HowToDoInJava

Java ArrayList

Last Updated: December 26, 2020

An **ArrayList in Java** represents a resizable list of objects. We can add, remove, find, sort and replace elements in this list.

`ArrayList` is the part of the collections framework. It extends `AbstractList` which implements `List` interface. The `List` extends `Collection` and `Iterable` interfaces in hierarchical order.

ArrayList Hierarchy

1. ArrayList Features

`ArrayList` has the following features –

1. **Ordered** – Elements in arraylist preserve their ordering which is by default the order in which they were added to the list.
2. **Index based** – Elements can be randomly accessed using index positions. Index start with `'0'`.
3. **Dynamic resizing** – `ArrayList` grows dynamically when more elements needs to be added than it's current size.

4. **Non synchronized** – `ArrayList` is not synchronized, by default. Programmer needs to use `synchronized` keyword appropiately or simply use **Vector** class.
5. **Duplicates allowed** – We can add duplicate elements in arraylist. It is not possible in sets.

2. How ArrayList Works?

`ArrayList` class is implemented with a backing array. The elements added or removed from arraylist are actually modified in the backing array. All `ArrayList` methods access this backing array and get/set elements in the same array.

`ArrayList` can be seen as *resizable-array implementation* in Java.

```
ArrayList.java

public class ArrayList<E> extends AbstractList<E>
        implements List<E>, RandomAccess,
               Cloneable, java.io.Serializable
{
  transient Object[] elementData;   //backing array
  private int size;         //array or list size

  //more code
}
```

3. Java Array vs ArrayList

An array is fixed size data structure where the size has to be declared during initialization. Once the size of an array is declared, it is not possible to resize the array without creating a new array.

```
Array

Integer[] numArray = new Integer[5];
```

The `ArrayList` offers to remove this sizing limitation. An `ArrayList` can be created with any initial size (default 16), and when we add more items, the size of the arraylist grows dynamically without any intervention by the programmer.

```
ArrayList

ArrayList<Integer> numList = new ArrayList<>();
```

> Many people refer to `ArrayList` as *dynamic array*.

4. Creating an ArrayList

4.1. How to create an `ArrayList`

To create `ArrayList`, we can call one of its constructors.

Constructor	ArrayList()
Description	It is default constructor. It creates an empty arraylist with initial capacity 16.
Constructor	ArrayList(int capacity)
Description	It creates an empty arraylist with the given initial `capacity`.
Constructor	ArrayList(Collection<? extends E> c)
Description	It creates an arraylist that is initialized with the elements of the collection `c`.

Given below program shows how to declare and initialize an arraylist in Java.

```
Create arraylist

ArrayList list = new ArrayList();

List<Integer> numbers = new ArrayList<>(6);

Collection setOfElements = ...;
List<Integer> numbers = new ArrayList<>(setOfElements);
```

4.2. Generic `ArrayList`

A generic arraylist clearly mentions the type of objects, it will store. It helps in avoiding a lot of defects caused by incorrect typecasting.

```
Create arraylist

//Non-generic arraylist - NOT RECOMMENDED !!
ArrayList list = new ArrayList();

//Generic Arraylist with default capacity
List<Integer> numbers = new ArrayList<>();

//Generic Arraylist with the given capacity
List<Integer> numbers = new ArrayList<>(6);

//Generic Arraylist initialized with another collection
List<Integer> numbers = new ArrayList<>( Arrays.asList(1,2,3,4,5) );
```

4.3. `ArrayList` of primitive types

In array list, we are supposed to add only objects. But in case, we are required to add primitive data types such as `int`, `float` etc, we can use their wrapper classes for providing type information during arraylist initialization.

When we add the `int` or `float` value to array list, values are automatically upcasted.

In given example, we have created an array list of `Integer` values. When we add `int` value `1`, it is automatically converted to `new Integer(1)`.

```
Store primitives in Arraylist

List<Integer> numbers = new ArrayList<>(6);

numbers.add(1); // This runs fine
```

4.4. Create and initialize `ArrayList` in single line

Generally, creating an arraylist in multi-step process. In first step, we create empty array list. In later steps, we populate the list with elements – one by one.

Using `Arrays.asList()` and constructor `ArrayList(collection)`, we can combine these steps in single statement.

```
ArrayList<String> charList = new ArrayList<>(Arrays.asList(("A", "B", "C"));
```

5. Get element from ArrayList

To get an element from the `ArrayList`, we have two ways.

5.1. get(index)

If we know the index location in advance, then we can call the `get(index)` which returns the *element* present at `index` location.

Please remember that `indexes` start with zero.

```
get method

ArrayList<String> alphabetsList = new ArrayList<>(Arrays.asList(("A", "B", "C"));

String aChar = alphabetsList.get(0);  // A
```

5.2. iterator.next()

Use `iterator()` or `listIterator()` to get the reference of `Iterator` instance. This iterator can be used to iterate the elements in the arraylist.

