**Introdução**   
O array é um grupo de posições contínuas na memória que possuem o mesmo nome e o mesmo tipo. Para referir-se a uma localização ou elemento particular no array, especificamos o nome do array e o número da posição(ou índice ou subscrito) do elemento particular no array.   
**Roteiro**

1. [**Array**](http://javanoroeste.com.br/javanoroeste/novo/artigos/arrays_colecoes.html#Array)**;**
2. [**Vetores**](http://javanoroeste.com.br/javanoroeste/novo/artigos/arrays_colecoes.html#Vetores)**;**
3. [**Coleções**](http://javanoroeste.com.br/javanoroeste/novo/artigos/arrays_colecoes.html#Cole%C3%A7%C3%B5es)**;**
4. [**Exemplo ArrayList**](http://javanoroeste.com.br/javanoroeste/novo/artigos/arrays_colecoes.html#Exemplo_ArrayList)**;**
5. [**Exemplo LinkedList**](http://javanoroeste.com.br/javanoroeste/novo/artigos/arrays_colecoes.html#Exemplo_LinkedList)**;**
6. [**Exemplo HashSet**](http://javanoroeste.com.br/javanoroeste/novo/artigos/arrays_colecoes.html#Exemplo_HashSet)**;**
7. [**Exemplo TreeSet**](http://javanoroeste.com.br/javanoroeste/novo/artigos/arrays_colecoes.html#Exemplo_TreeSet)**;**
8. [**Exemplo HashMap**](http://javanoroeste.com.br/javanoroeste/novo/artigos/arrays_colecoes.html#Exemplo_HashMap)**;**

**1. Array**

Um array em Java é tratado como um objeto, afinal ele advém da classe Array, e como todo objeto, é preciso:   
1)Declará-lo. Exemplo: int[] meuArray;  
    2)Construí-lo. Exemplo: meuArray = new int[2];  
    3)Inicializar. Exemplo: meuArray[0] = 2; meuArray[1] = 4.   
Neste caso foi criado um array com uma dimensão e duas posições do tipo inteiro, note os colchetes (“[]”). Cada conjunto indica uma dimensão. Como ilustração, podemos imaginar como exemplo, uma gaveta contendo várias pastas onde a posição de cada  pasta seria uma posição do array. A posição zero seria o primeiro elemento, posição 1 o segundo elemento e assim por diante até o último elemento(n -1).    
    Vamos supor que desejamos construir um array de números inteiros. Então devemos realizar uma declaração do mesmo, como segue:   
        int[] meuArray;   
O próximo passo é construir. Para isto, vamos definir que o array terá dez posições:  
        meuArray = new int[10];  
    Vamos iniciar as cinco primeiras posições do array com valores:       
         meuArray[0] = 22;  
         meuArray[1] = 2;  
         meuArray[2] = 242;  
         meuArray[3] = 4552;  
         meuArray[4] = 36.   
Agora, vamos construir um laço de repetição for para percorrer os elementos e produzir uma saída no terminal:   
        for (int i = 0; i < meuArray.length; i++){  
            System.out.println("posição " + i + " = " + meuArray[i]);  
        ]   
    O código completo fica desta maneira:

**public** class MeuArray {  
   
  **public** **static** void **main**( String args[]){  
 int[] meuArray;  
 meuArray = **new** int[10];  
 meuArray[0] = 22;  
 meuArray[1] = 2;  
 meuArray[2] = 242;  
 meuArray[3] = 4552;  
 meuArray[4] = 36;   
   
 **for** (int i = 0; i < meuArray.length; i++){  
 System.out.**println**("posição " + i + " = " + meuArray[i]);  
 }  
   
 }  
   
 }

    O programa ao ser executado vai produzir uma saída no terminal:   
    posição 0 = 22  
    posição 1 = 2  
    posição 2 = 242  
    posição 3 = 4552  
    posição 4 = 36  
    posição 5 = 0  
    posição 6 = 0  
    posição 7 = 0  
    posição 8 = 0  
    posição 9 = 0  
  
    Note que um dos parâmetros dentro da estrutura de repetição for é o método que obtém o tamanho do array, “length”.  
    Este método aplicado ao objeto meuArray(meuArray.length), retorna um número inteiro correspondente ao tamanho do array.  
    Existe uma classe que auxilia a utilização de array, que está localizada no pacote  java.util, tem o nome de Arrays, e possui os seguintes métodos:

* **binarySearch** – permite uma pesquisa nos elementos de um determinado array ordenado, retornando um atributo inteiro com a posição deste elemento;
* **equal** – permite a comparação entre dois arrays, retornando um booleao verdadeiro(true), caso os array sejam iguais;
* **fill** – realiza o preenchimento de todos os elementos de um determinado array;
* **sort** – faz uma ordenação nos elementos de um determinado array;
* **toString** – mostra os elementos de um determinado array.

