**ArrayList**

Vamos começar pela mais conhecida e usada, ArrayList. Este tipo de lista é implementado como um Array que é dimensionado dinamicamente, ou seja, sempre que é necessário o seu tamanho aumenta em 50% do tamanho da lista, significa que se você tiver uma lista de tamanho igual a 10 e ela “encher”, seu tamanho aumentará para 15 automaticamente.

Além disso a ArrayList permite que elementos sejam acessados diretamente pelos métodos get() e set(), e adicionados através de add() e remove().

**Listagem 1**: Usando ArrayList

|  |  |
| --- | --- |
|  | import java.util.ArrayList;  import java.util.Iterator;    public class Principal {       public static void main(String args[]) {            ArrayList al = new ArrayList();          al.add(3);          al.add(2);          al.add(1);          al.add(4);          al.add(5);          al.add(6);          al.add(6);            Iterator iter1 = al.iterator();          while(iter1.hasNext()){                  System.out.println(iter1.next());          }          System.out.println(al.get(2));      }  } |

O que você vai perceber no código acima é que o ArrayList não remove os elementos duplicados, e ainda podemos acessar qualquer elemento diretamente através do seu index, mas tudo tem um custo e veremos mais adiante.

Todo ArrayList começa com um tamanho fixo, que vai aumentando conforme necessário, mas o custo deste aumento é alto, pois é feita uma cópia do array atual para um novo array com um novo tamanho, então imagine um array com 10mil elementos que será copiado para um novo array para criação de mais 5 mil elementos ? De fato é um alto custo. Então é altamente aconselhável que você já inicie seu Array com uma quantidade de elementos que atenda ao seu objetivo atual, sem a necessidade de criação dinâmica de novos espaços, ou seja, se você saber que terá que armazenar de 300 a 400 objetos em um Array, defina 500, pois é melhor sobrar espaço do que utilizar recurso do processador sem necessidade.

Há ainda mais um ponto muito importante sobre ArrayList: Estes não são sincronizados, consequentemente não são thread-safe, ou seja, se sua aplicação precisa trabalhar com thread-safe em determinado ponto onde uma Lista é necessária, então descarte ArrayList, a não ser que você faça isso explicitamente, que obviamente não é o correto, veremos mais a diante uma Lista que é sincronizada.

**Vetor**

Do ponto de vista da API, ou seja, da forma como é utilizado, o Vetor e o ArayList são muito similares, podemos arriscar até em dizer: iguais. Se você não conhece a fundo o conceito de Vetor e ArrayList usará ambos como se fossem o mesmo, sem sentir nenhuma diferença, veja na listagem 2 um exemplo disso.

**Listagem 2**: Usando Vetor

|  |  |
| --- | --- |
|  | import java.util.Iterator;  import java.util.Vetor;    public class Principal {       public static void main(String args[]) {          Vetor al = new Vetor();          al.add(3);          al.add(2);          al.add(1);          al.add(4);          al.add(5);          al.add(6);          al.add(6);            Iterator iter1 = al.iterator();          while(iter1.hasNext()){                  System.out.println(iter1.next());          }          System.out.println(al.get(2));      }   } |

Perceba uma coisa na listagem 2: usamos a mesma estrutura da listagem 1, mudando apenas a lista de ArrayList para Vetor, mas o restante do código permanece o mesmo, e a saída também será idêntica.

Então qual a diferença entre Vetor e ArrayList ?

* Primeiramente vamos falar sobre o fato de Vetor ser sincronizado e o ArrayList não. Significa dizer que se você possui uma aplicação que precisa ser thread-safe em determinado ponto, use Vetor e você estará garantido.
* Outro ponto importante é a alocação dinâmica do Vetor, que é diferente do ArrayList. Lembra que falamos que o ArrayList aumenta 50% do seu tamanho quando a lista está cheia ? O Vetor aumenta o dobro, ou seja, se você tem uma lista de 10 elementos cheia, essa lista aumentará para 20, com 10 posições vazias. Mas isso não é ruim ? Depende do que você precisar, se você está precisando aumentar a quantidade de elementos com muita frequência, então o ideal é usar o Vetor que aumenta o dobro e você ficará com muito mais espaço do que no ArrayList que precisará ficar aumentando com mais frequência, diminuindo assim a performance da sua aplicação.

A **classe ArrayList** na implementação de vetores dinâmicos (ou redimensionáveis). Veja a seguir a relação dos principais métodos da classe.

