in proper sequence.
int	lastIndexOf(Object o) Returns the index of the last occurrence of the specified element in this list, or <code>-1</code> if this list does not contain the element.
ListIterator<E>	listIterator() Returns a list iterator over the elements in this list (in proper sequence).
ListIterator<E>	listIterator(int index) Returns a list iterator over the elements in this list (in proper sequence), starting at the specified position in the list.

Métodos da Classe *ArrayList*

Métodos

E	<code>remove(int index)</code> Removes the element at the specified position in this list.
<code>boolean</code>	<code>remove(Object o)</code> Removes the first occurrence of the specified element from this list, if it is present.
<code>boolean</code>	<code>removeAll(Collection<?> c)</code> Removes from this list all of its elements that are contained in the specified collection.
<code>protected void</code>	<code>removeRange(int fromIndex, int toIndex)</code> Removes from this list all of the elements whose index is between <code>fromIndex</code> , inclusive, and <code>toIndex</code> , exclusive.
<code>boolean</code>	<code>retainAll(Collection<?> c)</code> Retains only the elements in this list that are contained in the specified collection.
E	<code>set(int index, E element)</code> Replaces the element at the specified position in this list with the specified element.
<code>int</code>	<code>size()</code> Returns the number of elements in this list.
<code>List<E></code>	<code>subList(int fromIndex, int toIndex)</code> Returns a view of the portion of this list between the specified <code>fromIndex</code> , inclusive, and <code>toIndex</code> , exclusive.
<code>Object[]</code>	<code>toArray()</code> Returns an array containing all of the elements in this list in proper sequence (from first to last element).
<code><T> T[]</code>	<code>toArray(T[] a)</code> Returns an array containing all of the elements in this list in proper sequence (from first to last element); the runtime type of the returned array is that of the specified array.
<code>void</code>	<code>trimToSize()</code> Trims the capacity of this <code>ArrayList</code> instance to be the list's current size.

Construtores

`LinkedList()`

Constructs an empty list.

`LinkedList(Collection<? extends E> c)`

Constructs a list containing the elements of the specified collection, in the order they are returned by the collection's iterator.

Métodos da Classe *LinkedList*

Métodos

boolean	add(E e) Appends the specified element to the end of this list.
void	add(int index, E element) Inserts the specified element at the specified position in this list.
boolean	addAll(Collection<? extends E> c) Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator.
boolean	addAll(int index, Collection<? extends E> c) Inserts all of the elements in the specified collection into this list, starting at the specified position.
void	addFirst(E e) Inserts the specified element at the beginning of this list.
void	addLast(E e) Appends the specified element to the end of this list.
void	clear() Removes all of the elements from this list.
Object	clone() Returns a shallow copy of this <i>LinkedList</i> .
boolean	contains(Object o) Returns true if this list contains the specified element.
Iterator<E>	descendingIterator() Returns an iterator over the elements in this deque in reverse sequential order.
E	element() Retrieves, but does not remove, the head (first element) of this list.
E	get(int index) Returns the element at the specified position in this list.
E	getFirst() Returns the first element in this list.
E	getLast() Returns the last element in this list.

Métodos

int	indexOf(Object o)	Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element.
int	lastIndexOf(Object o)	Returns the index of the last occurrence of the specified element in this list, or -1 if this list does not contain the element.
ListIterator<E>	listIterator(int index)	Returns a list-iterator of the elements in this list (in proper sequence), starting at the specified position in the list.
boolean	offer(E e)	Adds the specified element as the tail (last element) of this list.
boolean	offerFirst(E e)	Inserts the specified element at the front of this list.
boolean	offerLast(E e)	Inserts the specified element at the end of this list.
E	peek()	Retrieves, but does not remove, the head (first element) of this list.
E	peekFirst()	Retrieves, but does not remove, the first element of this list, or returns null if this list is empty.
E	peekLast()	Retrieves, but does not remove, the last element of this list, or returns null if this list is empty.
E	poll()	Retrieves and removes the head (first element) of this list.
E	pollFirst()	Retrieves and removes the first element of this list, or returns null if this list is empty.
E	pollLast()	Retrieves and removes the last element of this list, or returns null if this list is empty.
E	pop()	Pops an element from the stack represented by this list.
void	push(E e)	Pushes an element onto the stack represented by this list.