    Uma outra forma de inicializar um array, seria declarar o array já inicializando-o  com valores:  
            int[] meuArray = {22,2,242,4552,36};  
      
Suponha agora que desejamos descobrir no objeto meuArray a posição que contem o elemento 4552. O método binarySearch  da classe Arrays é o mais indicado para isto, porém, no array meuArray não está ordenado, sendo uma condição para utilização do método binarySearch . O primeiro passo então seria ordenar o meuArray utilizando o método sort:  
    java.util.Arrays.sort(meuArray);  
      
    Agora podemos utilizar o método  binarySearch desta maneira:  
  
    java.util.Arrays.binarySearch(meuArray,4552);  
  
    Alterando o código exemplo temos:

**public** class MeuArray {  
   
  **public** **static** void **main**( String args[]){  
 int[] meuArray;  
 meuArray = **new** int[10];  
 meuArray[0] = 22;  
 meuArray[1] = 2;  
 meuArray[2] = 242;  
 meuArray[3] = 4552;  
 meuArray[4] = 36;   
   
 java.util.Arrays.**sort**(meuArray);  
   
  **for** (int i = 0; i < meuArray.length; i++){  
 System.out.**println**("posição " + i + " = " + meuArray[i]);  
 }  
   
 System.out.**println**("posição do elemento 4552 :" + java.util.Arrays.**binarySearch**(meuArray,4552));  
   
   
 }  
   
 }

    Vamos agora construir um objeto array idêntico ao meuArray da seguinte forma:  
  
        int [] espelho =  meuArray;  
      
    O método equals da classe Arrays, permite comparar dois arrays retornando “true” ou “false”, como mostra o código a seguir::  
  
    if (java.util.Arrays.equals(meuArray,espelho)){  
            System.out.println("são identicos...");  
    }else{  
            System.out.println("são diferentes...");  
    }  
    O método fill da classe Arrays, preenche todos os elementos de um array com um dado valor, da seguinte forma:   
        java.util.Arrays.fill(espelho,5522);  
    Percorrendo todos os elementos deste array verificamos que os valores são os mesmos para todas as posições do array espelho:   
        posição 0 = 5522  
        posição 1 = 5522  
        posição 2 = 5522  
        posição 3 = 5522  
        posição 4 = 5522  
        posição 5 = 5522  
        posição 6 = 5522  
        posição 7 = 5522  
        posição 8 = 5522  
        posição 9 = 5522  
  
    Para finalizar, o método toString() da classe Arrays permite exibir todos elementos de um objeto array, como mostra o código a seguir:     System.out.println("Conteúdo do objeto array : " + java.util.Arrays.toString(meuArray));

**public** class MeuArray {  
   
  **public** **static** void **main**( String args[]){  
 int[] meuArray;  
 meuArray = **new** int[10];  
 meuArray[0] = 22;  
 meuArray[1] = 2;  
 meuArray[2] = 242;  
 meuArray[3] = 4552;  
 meuArray[4] = 36;   
   
 java.util.Arrays.**sort**(meuArray);  
   
  **for** (int i = 0; i < meuArray.length; i++){  
 System.out.**println**("posiÃ§Ã£o " + i + " = " + meuArray[i]);  
 }  
   
 System.out.**println**("Conteúdo do objeto array : " + java.util.Arrays.**toString**(meuArray));  
   
 System.out.**println**("posição do elemento 4552 :" + java.util.Arrays.**binarySearch**(meuArray,4552));  
   
 int [] espelho = meuArray;  
   
  **if** (java.util.Arrays.**equals**(meuArray,espelho)){  
 System.out.**println**("são identicos...");  
  **else**{  
 System.out.**println**("são diferentes...");  
 }   
   
 java.util.Arrays.**fill**(espelho,5522);  
  
  **for** (int j = 0; j < meuArray.length; j++){  
 System.out.**println**("posição " + j + " = " + meuArray[j]);  
 }   
   
 System.out.**println**("Conteúdo do objeto array : " + java.util.Arrays.**toString**(meuArray));  
   
 }  
   
 }

    A saída produzida no terminal é a que segue:  
  
    Conteúdo do objeto array : [0, 0, 0, 0, 0, 2, 22, 36, 242, 4552]  
  