* boolean add(Object element): Adiciona o elemento especificado no final da lista.
* void add(int index, Object element): Insere o elemento especificado na posição indicada da lista.
* void clear(): Remove todos os elementos da lista.
* boolean contains(Object element): Retorna verdadeiro se a lista contém o elemento especificado e falso caso contrário.
* Object get(int index): Retorna o i-ésimo elemento da lista.
* int indexOf(Object element): Retorna a posição da primeira ocorrência do elemento especificado na lista.
* boolean isEmpty(): Retorna verdadeiro se a lista estiver vazia e falso caso contrário.
* int lastIndexOf(Object element): Retorna a posição da última ocorrência do elemento especificado na lista.
* Object remove(int index): Remove o i-ésimo elemento da lista.
* Object set(int index, Object element): Substitui o i-ésimo elemento da lista pelo elemento especificado.
* int size(): Retorna o número de elementos da lista.

Na **Listagem 1**, foram implementados trechos de códigos que demonstram a **utilização da classe ArrayList** para criar, manter e percorrer uma lista de contatos. As seguintes funcionalidades foram implementadas: a) declarando e instanciando um objeto agenda; b) usando o método add() para gravar 4 contatos na agenda; c) mostrando os "n" contatos da agenda (usando o índice); d) removendo o i-ésimo elemento da agenda; e) mostrando os "n" contatos da agenda (usando for-each); e, f) mostrando os "n" contatos da agenda (com iterator).

|  |  |
| --- | --- |
|  | import java.util.ArrayList;  import java.util.Iterator;  import java.util.Scanner;    public class Exemplo {     public static void main(String[] args) {      Scanner ler = new Scanner(System.in);        // [ A ] declarando e instanciando uma agenda de contatos      ArrayList<String> agenda = new ArrayList();        // [ B ] usando o método add() para gravar 4 contatos na agenda      agenda.add("Juca Bala;11 1111-1111");      agenda.add("Marcos Paqueta;22 2222-2222");      agenda.add("Maria Antonieta;33 3333-3333");      agenda.add("Antônio Conselheiro;44 4444-4444");        int i;        // [ C ] mostrando os "n" contatos da agenda (usando o índice)      // número de elementos da agenda: método size()      System.out.printf("Percorrendo o ArrayList (usando o índice)\n");      int n = agenda.size();      for (i=0; i<n; i++) {        System.out.printf("Posição %d- %s\n", i, agenda.get(i));      }        System.out.printf("\nInforme a posição a ser excluída:\n");      i = ler.nextInt();        try {        // [ D ] remove o i-ésimo contato da agenda        agenda.remove(i);      } catch (IndexOutOfBoundsException e) {          // exceção lançada para indicar que um índice (i)          // está fora do intervalo válido (de 0 até agenda.size()-1)          System.out.printf("\nErro: posição inválida (%s).",            e.getMessage());      }        // [ E ] mostrando os "n" contatos da agenda (usando for-each)      System.out.printf("\nPercorrendo o ArrayList (usando for-each)\n");      i = 0;      for (String contato: agenda) {        System.out.printf("Posição %d- %s\n", i, contato);        i++;      }        // [ F ] mostrando os "n" contatos da agenda (com iterator)      System.out.printf("\nPercorrendo o ArrayList (usando iterator)\n");      i = 0;      Iterator<String> iterator = agenda.iterator();      while (iterator.hasNext()) {        System.out.printf("Posição %d- %s\n", i, iterator.next());        i++;      }    }    } |

### Estruturas de dados estáticas e dinâmicas

[Estruturas de dados](http://www.javaworld.com/article/2073390/core-java/datastructures-and-algorithms-part-1.html), assim como arrays, podem ser estáticas ou dinâmicas. Por exemplo, um array pode ser usado para armazenar um grupo de elementos. Geralmente, arrays são estáticos naturalmente e, portanto, não permitem adicionar um novo elemento. Mas os arrays também podem ser definidos como dinâmicos, ou seja, você pode alterar seu comprimento e adicionar novos elementos rapidamente.

Em Java, você pode declarar uma matriz usando o seguinte código:

|  |  |
| --- | --- |
|  | Int[] Arr; //declaring array   Arr = new int[5]; //initializing an array with the size of 5. |

Arr agora pode armazenar 5 elementos. Você não pode adicionar elementos à matriz Arr, tornando-a estática por natureza.

Então, como resolver o problema de criar um array dinâmico? Java ArrayList faz o trabalho!

### ArrayList

ArrayList é uma matriz dinâmica que permite que um programador armazene qualquer número de itens. Se estiver confuso, vamos entender primeiro os conceitos estáticos e dinâmicos em ciência da computação.

ArrayList expande a interface List, o que ajuda a tirar o máximo proveito dela. Usar a estrutura de dados Array pode ser bom para algumas situações, mas ArrayList dá um melhor controle sobre os dados utilizados no programa.

O ArrayList melhora o funcionamento de uma matriz. Como programador, a flexibilidade de adicionar novos elementos pode ser usada extensivamente em diferentes cenários. ArrayList também pode ser útil no gerenciamento de consumo de memória de programa, já que elementos desnecessários são descartados. Este consumo de memória melhorado é ótimo para aplicações em tempo real.