Métodos da Classe *LinkedList*

Métodos

E	<code>remove()</code> Retrieves and removes the head (first element) of this list.
E	<code>remove(int index)</code> Removes the element at the specified position in this list.
boolean	<code>remove(Object o)</code> Removes the first occurrence of the specified element from this list, if it is present.
E	<code>removeFirst()</code> Removes and returns the first element from this list.
boolean	<code>removeFirstOccurrence(Object o)</code> Removes the first occurrence of the specified element in this list (when traversing the list from head to tail).
E	<code>removeLast()</code> Removes and returns the last element from this list.
boolean	<code>removeLastOccurrence(Object o)</code> Removes the last occurrence of the specified element in this list (when traversing the list from head to tail).
E	<code>set(int index, E element)</code> Replaces the element at the specified position in this list with the specified element.
int	<code>size()</code> Returns the number of elements in this list.
Object[]	<code>toArray()</code> Returns an array containing all of the elements in this list in proper sequence (from first to last element).
<T> T[]	<code>toArray(T[] a)</code> Returns an array containing all of the elements in this list in proper sequence (from first to last element); the runtime type of the returned array is that of the specified array.

Exemplo 1

```
1 // Figura 20.2: CollectionTest.java
2 // A interface Collection demonstrada via um objeto ArrayList.
3 import java.util.List;
4 import java.util.ArrayList;
5 import java.util.Collection;
6 import java.util.Iterator;
7
8 public class CollectionTest
9 {
10     public static void main( String[] args )
11     {
12         // adiciona elementos no array colors a listar
13         String[] colors = { "MAGENTA", "RED", "WHITE", "BLUE", "CYAN" };
14         List< String > list = new ArrayList< String >(); ←
15
16         for ( String color : colors )
17             list.add( color ); // adiciona a cor ao fim da lista
18
19         // adiciona elementos no array removeColors em removeList
20         String[] removeColors = { "RED", "WHITE", "BLUE" };
21         List< String > removeList = new ArrayList< String >();
22 }
```

É uma boa prática referenciar uma coleção via uma variável de tipo de interface - é mais fácil alterar a coleção posteriormente

Figura 20.2 | A interface Collection demonstrada via um objeto ArrayList. (Parte I de 3.)

Exemplo 1

```
23     for ( String color : removeColors )
24         removeList.add( color );
25
26     // gera a saída do conteúdo da lista
27     System.out.println( "ArrayList: " );
28
29     for ( int count = 0; count < list.size(); count++ )
30         System.out.printf( "%s ", list.get( count ) );
31
32     // remove da lista as cores contidas em removeList
33     removeColors( list, removeList );
34
35     // gera a saída do conteúdo da lista
36     System.out.println( "\n\nArrayList after calling removeColors: " );
37
38     for ( String color : list )
39         System.out.printf( "%s ", color );
40 } // fim de main
41
```

Exemplo 1

```
42  // remove cores especificadas em collection2 a partir de collection1
43  private static void removeColors(Collection< String > collection1,
44      Collection< String > collection2 )
45  {
46      // obtém o iterador
47      Iterator< String > iterator = collection1.iterator();
48
49      // loop enquanto a coleção tiver itens
50      while (iterator.hasNext() )
51      {
52          if ( collection2.contains( iterator.next() ) )
53              iterator.remove(); // remove Color atual
54      } // fim do while
55  } // fim do método removeColors
56 } // fim da classe CollectionTest
```

O método funciona com
qualquer Collection

ArrayList:
MAGENTA RED WHITE BLUE CYAN

ArrayList after calling removeColors:
MAGENTA CYAN



Erro comum de programação 20.1

Se uma coleção for modificada por um dos seus métodos depois que um iterador é criado para essa coleção, o iterador torna-se imediatamente inválido — as operações realizadas com o iterador depois desse ponto lançam `ConcurrentModificationExceptions`. Por essa razão, diz-se que os iteradores “respondem rápido”.