    Para entender com é u arrray com mais de uma dimensão, pense que uma dimensão de um array seria como uma gaveta com pastas, e duas dimensões seria um armário completo. Assim, para localizarmos uma pasta é necessário localizar primeiro qual gaveta, para depois localizar a pasta dentro desta gaveta.     Observe o código a seguir:

**public** class Armario {  
   
  */\*\* Inicia programa Armario \*/*  
 **public** **static** void main (String args[]){  
   
  
 String [][] dado = **new** String[3][3];  
   
  **for** (int gaveta = 0 ; gaveta < dado.length; gaveta++){  
  **for**(int pasta = 0; pasta < dado[gaveta].length; pasta++){  
 dado[gaveta][pasta] = "Gaveta " + gaveta + " pasta " + pasta;   
 }  
 }  
  
  **for** (int gaveta = 0 ; gaveta < dado.length; gaveta++){  
  **for**(int pasta = 0; pasta < dado[gaveta].length; pasta++){  
 System.out.**println**(dado[gaveta][pasta]);   
 }  
 }   
 }  
   
 }

    Um exemplo de programa que ordena um array.

*// Pacote Java*  
 import javax.swing.JOptionPane;  
  
 */\*\**  
 *\**  
 *\* @author root*  
 *\*/*  
  
  
 **public** class OrdenaArray{   
   
 **public** **static** void **ordena**(int[] meuarray){  
   
 *// tamanho do array*  
 int x = meuarray.length;  
 int incr = x / 2;  
   
 **while** (incr >= 1){   
  **for** (int i = incr; i < x; i++){   
 int temp = meuarray[i];  
 int j = i;  
  **while** (j >= incr && temp < meuarray[j - incr]){  
 meuarray[j] = meuarray[j - incr];  
 j -= incr;  
 }  
 meuarray[j] = temp;  
 }  
 incr /= 2;  
 }  
 }  
  
 **public** **static** void **printarray**(int[] a){   
 String var = "";  
  **for** (int i = 0; i < a.length; i++){  
 var = var.**concat**(a[i] + "\n\r");  
  
 }  
  
 JOptionPane.**showMessageDialog**( **null**, var, "Array ordenado", JOptionPane.INFORMATION\_MESSAGE );  
 }  
  
 **public** **static** void **main**(String[] args){  
 *// constroi um array com dez inteiro*  
 int[] meuarray = **new** int[10];  
   
 int i;  
   
  *// preenche o array com valores randomicos*  
  **for** (i = 0; i < meuarray.length; i++)  
 meuarray[i] = (int)(Math.**random**() \* 100);  
 **printarray**(meuarray);  
 **ordena**(meuarray);  
 **printarray**(meuarray);  
 }  
 }

|  |  |
| --- | --- |
| Antes | Depois |

**2. Vetores**

    Em muitas linguagens de programação – em particular C – é necessário fixar o tamanho de todos os arrays em tempo de compilação. Os programadores odeiam isto porque os força a fazerem suposições e assumirem compromissos desconfortáveis.   
    Em Java, a situação é muito melhor. Pode-se definir o tamanho de um array em tempo de execução.   
    int n;  
    ...  
    Item[] itensPedidos = new Item[n+1];  
  
    É claro, esse código não resolve totalmente o problema de modificar dinamicamente arrays em tempo de execução. Uma vez definido o tamanho do array, não se pode alterá-lo facilmente. Em vez disso, a maneira mais fácil de lidar com essa situação comum em Java é usar outro objeto Java que funcione como um array e que cresça ou diminua automaticamente. Esse objeto é chamado de vetor(vector). Assim, em Java, os vetores são objetos parecidos com arrays que podem crescer ou diminuir automaticamente sem que seja necessário escrever nenhum código para isso.   
    Há uma diferença importante entre vetor e um array. Os arrays são um recurso da linguagem Java e há um tipo de array T[] para cada elemento do tipo T. Contudo, a classe Vector é uma classe de biblioteca, definida no pacote java.util, e há “um único tamanho de vetor que atende a todos” contendo elementos do tipo Object. Em particular, será necessário uma conversão de tipo sempre que se quiser obter um itm de um vetor.   
    Cria-se um novo vetor especificando sua capacidade inicial no construtor Vector.   
    Vector itensOrdenados = new Vector(3);  
    // começa com espaço para um item pedido,  
    // mais dois itens para encargos de remessa a taxas  
      