No entanto, deve-se notar que ArrayList nunca deve ser usado para informações que você não deseja alterar. Por exemplo, se você estiver armazenando o número de meses, ou seja, de 1 a 12, é aconselhável usar uma matriz que forneça comportamento estático.

### Importando o pacote certo

Para fazer o uso adequado do ArrayList, primeiro você precisa importar o pacote da biblioteca java.util.

|  |  |
| --- | --- |
|  | import java.util.ArrayList; |

Uma vez importado, tudo o que você precisa fazer é criar uma nova instância usando a seguinte linha de código:

|  |  |
| --- | --- |
|  | ArrayList myList = new ArrayList(); |

Isso criará uma nova ArrayList chamada myList.

### Operações

Depois de termos criado o ArrayList, podemos usar diferentes métodos nele. Vamos passar por alguns deles.

|  |  |
| --- | --- |
|  | add(object A)   myList.add(“new-item”);   myList.add(“placement”);   myList.add(“five”);   myList.add(5); |

Como você pode ver, nós usamos o método add () para adicionar novos elementos ao myList ArrayList. No entanto, precisamos entender que todos eles são objetos e não tipos de dados puros. Os três primeiros elementos são objetos de seqüência de caracteres e o quarto é um objeto inteiro. Se você desejar adicionar um elemento em um determinado local, você precisa usar o método add passando dois argumentos.

|  |  |
| --- | --- |
|  | add(int index, Object A)  get(int index) |

Com o método get (), você pode facilmente recuperar informações com a ajuda de um índice. Tudo o que você precisa passar é o valor do índice e o valor do elemento será recuperado.

|  |  |
| --- | --- |
|  | myList.get(2)  ⇒ five. //ArrayList index starts from 0.  remove(Object A) |

Com a ajuda de remove (), você pode remover elementos. O método remove remove o índice como um argumento e o ajudará a gerenciar bem os recursos do programa.

|  |  |
| --- | --- |
|  | myList.remove(0); //this will remove “new-item” string object. |

Há muitos outros métodos que você pode usar no ArrayList. Alguns deles estão listados abaixo:

|  |  |
| --- | --- |
|  | set(int index, Object A)  int indexOf(Object A) |

Você pode encontrar toda a lista [aqui](https://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html).

### Iterando sobre a ArrayList

Existe uma maneira de iterar sobre ArrayList? Se sim, como fazê-lo? Você pode usar o objeto Iterator (). Tudo o que você precisa fazer é importar a classe Iterator da biblioteca java.util usando a seguinte linha de código:

|  |  |
| --- | --- |
|  | import java.util.Iterator; |

Como mencionado anteriormente, a classe Iterator tem alguns métodos que você pode utilizar para iterar sobre ArrayList.

HasNext () é o método perfeito para trabalhar com ArrayList. Tudo o que você precisa para executar um loop e verificar se ele retorna true ou false.

Primeiro, você precisa anexar ArrayList ao objeto iterator.

|  |  |
| --- | --- |
|  | Iterator ity = myList.iterator();  Now, we can use the “ity” iterator object.  while(ity.hasNext()) {    System.out.println(it.next());  } |

O método next () permite passar por todos os elementos da lista.

Veja como é:

|  |  |
| --- | --- |
|  | import java.util.\*;  public class ArrayListEx {  public static void main(String args[]) {    ArrayList<String> myList= new ArrayList<String>();    /\*Adding new elements to the myList\*/  myList.add("Ruby");    myList.add(“Java");  myList.add("JavaScript");  myList.add("HTML");  myList.add("CSS");    /\* Quick way of displaying all the arraylist items \*/  System.out.println("Currently the array list has following elements:"+myList);    /\*Adding elements using index\*/  obj.add(0, "C++");  obj.add(1, "C");    /\*Removing elements\*/  obj.remove("C");  obj.remove("Ruby");  System.out.println("Current array list is:"+myList);    /\*Removing element using index\*/  obj.remove(1);   System.out.println("Current array list is:"+myList);    }  } |

## **Java ArrayList**

A ArrayListclasse é uma [matriz](https://www.w3schools.com/java/java_arrays.asp) redimensionável , que pode ser encontrada no java.utilpacote.

A diferença entre uma matriz interna e uma ArrayListem Java é que o tamanho de uma matriz não pode ser modificado (se você deseja adicionar ou remover elementos de / para uma matriz, é necessário criar um novo). Os elementos while podem ser adicionados e removidos de um ArrayListquando você quiser. A sintaxe também é um pouco diferente:

### Exemplo

Crie um ArrayListobjeto chamado **cars** que armazenará strings:

import java.util.ArrayList; // import the ArrayList class  
  
ArrayList<String> cars = new ArrayList<String>(); // Create an ArrayList object

Se você não sabe o que é um pacote, leia nosso [Tutorial de Pacotes Java](https://www.w3schools.com/java/java_packages.asp) .