Exemplo 2

```
1 // Figura 20.3: ListTest.java
2 // Lists, LinkedLists e ListIterators.
3 import java.util.List;
4 import java.util.LinkedList;
5 import java.util.ListIterator;
6
7 public class ListTest
8 {
9     public static void main( String[] args )
10    {
11        // adiciona elementos colors a list1
12        String[] colors =
13            { "black", "yellow", "green", "blue", "violet", "silver" };
14        List< String > list1 = new LinkedList< String >();
15
16        for ( String color : colors )
17            list1.add( color );
18
19        // adiciona elementos colors2 à list2
20        String[] colors2 =
21            { "gold", "white", "brown", "blue", "gray", "silver" };
22        List< String > list2 = new LinkedList< String >();
23
```

Exemplo 2

```
24     for ( String color : colors2 )
25         list2.add( color );
26
27     list1.addAll( list2 ); // concatena as listas
28     list2 = null; // libera recursos
29     printList( list1 ); // imprime elementos list1
30
31     convertToUppercaseStrings( list1 ); // converte para string maiúscula
32     printList( list1 ); // imprime elementos list1
33
34     System.out.print( "\nDeleting elements 4 to 6..." );
35     removeItems( list1, 4, 7 ); // remove itens 4-6 da lista
36     printList( list1 ); // imprime elementos list1
37     printReversedList( list1 ); // imprime lista na ordem inversa
38 } // fim de main
39
40 // gera saída do conteúdo de List
41 private static void printList(List< String > list )
42 {
43     System.out.println( "\nlist: " );
44
45     for ( String color : list )
46         System.out.printf( "%s ", color );
47
```

Exemplo 2

```
48     System.out.println();
49 } // fim do método printList
50
51 // localiza objetos String e converte em letras maiúsculas
52 private static void convertToUppercaseStrings(List< String > list )
53 {
54     ListIterator< String > iterator = list.listIterator();
55
56     while (iterator.hasNext() )
57     {
58         String color = iterator.next(); // obtém o item
59         iterator.set( color.toUpperCase() ); // converte em letras maiúsculas
60     } // fim do while
61 } // fim do método convertToUppercaseStrings
62
63 // obtém sublista e utiliza método clear para excluir itens da sublista
64 private static void removeItems(List< String > list,
65     int start, int end )
66 {
67     list.subList( start, end ).clear(); // remove os itens ← subList retorna uma
68 } // fim do método removeItems
69
```

Exemplo 2

```
70  // imprime lista invertida
71  private static void printReversedList(List< String > list )
72  {
73      ListIterator< String > iterator = list.listIterator( list.size() );
74
75      System.out.println( "\nReversed List:" );
76
77      // imprime lista na ordem inversa
78      while ( iterator.hasPrevious() )
79          System.out.printf( "%s ", iterator.previous() );
80  } // fim do método printReversedList
81 } // fim da classe ListTest
```

```
list:
black yellow green blue violet silver gold white brown blue gray silver

list:
BLACK YELLOW GREEN BLUE VIOLET SILVER GOLD WHITE BROWN BLUE GRAY SILVER

Deleting elements 4 to 6...
list:
BLACK YELLOW GREEN BLUE WHITE BROWN BLUE GRAY SILVER

Reversed List:
SILVER GRAY BLUE BROWN WHITE BLUE GREEN YELLOW BLACK
```

Exemplo 3

```
1 // Figura 20.15: PriorityQueueTest.java
2 // Programa de teste PriorityQueue.
3 import java.util.PriorityQueue;
4
5 public class PriorityQueueTest
6 {
7     public static void main( String[] args )
8     {
9         // fila de capacidade 11
10        PriorityQueue< Double > queue = new PriorityQueue< Double >(); ← A ordem natural determina
11
12        // insere elementos na fila
13        queue.offer( 3.2 );
14        queue.offer( 9.8 );
15        queue.offer( 5.4 );
16
17        System.out.print( "Polling from queue: " );
18    }
}
```

Figura 20.15 | Programa de teste PriorityQueue. (Parte I de 2.)