    Há um importante distinção entre a capacidade de um vetor e o tomanho de um array. Se você reservar um array com três itens, então o array terá três entradas prontas para serem usadas. Um vetor com uma capacidade de três elementos tem potencial de conter três elementos (e, na verdade, mais de três), mas no início, mesmo depois de sua construção inicial, um vetor não contém elemento nenhum.  
    Use o método add para adicionar novos elementos a um vetor. Por exemplo, suponhamos que você tenha uma classe chamada Item e use código a seguir para criar três objetos Item.   
    Item proxItem = new Item(...);  
    Item ICMS = new Item(...);  
    Item remessa = new Item(...);  
  
    Depois, use o código a seguir para adicionar esses itens a um vetor chamado itensOrdenados(que começa com uma capacidade de três objetos, conforme indicado no código).  
    itensOrdenados.add(proxItem);  
    itensOrdenados.add(ICMS);      
    itensOrdenados.add(remessa);  
  
    Suponhamos que você criou o vetor de modo que tenha uma capacidade original de três itens. Se você inserir outro item, então  terá execido a capacidade do vetor do nosso exemplo. É aqui que os vetores fazem sua mágica: o vetor se redimensiona e realoca memória automáticamente. O vetor encontra uma área maior de memória e copia automaticamente todos os objetos que ele contém atualmente para esse local.     Quanto espaço é alocado? A priori, o espaço alocado dobra cada vez que o vetor é redimendionado. Devido ao problema de crescimento exponencial em potencial, talvez não se queira contar com esse processedimento para realocar a grande quantidade de memória usada em vetores grandes. Em vez disso, pode-se especificar um incremento de capacidade como segundo argumento do construtor ao ser criado o vetor. Por exemplo:   
    Vetor itensOrdenados = new Vector (3,10);      
  
    Agora o vetor vai crescer com incremento de 10 em cada realocação.  
    Por outro lado, se um programa Java tiver de realocar o espaço de um vetor frequentemente, isso vai reduzir o desempenho do programa, de modo que é importante fazer uma estimativa razoável da capacidade inicial e do incremento da capacidade – em que 10 pode ser um incremento muito pequeno.  
  
    Vector()  
    Constrói um vetor vázio (com capacidade inicial 10 e incremento de capacidade igual ao dobro sempre qua a capacidade atual é excedida).  
  
    Vector(int capacidadeInicial)  
    Constrói um vetor vazio com a capacidade especificada.  
  
    Vector(int capacidadeInicial, int incrementoCapacidade)  
    Constrói um vetor vázio com capacidade especificada e o incremento especificado)  
  
    boolean add(Object)  
    Adiciona um elemento ao final do vetor de modo que se torne o último elemento do vetor. Sempre retorna true (verdadeiro).  
  
    void set size(int i)  
    Especifica o tamanho do vetor em n elementos exatamente. Se n for maior que o tamanho atual, serão adicionados elementos nulos ao final do vetor. Se n for menos que o tamanho atual, todos os elementos de índice n em diante serão removidos.  
  
    void set (int indice, Object obj)  
    Coloca um valor no vetor em um índice especificado, sobrescrevendo o conteúdo anterior.  
  
    Object get (int indice)  
    Obtém o valor armazenado em um índice especificado.  
  
    add (int indice, Object obj)  
    Desloca para cima os elementos para poder inserir um outro elemento.  
  
**3.Coleções**   
    Estruturas de dados são mecanismos para manipular coleções de elementos em um programa. O pacote  java.util oferece algumas classes pré-definidas na API padrão de Java que implementam funcionalidades associadas a estruturas de dados.    
    As classes de coleções da API Java compõem o chamado framework de coleções, que foi completamente redefinido a partir da versão 1.2 de Java. As classes até então existentes, as classes de coleção históricas, foram reprojetadas para se adequar ao novo framework e são mantidas por motivos de compatibilidade, embora a recomendação seja utilizar as novas classes de coleções.   
    O framework de coleções foi introduzido a partir da plataforma Java 2, oferecendo um novo modelo de classes e interfaces para a manipulação de estruturas de dados. O modelo histórico não foi descontinuado, havendo ocorrido algumas adaptações quando necessário para adequar suas classes ao novo framework.   
    O framework tem por raiz duas interfaces básicas: Collection, usada para a manipulação de coleções onde os elementos são objetos individuais, e Map, usada para a manipulação de estruturas de pares de objetos, sendo um deles usado como a chave de acesso.   
    Um objeto  Collection  representa um agrupamento genérico de objetos simples. Objetos que implementam esta interface devem manter uma estrutura interna que permita acrescentar objetos à coleção, removê-los e consultar se um objeto é ou não parte da coleção.  
    Entre os métodos básicos especificados nesta interface estão:

* add() para adicionar o elemento especificado à coleção.
* remove() para remover o elemento especificado da coleção.
* contains() para verificar se a coleção contém o elemento especificado.
* size() para obter a quantidade de elementos na coleção.