The `next()` method returns the element at current `index` location and increment the index count by one. Call `hasNext()` method to check if there are more elements in the list to iterate.

```
Iterate arraylist

ArrayList<Integer> digits = new ArrayList<>(Arrays.asList(1,2,3,4,5,6));

Iterator<Integer> iterator = digits.iterator();

while(iterator.hasNext())
```

```
{
  System.out.println(iterator.next());
}
```

Program output.

```
Console

1
2
3
4
5
6
```

6. Iterating over an ArrayList

6.1. Iterator

Java example to iterate over an arraylist using the Iterator.

```
Iterate arraylist with Iterator interface

ArrayList<Integer> digits = new ArrayList<>(Arrays.asList(1,2,3,4,5,6));

Iterator<Integer> iterator = digits.iterator();

while(iterator.hasNext())
{
  System.out.println(iterator.next());
}
```

6.2. For loop

Java example to iterate over an arraylist using for loop. When using for loop, we need to get the current element using the current index counter.

```
Iterate arraylist with for loop

ArrayList<Integer> digits = new ArrayList<>(Arrays.asList(1,2,3,4,5,6));

for(int i = 0; i < digits.size(); i++)
{
```

```
    System.out.print(digits.get(i));
}
```

6.3. forEach loop

`forEach loop` works pretty much same to simple for loop. The only difference is that the JVM manages the counter initialization and increment. We get the next element in each iteration in the loop.

```
Iterate arraylist with forEach loop

ArrayList<Integer> digits = new ArrayList<>(Arrays.asList(1,2,3,4,5,6));

for(Integer d : digits)
{
    System.out.print(d);
}
```

7. Finding the length of the ArrayList

To get the length of the arraylist, we use the `size()` method.

```
Size of array list

ArrayList<Integer> digits = new ArrayList<>(Arrays.asList(1,2,3,4,5,6));

System.out.print( digits.size() );    // 6
```

8. Sorting an ArrayList

ArrayList `sort()` method sorts the list according to the order induced by the specified `Comparator` instance. All elements in the list must must be mutually `Comparable`.

```
AgeSorter.java

public class AgeSorter implements Comparator<Employee>
{
    @Override
    public int compare(Employee e1, Employee e2) {
        //comparison logic
    }
}
```

```
AgeSorter.java

ArrayList<Employee> employees = new ArrayList<>();

employees.add(new Employee(...));
employees.add(new Employee(...));
employees.add(new Employee(...));

employees.sort(new NameSorter());
```

9. ArrayList Methods

ArrayList add() method example
ArrayList addAll() method example
ArrayList clear() method example
ArrayList clone() – How to clone an ArrayList
ArrayList contains() method example
ArrayList ensureCapacity() method example
ArrayList forEach() method example
ArrayList get() method example
ArrayList indexOf() method example
ArrayList lastIndexOf() method example
ArrayList listIterator() method example
ArrayList remove() method example
ArrayList removeAll() method example
ArrayList retainAll() method example
ArrayList replaceAll() method example
ArrayList removeIf() method example
ArrayList sort() method example
ArrayList spliterator() method example
ArrayList subList() method example
ArrayList toArray() method example

10. Java ArrayList Examples

10.1. Create arraylist

Initialize ArrayList

Iterate through ArrayList

10.2. Add elements and remove elements

Add element at particular index of ArrayList

Remove element from ArrayList

Add multiple items to ArrayList

10.3. Sort arraylist

Sort ArrayList

Sort ArrayList of Objects using Comparable and Comparator

Sort ArrayList of objects by multiple fields

Sort ArrayList of objects using Collections.sort() method

10.4. Get/Search

Get Sub List of ArrayList

Find the index of last index of the element in the ArrayList

Get the index of the element in the ArrayList

Get element from ArrayList

Check if element exists in ArrayList

10.5. Working with ArrayList

Compare two ArrayLists

Synchronize ArrayList

Swap two elements in ArrayList

Serialize ArrayList

Join two ArrayList

Make ArrayList Empty

Check whether ArrayList is empty or not

Replace the value of existing element in ArrayList

Remove duplicate elements in ArrayList

10.6. Conversions

Convert LinkedList to ArrayList

Convert Vector to ArrayList

Convert ArrayList to String Array

Convert Array to ArrayList

Convert HashSet to ArrayList

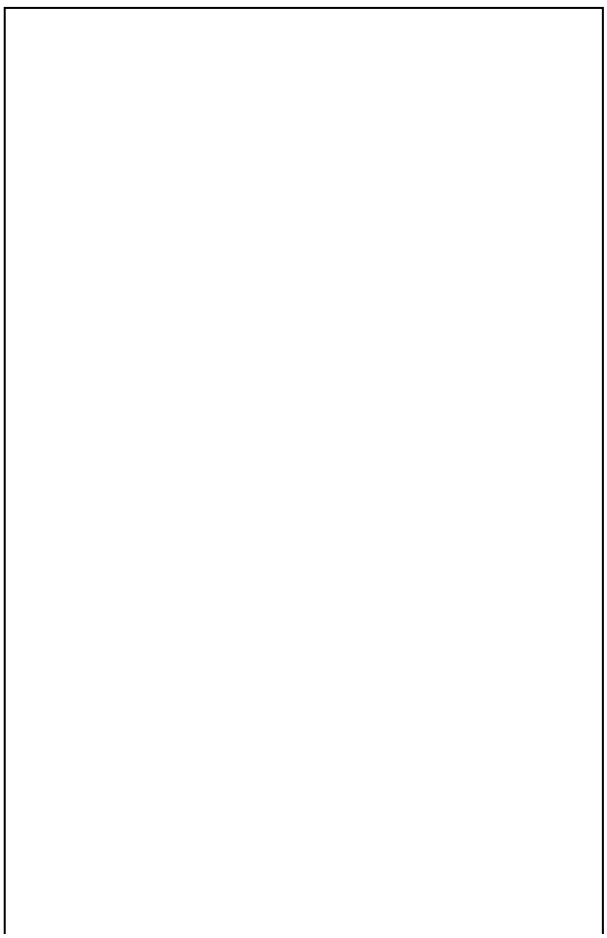
10.7. Difference between collections

ArrayList vs Vector

ArrayList vs LinkedList

Search ...


Original series
Now
Sig
© 2022
Subscri



HowToDoInJava

A blog about Java and related technologies, the best practices, algorithms, and interview questions.

Meta Links

- About Me
- Contact Us

- Privacy policy
- Advertise
- Guest Posts

Blogs

REST API Tutorial