Exemplo 3

```
19     // exibe elementos na fila
20     while (queue.size() > 0 )
21     {
22         System.out.printf( "%.1f ", queue.peek() ); // visualiza elemento superior
23         queue.poll(); // remove elemento superior
24     } // fim do while
25 } // fim de main
26 } // fim da classe PriorityQueueTest
```

```
Polling from queue: 3.2 5.4 9.8
```

- A classe **Collections** fornece vários algoritmos de alto desempenho para manipular elementos de coleção;
- Todos os algoritmos são implementados como métodos *static*.

Métodos da classe *Collections*

Método	Descrição
<code>sort</code>	Classifica os elementos de uma <code>List</code> .
<code>binarySearch</code>	Localiza um objeto em uma <code>List</code> .
<code>reverse</code>	Inverte os elementos de uma <code>List</code> .
<code>shuffle</code>	Ordena aleatoriamente os elementos de uma <code>List</code> .
<code>fill</code>	Configura todo elemento <code>List</code> para referir-se a um objeto especificado.
<code>copy</code>	Copia referências de uma <code>List</code> em outra.
<code>min</code>	Retorna o menor elemento em uma <code>Collection</code> .
<code>max</code>	Retorna o maior elemento em uma <code>Collection</code> .
<code>addAll</code>	Acrescenta todos os elementos em um array a uma <code>Collection</code> .
<code>frequency</code>	Calcula quantos elementos da coleção são iguais ao elemento especificado.
<code>disjoint</code>	Determina se duas coleções não têm nenhum elemento em comum.

Exemplo *Collections* 1

```
1 // Figura 20.6: Sort1.java
2 // Método Collections sort.
3 import java.util.List;
4 import java.util.Arrays;
5 import java.util.Collections;
6
7 public class Sort1
8 {
9     public static void main( String[] args )
10    {
11        String[] suits = { "Hearts", "Diamonds", "Clubs", "Spades" };
12
13        // Cria e exibe uma lista que contém os elementos do array de ternos
14        List< String > list = Arrays.asList( suits ); // cria List
15        System.out.printf( "Unsorted array elements: %s\n", list );
16
17        Collections.sort( list ); // classifica ArrayList ←
18                                Elementos list devem ser Comparable
19
20        // gera a saída da lista
21        System.out.printf( "Sorted array elements: %s\n", list );
22    } // fim de main
23 } // fim da classe Sort1
```

Exemplo *Collections* 2

```
1  // Figura 20.8: TimeComparator.java
2  // Classe Comparator personalizada que compara dois objetos Time2.
3  import java.util.Comparator;
4
5  public class TimeComparator implements Comparator< Time2 > ← Comparator personalizado
6  {                                         para objetos Time2
7      public int compare(Time2 time1, Time2 time2 )
8      {
9          int hourCompare = time1.getHour() - time2.getHour(); // compara hora
10
11         // testa a primeira hora
12         if ( hourCompare != 0 )
13             return hourCompare;
14
15         int minuteCompare =
16             time1.getMinute() - time2.getMinute(); // compara minuto
17
18         // então testa o minuto
19         if ( minuteCompare != 0 )
20             return minuteCompare;
21
22         int secondCompare =
23             time1.getSecond() - time2.getSecond(); // compara segundo
```

Exemplo *Collections* 2

```
24
25     return secondCompare; // retorna o resultado da comparação de segundos
26 } // fim do método compare
27 } // fim da classe TimeComparator
```

Figura 20.8 | Classe Comparator personalizada que compara dois objetos Time2. (Parte 2 de 2.)