    Para percorrer os elementos de uma coleção, um objeto Iterator é utilizado. O método iterator() é usado para obter esse objeto que permite varrer todos os elementos da coleção.   
    A interface  Iterator associa, a cada coleção, um mecanismo para percorrer seu conteúdo independentemente da sua estrutura interna. Um objeto desta classe oferece os métodos:

* hasNext() para indicar se há mais elementos na coleção;
* next() para obter o próximo elemento da coleção; e
* remove() para retirar um elemento da coleção.

    A interface  ListIterator ï¿½ uma extensão de Iterator que permite a varredura da coleção nas duas direções, especificando adicionalmente os métodos hasPrevious() e previous().   
    Duas outras interfaces são derivadas de Collection, Set e List. São especificações que de alguma forma restrigem ou complementam as funcionalidades básicas especificadas em Collection.   
    A interface Set é uma extensão de Collection que não acrescenta nenhum método à especificação básica. No entanto, o comportamento dessa coleção é diferenciado, pois um objeto Set não armazena duas referências para um mesmo objeto -- apenas uma cópia é admitida.  
    A interface SortedSet é uma extensão de Set que agrega o conceito de ordenação ao conjunto. Para tanto, os elementos de um SortedSet devem ser comparáveis.   
    Novos métodos que fazem uso dessa propriedade foram introduzidos, tais como:

* first() para obter o primeiro elemento do conjunto.
* last() para obter o último elemento do conjunto.
* subSet() para obter o subconjunto com todos os elementos contidos entre dois elementos especificados.

    A interface List  é uma extensão de Collection que acrescenta ao agrupamento o conceito de indexação, ou seja, informação sobre a posição do elemento em relação aos demais membros do grupo.   
    Além dos métodos básicos de coleções, um objeto List tem métodos tais como:

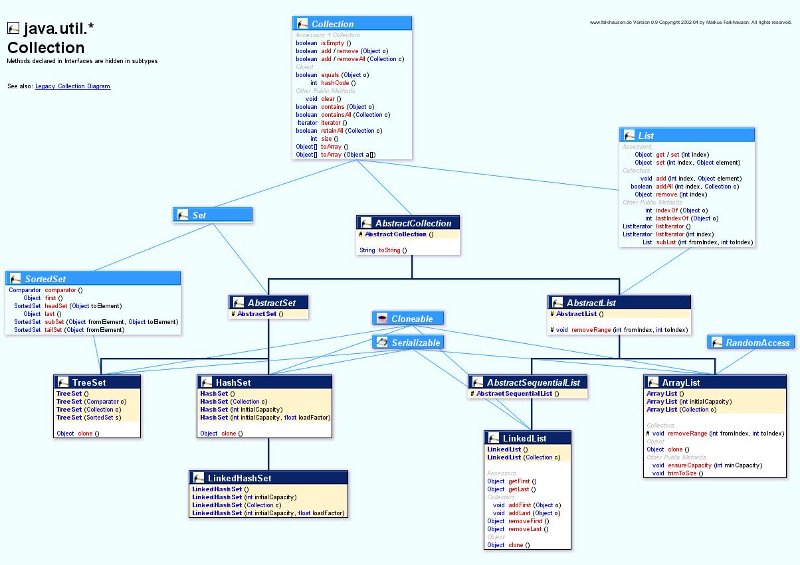
* get() para obter o elemento armazenado na posição especificada;
* indexOf() para obter a posição da primeira ocorrência do elemento especificado; e
* subList() para obter uma sublista contendo os elementos compreendidos entre dois elementos especificados, incluindo o primeiro deles mas não o segundo.

    A interface  Map especifica as funcionalidades necessárias para manipular um grupo de objetos onde cada elemento é um par que associa um objeto chave a um objeto valor.   
Entre outros, os seguintes métodos são oferecidos:

* put(), que associa um novo objeto valor ao objeto chave especificado;
* <="" li="">
* containsKey(), que indica se a chave especificada está presente na coleção;
* size(), que retorna a quantidade de elementos (pares chave-valor) na coleção.

    Se adicionalmente deseja-se que esse grupo de pares de objetos seja mantido por ordem dos valores das chaves, a interface mais especializada  SortedMap pode ser utilizada, desde que os objetos usados como chave sejam comparáveis.   
     Além das funcionalidades acima, esta interface acrescenta a especificação de métodos tais como:

* firstKey() para retornar o elemento da coleção com a chave de menor valor.
* lastKey() para retornar o elemento da coleção com a chave de maior valor.
* subMap() para obter o subconjunto dos elementos compreendidos entre as chaves especificadas.