## **Adicionar itens**

A ArrayListclasse tem muitos métodos úteis. Por exemplo, para adicionar elementos ao ArrayList, use o add()método:

### Exemplo

import java.util.ArrayList;  
  
public class MyClass {  
  public static void main(String[] args) {  
    ArrayList<String> cars = new ArrayList<String>();  
    cars.add("Volvo");  
    cars.add("BMW");  
    cars.add("Ford");  
    cars.add("Mazda");  
    System.out.println(cars);  
  }  
}

## **Acessar um item**

Para acessar um elemento no ArrayList, use o get()método e consulte o número do índice:

### Exemplo

cars.get(0);

**Lembre-se: os** índices de matriz começam com 0: [0] é o primeiro elemento. [1] é o segundo elemento, etc.

## **Alterar um item**

Para modificar um elemento, use o set()método e consulte o número do índice:

### Exemplo

cars.set(0, "Opel");

## **Remover um item**

Para remover um elemento, use o remove()método e consulte o número do índice:

### Exemplo

cars.remove(0);

Para remover todos os elementos no ArrayList, use o clear()método:

### Exemplo

cars.clear();

## **Tamanho da lista**

Para descobrir quantos elementos um ArrayList possui, use o sizemétodo:

### Exemplo

cars.size();

## **Loop através de um ArrayList**

Faça um loop pelos elementos de um ArrayListcom um forloop e use o size()método para especificar quantas vezes o loop deve ser executado:

### Exemplo

public class MyClass {  
  public static void main(String[] args) {  
    ArrayList<String> cars = new ArrayList<String>();  
    cars.add("Volvo");  
    cars.add("BMW");  
    cars.add("Ford");  
    cars.add("Mazda");  
    for (int i = 0; i < cars.size(); i++) {  
      System.out.println(cars.get(i));  
    }  
  }  
}

Você também pode percorrer um loop ArrayListcom o **for-each** :

### Exemplo

public class MyClass {  
  public static void main(String[] args) {  
    ArrayList<String> cars = new ArrayList<String>();  
    cars.add("Volvo");  
    cars.add("BMW");  
    cars.add("Ford");  
    cars.add("Mazda");  
    for (String i : cars) {  
      System.out.println(i);  
    }  
  }  
}

## **Outros tipos**

Elementos em um ArrayList são realmente objetos. Nos exemplos acima, criamos elementos (objetos) do tipo "String". Lembre-se de que uma String em Java é um objeto (não um tipo primitivo). Para usar outros tipos, como int, você deve especificar um equivalente [classe de mensagens publicitárias](https://www.w3schools.com/java/java_wrapper_classes.asp) : Integer. Para outros tipos primitivos, use: Booleanpara booleano, Characterchar, Doubledouble, etc:

### Exemplo

Crie um ArrayListnúmero para armazenar (adicione elementos do tipo Integer):

import java.util.ArrayList;  
  
public class MyClass {  
  public static void main(String[] args) {  
    ArrayList<Integer> myNumbers = new ArrayList<Integer>();  
    myNumbers.add(10);  
    myNumbers.add(15);  
    myNumbers.add(20);  
    myNumbers.add(25);  
    for (int i : myNumbers) {  
      System.out.println(i);  
    }  
  }  
}

## **Classificar um ArrayList**

Outra classe útil no java.utilpacote é a Collectionsclasse, que inclui o sort()método para classificar as listas alfabeticamente ou numericamente:

### Exemplo

Classificar uma matriz de strings:

import java.util.ArrayList;  
import java.util.Collections;  // Import the Collections class  
  
public class MyClass {  
  public static void main(String[] args) {  
    ArrayList<String> cars = new ArrayList<String>();  
    cars.add("Volvo");  
    cars.add("BMW");  
    cars.add("Ford");  
    cars.add("Mazda");  
  
    Collections.sort(cars);  // Sort cars  
  
    for (String i : cars) {  
      System.out.println(i);  
    }  
  }  
}

### Exemplo

Classifique uma ArrayList de números inteiros:

import java.util.ArrayList;  
import java.util.Collections;  // Import the Collections class  
  
public class MyClass {  
  public static void main(String[] args) {  
    ArrayList<Integer> myNumbers = new ArrayList<Integer>();  
    myNumbers.add(33);  
    myNumbers.add(15);  
    myNumbers.add(20);  
    myNumbers.add(34);  
    myNumbers.add(8);  
    myNumbers.add(12);  
  
    Collections.sort(myNumbers);  // Sort myNumbers  
  
    for (int i : myNumbers) {  
      System.out.println(i);  
    }  
  }  
}