Exemplo *Collections* 2

```
1  // Figura 20.9: Sort3.java
2  // Método Collections.sort com um objeto Comparator personalizado.
3  import java.util.List;
4  import java.util.ArrayList;
5  import java.util.Collections;
6
7  public class Sort3
8  {
9      public static void main( String[] args )
10     {
11         List< Time2 > list = new ArrayList< Time2 >(); // cria List
12
13         list.add( new Time2( 6, 24, 34 ) );
14         list.add( new Time2( 18, 14, 58 ) );
15         list.add( new Time2( 6, 05, 34 ) );
16         list.add( new Time2( 12, 14, 58 ) );
17         list.add( new Time2( 6, 24, 22 ) );
18
19         // gera a saída de elementos List
20         System.out.printf( "Unsorted array elements:\n%s\n", list );
21     }
}
```

Figura 20.9 | Método Collections.sort com um objeto Comparator personalizado. (Parte 1 de 2.)

Exemplo *Collections* 2

```
22  // classifica em ordem utilizando um comparador
23  Collections.sort( list, new TimeComparator() );
24
25  // gera a saída de elementos List
26  System.out.printf( "Sorted list elements:\n%s\n", list );
27 } // fim de main
28 } // fim da classe Sort3
```

Objetos Time2 não podiam ser classificados antes de se criar o TimeComparator; a técnica pode ser utilizada para tornar objetos de praticamente qualquer classe classificáveis

```
Unsorted array elements:
[6:24:34 AM, 6:14:58 PM, 6:05:34 AM, 12:14:58 PM, 6:24:22 AM]
Sorted list elements:
[6:05:34 AM, 6:24:22 AM, 6:24:34 AM, 12:14:58 PM, 6:14:58 PM]
```

Interface Set/Sorted

- A **Set** é uma *Collection* não ordenada de elementos **únicos**.
- *SortedSet* é uma extensão de *Set* que mantém uma ordem relativa entre os elementos.
- Principais implementações: **HashSet** e **TreeSet**.

Modifier and Type	Method and Description
boolean	<code>add(E e)</code> Adds the specified element to this set if it is not already present (optional operation).
boolean	<code>addAll(Collection<? extends E> c)</code> Adds all of the elements in the specified collection to this set if they're not already present (optional operation).
void	<code>clear()</code> Removes all of the elements from this set (optional operation).
boolean	<code>contains(Object o)</code> Returns true if this set contains the specified element.
boolean	<code>containsAll(Collection<?> c)</code> Returns true if this set contains all of the elements of the specified collection.
boolean	<code>equals(Object o)</code> Compares the specified object with this set for equality.
int	<code>hashCode()</code> Returns the hash code value for this set.
boolean	<code>isEmpty()</code> Returns true if this set contains no elements.
<code>Iterator<E></code>	<code>iterator()</code> Returns an iterator over the elements in this set.
boolean	<code>remove(Object o)</code> Removes the specified element from this set if it is present (optional operation).
boolean	<code>removeAll(Collection<?> c)</code> Removes from this set all of its elements that are contained in the specified collection (optional operation).
boolean	<code>retainAll(Collection<?> c)</code> Retains only the elements in this set that are contained in the specified collection (optional operation).
int	<code>size()</code> Returns the number of elements in this set (its cardinality).
<code>Object[]</code>	<code>toArray()</code> Returns an array containing all of the elements in this set.
<code><T> T[]</code>	<code>toArray(T[] a)</code> Returns an array containing all of the elements in this set; the runtime type of the returned array is that of the specified array.

Exemplo Interface *Set/SortedSet*

```
1  // Figura 20.17: SortedSetTest.java
2  // Utilizando SortedSets e TreeSets.
3  import java.util.Arrays;
4  import java.util.SortedSet;
5  import java.util.TreeSet;
6
7  public class SortedSetTest
8  {
9      public static void main( String[] args )
10     {
11         // cria TreeSet a partir do array colors
12         String[] colors = { "yellow", "green", "black", "tan", "grey",
13             "white", "orange", "red", "green" };
14         SortedSet< String > tree =
15             new TreeSet< String >( Arrays.asList( colors ) );
16
17         System.out.print( "sorted set: " );
18         printSet( tree ); // conteúdo de saída de árvore
19
20         // obtém headSet com base em "orange"
21         System.out.print( "headSet (\"orange\"): " );
22         printSet( tree.headSet( "orange" ) );
--
```