    A API de Java oferece algumas implementações para essas interfaces, tais como HashMap e TreeMap  
  
**4.Exemplo ArrayList**

*// Pacotes Java*  
 import java.awt.Color;  
 import java.util.\*;  
 import javax.swing.JOptionPane;  
   
 **public** class TesteColecao {  
 **private** String colors[] = { "vervelha", "branca", "azul" };  
   
 *// cria um ArrayList, adiciona objetos e manipula*  
 **public** **TesteColecao**()  
 {  
 ArrayList list = **new** **ArrayList**();   
 String var = "";  
   
 *// Adiciona um objeto para lista*  
 list.**add**( Color.magenta ); *// Adiciona um objeto color*  
   
 **for** ( int count = 0; count < colors.length; count++ )  
 list.**add**( colors[ count ] );   
   
 list.**add**( Color.cyan ); *// Adiciona um objeto color*  
   
 *// saída do conteúdo da lista*  
 var = "ArrayList: \n\r";  
  
   
 **for** ( int count = 0; count < list.**size**(); count++ )  
 var += list.**get**( count ) + "\n\r ";  
  
   
 JOptionPane.**showMessageDialog**( **null**, var, "ArrayList", JOptionPane.INFORMATION\_MESSAGE );  
 *// remove todos os objetos String*  
 **removeStrings**( list );  
   
 *// saída do conteúdo da lista*  
 var = "ArrayList depos da chamada remove Strings: \n\r";  
   
 **for** ( int count = 0; count < list.**size**(); count++ ){  
 var += list.**get**( count ) + "\n\r ";  
  
 }  
 JOptionPane.**showMessageDialog**( **null**, var, "ArrayList", JOptionPane.INFORMATION\_MESSAGE );  
 }  
   
 *// remove objetos String objects da Coleção*  
 **public** void **removeStrings**( Collection collection )  
 {  
 *// recebe iterator*  
 Iterator iterator = collection.**iterator**();   
   
 *// laço de repetição while na coleção verifica item*  
 **while** ( iterator.**hasNext**() )   
   
 **if** ( iterator.**next**() **instanceof** String )  
 iterator.**remove**(); *// remove objeto String*  
 }  
   
 *// executa aplicação*  
 **public** **static** void **main**( String args[] )  
 {  
 **new** **TesteColecao**();  
 }   
   
 } *// fim da classe TesteColecao*

|  |  |
| --- | --- |
| ColecaoLista | ColecaoLista |

   **5. Exemplo LinkedList**

import java.util.\*;  
 import javax.swing.JOptionPane;  
  
 **public** class TesteList {  
 **private** String colors[] = { "preta", "amarela", "verde",  
 "azul", "violeta", "prata" };  
 **private** String colors2[] = { "ouro", "branca", "marrom",  
 "azul", "cinza", "prata" };  
  
 *// atribuir e manipular objetos LinkedList*  
 **public** **TesteList**()  
 {  
 LinkedList link = **new** **LinkedList**();  
 LinkedList link2 = **new** **LinkedList**();  
  
 *// adiciona elementos para cada lista*  
 **for** ( int count = 0; count < colors.length; count++ ) {  
 link.**add**( colors[ count ] );  
 link2.**add**( colors2[ count ] );   
 }  
  
 link.**addAll**( link2 ); *// concatena lista*  
 link2 = **null**; *// release resources*  
  
 **imprimeLista**( link );  
  
 **uppercaseStrings**( link );  
  
 **imprimeLista**( link );  
   
 JOptionPane.**showMessageDialog**( **null**, "Deletando elementos de 4 a 6...", "Lista", JOptionPane.INFORMATION\_MESSAGE );  
   
 **removeItems**( link, 4, 7 );  
  
 **imprimeLista**( link );  
 }  
  
 *// saida conteudo List*  
 **public** void **imprimeLista**( List list )  
 {  
   
 String var = "";  
 var = "lista: \n\r" ;  
  
 **for** ( int count = 0; count < list.**size**(); count++ ){  
 var += list.**get**( count ) + "\n\r";  
 }  
  
 JOptionPane.**showMessageDialog**( **null**, var, "Lista", JOptionPane.INFORMATION\_MESSAGE );  
 }   
  
 *// objeto String local e converte para uppercase*  
 **public** void **uppercaseStrings**( List list )  
 {  
 ListIterator iterator = list.**listIterator**();  
  
 **while** ( iterator.**hasNext**() ) {  
 Object object = iterator.**next**(); *// get item*  
  
 **if** ( object **instanceof** String ) *// check for String*  
 iterator.**set**(   
 ( ( String ) object ).**toUpperCase**() );   
 }  
 }  
  
 *// obtem sublista e usa metodo clear para deletar od itens*  
 **public** void **removeItems**( List list, int start, int end )  
 {  
 list.**subList**( start, end ).**clear**(); *// remove items*  
 }  
  
 *// executa aplicação*  
 **public** **static** void **main**( String args[] )  
 {  
 **new** **TesteList**();  
 }   
  
 } *// fim da classe TesteList*

|  |  |
| --- | --- |
| TesteList | TesteList |
| TesteList | TesteList |

**6.Exemplo HashSet**

*// Pacotes Java*  
 import java.util.\*;  
 import javax.swing.JOptionPane;  
  
 **public** class TesteSet {  
 **private** String colors[] = { "vermelha", "branca", "azul",  
 "verde", "cinza", "laranja", "bronze", "branca", "cinza",  
 "pesego", "ciza", "laranja" };  
  
 *// Cria e imprime ArrayList*  
 **public** **TesteSet**()  
 {  
 ArrayList list;  
  
 list = **new** **ArrayList**( Arrays.**asList**( colors ) );  
 String var = "";  
 JOptionPane.**showMessageDialog**( **null**, list, "ArrayList", JOptionPane.INFORMATION\_MESSAGE );  
 **imprimeNaoDuplicados**( list );  
 }  
  
 *// cria um conjunto de array para eleiminar duplicados*  
 **public** void **imprimeNaoDuplicados**( Collection collection )  
 {  
 *// cria um HashSet e obtem iterator*  
 HashSet set = **new** **HashSet**( collection );   
 Iterator iterator = set.**iterator**();   
 String var = "NÃ£o duplicados: \n\r";  
   
 **while** ( iterator.**hasNext**() ){  
 var += iterator.**next**() + "\n\r";  
  
 }  
  
 JOptionPane.**showMessageDialog**( **null**, var, "HashSet", JOptionPane.INFORMATION\_MESSAGE );  
 }  
  
 *// executa aplicação*  
 **public** **static** void **main**( String args[] )  
 {  
 **new** **TesteSet**();  
 }   
  
 } *// fim da classe TesteSet*

|  |
| --- |
| TesteSet |
| TesteSet |

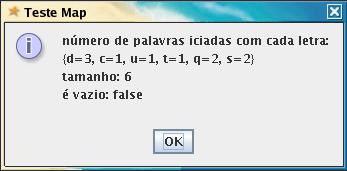
**7.Exemplo TreeSet**

import java.util.\*;  
 import javax.swing.JOptionPane;  
  
 **public** class TesteOrdenacaoSet {  
 **private** **static** String names[] = { "amarela", "verde", "preta",  
 "pesego", "cinza", "branca", "laranja", "vermelha", "verde" };  
 **private** String var = "";  
  
 *// cria uma coleção ordenado do TreeSet, e manipula-o*  
 **public** **TesteOrdenacaoSet**()  
 {  
 TreeSet tree = **new** **TreeSet**( Arrays.**asList**( names ) );  
 var = "Coleção total \n\r";  
  
 **imprimeConjunto**( tree );  
  
 *// pega a base da coleção sobre "laranja"*  
 var = "headSet (\"laranja\") \n\r";  
 **imprimeConjunto**( tree.**headSet**( "laranja" ) );  
  
 *// pega o final da coleção sobre "lararanja"*  
 var = "tailSet (\"laranja\") \n\r";  
 System.out.**print**( "tailSet (\"laranja\"): " );  
 **imprimeConjunto**( tree.**tailSet**( "laranja" ) );  
  
 *// pega o primeiro e o último elemento*  
 var = "primeiro: " + tree.**first**() + "\n\r";  
 var += "ultimo : " + tree.**last**();  
 JOptionPane.**showMessageDialog**( **null**, var, "TreeSet", JOptionPane.INFORMATION\_MESSAGE );  
 }  
  
 *// imprime a coleção*  
 **public** void **imprimeConjunto**( SortedSet set )  
 {  
 Iterator iterator = set.**iterator**();  
   
  
 **while** ( iterator.**hasNext**() ){   
 var += iterator.**next**() + "\n\r ";  
 }  
  
 JOptionPane.**showMessageDialog**( **null**, var, "TreeSet", JOptionPane.INFORMATION\_MESSAGE );  
 System.out.**println**();  
 }   
  