Exemplo Interface *Set/SortedSet*

```
24      // obtém tailSet baseado em "orange"
25      System.out.print( "tailSet (\"orange\"): " );
26      printSet(tree.tailSet( "orange" ) );
27
28      // obtém o primeiro e o último elementos
29      System.out.printf( "first: %s\n", tree.first() );
30      System.out.printf( "last : %s\n", tree.last() );
31  } // fim de main
32
33  // imprime a SortedSet utilizando a instrução for aprimorada
34  private static void printSet( SortedSet< String > set )
35  {
36      for ( String s : set )
37          System.out.printf( "%s ", s );
38
39      System.out.println();
40  } // fim do método printSet
41 } // fim da classe SortedSetTest
```

```
sorted set: black green grey orange red tan white yellow
headSet ("orange"): black green grey
tailSet ("orange"): orange red tan white yellow
first: black
last : yellow
```

- **Maps** associam chaves a valores;
- Chaves de um *Map* são únicas, mas os valores associados, não;
- Interface **SortedMap** estende a *Map*, porém mantém suas chaves em ordem de classificação.
- Principais implementações: **Hashtable**, **HashMap** e **TreeMap**.

Exemplo Interface Map/SortedMap

```
1 // Figura 20.18: WordTypeCount.java
2 // O programa conta o número de ocorrências de cada palavra em uma String.
3 import java.util.Map;
4 import java.util.HashMap;
5 import java.util.Set;
6 import java.util.TreeSet;
7 import java.util.Scanner;
8
9 public class WordTypeCount
10 {
11     public static void main( String[] args )
12     {
13         // cria HashMap para armazenar chaves de String e valores Integer
14         Map< String, Integer > myMap = new HashMap< String, Integer >();
15
16         createMap( myMap ); // cria mapa com base na entrada do usuário
17         displayMap( myMap ); // exibe o conteúdo do mapa
18     } // fim de main
19
20     // cria mapa de entrada do usuário
21     private static void createMap( Map< String, Integer > map )
22     {
```

Exemplo Interface Map/SortedMap

```
23     Scanner scanner = new Scanner( System.in ); // cria o scanner
24     System.out.println( "Enter a string:" ); // solicita a entrada do usuário
25     String input = scanner.nextLine();
26
27     // tokeniza a entrada
28     String[] tokens = input.split( " " );
29
30     // processamento de texto de entrada
31     for ( String token : tokens )
32     {
33         String word = token.toLowerCase(); // obtém palavra em letras minúsculas
34
35         // se o mapa contiver a palavra
36         if (map.containsKey( word ) ) // is word in map
37         {
38             int count = map.get( word ); // obtém a contagem atual
39             map.put( word, count + 1 ); // incrementa a contagem
40         } // fim do if
41         else
42             map.put( word, 1 ); // adiciona nova palavra com uma contagem de 1 para mapa
43     } // fim do for
44 } // fim do método createMap
```

Exemplo Interface Map/SortedMap

```
45
46     // exibe o conteúdo do mapa
47     private static void displayMap( Map< String, Integer > map )
48     {
49         Set< String > keys = map.keySet(); //obtém chaves
50
51         // classifica as chaves
52         TreeSet< String > sortedKeys = new TreeSet< String >( keys );
53
54         System.out.println( "\nMap contains:\nKey\t\tValue" );
55
56         // gera a saída de cada chave no mapa
57         for ( String key : sortedKeys )
58             System.out.printf( "%-10s%10s\n", key, map.get( key ) );
59
60         System.out.printf(
61             "\nsize: %d\nisEmpty: %b\n", map.size(), map.isEmpty() );
62     } // fim do método displayMap
63 } // fim da classe WordTypeCount
```

Exemplo Interface *Map/SortedMap*

```
Enter a string:
```

```
this is a sample sentence with several words this is another sample
sentence with several different words
```

```
Map contains:
```

Key	Value
a	1
another	1
different	1
is	2
sample	2
sentence	2
several	2
this	2
with	2
words	2

```
size: 10
```

```
isEmpty: false
```