 *// executa a aplicação*  
 **public** **static** void **main**( String args[] )  
 {  
 **new** **TesteOrdenacaoSet**();  
 }   
  
 } *// fim da classe TesteOrdenacaoSet*

|  |  |
| --- | --- |
| TesteOrdenacaoSet | TesteOrdenacaoSet |
| TesteOrdenacaoSet | TesteOrdenacaoSet |

**8. Exemplo HashMap**

*// Pacotes Java*  
 import java.util.\*;  
 import javax.swing.JOptionPane;  
  
 **public** class TesteMap {  
 **private** **static** String names[] = { "um", "dois", "tres",  
 "quatro", "cinco", "seis", "sete", "dois", "dez", "quatro" };  
 **private** String var = "";  
  
 *// constroe um HashMap e imprime conteudo*  
 **public** **TesteMap**()  
 {  
 HashMap map = **new** **HashMap**();  
 Integer i;  
  
 **for** ( int count = 0; count < names.length; count++ ) {  
 i = ( Integer ) map.**get**(   
 **new** **Character**( names[ count ].**charAt**( 0 ) ) );  
  
 *// se chave não esta no map então é dado um valor e*  
 *// do contrário incrementa este valor em 1*  
 **if** ( i == **null** )  
 map.**put**(   
 **new** **Character**( names[ count ].**charAt**( 0 ) ),  
 **new** **Integer**( 1 ) );  
 **else**  
 map.**put**(   
 **new** **Character**( names[ count ].**charAt**( 0 ) ),  
 **new** **Integer**( i.**intValue**() + 1 ) );  
 }  
  
 var = "Número de palavras iciadas com cada letra: \n\r";  
   
 **imprimetMap**( map );  
 }  
  
 *// saí­da do conteúdo do map*  
 **public** void **imprimetMap**( Map mapRef )  
 {  
 var += mapRef.**toString**() + "\n\r";  
 var += "tamanho: " + mapRef.**size**() + "\n\r";  
 var += "É vazio: " + mapRef.**isEmpty**() + "\n\r";  
 JOptionPane.**showMessageDialog**( **null**, var, "Teste Map", JOptionPane.INFORMATION\_MESSAGE );   
 }   
  
 *// executa a aplicação*  
 **public** **static** void **main**( String args[] )  
 {  
 **new** **TesteMap**();  
 }   
  
 } *// fim classe Teste Map*

  
  
Autores: Carlos Fernando Gonçalves e Marli Esprega Gonçalves ([Antares Information Systems](http://www.webgis.com.br/antares))

* **O que é Java?**
  + [Conheça mais sobre a tecnologia Java, aqui.](http://javanoroeste.com.br/javanoroeste/novo/artigos/tecnologia_java.html)
* **Destaques**
  + [Artigo Iniciando SUN Spot com NetBeans 6.1.](http://javanoroeste.com.br/javanoroeste/novo/artigos/iniciandosunspot.html)
  + [Java ME - Trabalho Sistema de Apoio Policial](http://javanoroeste.com.br/javanoroeste/novo/artigos/javame.html)
  + [Tecnologia Java Card](http://javanoroeste.com.br/javanoroeste/novo/artigos/javacard.html)
  + [Imagens SVG e Java ME](http://javanoroeste.com.br/javanoroeste/novo/artigos/svg_javame.html)
  + [Socket + JME](http://javanoroeste.com.br/javanoroeste/novo/artigos/socketJ2ME.pdf)
* **Plataforma Java**
  + [Java básico](http://javanoroeste.com.br/javanoroeste/novo/javabasico.html)
  + [Java ME](http://javanoroeste.com.br/javanoroeste/novo/javame.html)
  + [Java Web](http://javanoroeste.com.br/javanoroeste/novo/javaweb.html)
* **Instalação Java**
  + [Transfira o Java Development Kit](https://cds.sun.com/is-bin/INTERSHOP.enfinity/WFS/CDS-CDS_Developer-Site/en_US/-/USD/ViewProductDetail-Start?ProductRef=jdk-6u13-oth-JPR@CDS-CDS_Developer)
  + [Instalação do Java em ambiente Linux](http://javanoroeste.com.br/javanoroeste/novo/artigos/arrays_colecoes.html)
  + [Instalação do Java em ambiente Windows](http://javanoroeste.com.br/javanoroeste/novo/artigos/arrays_colecoes.html)

Copyright © 2004, Java Noroeste