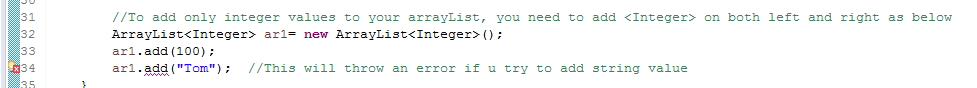
<https://www.callicoder.com/java-arraylist/> very nice one

<https://www.youtube.com/watch?v=UesFP2vXih0> Naveen video



**package** Arrays;

**import** java.util.ArrayList;

**public** **class** ArrayListConcept {

**public** **static** **void** main(String[] args) {

//To an arrayList u can add any number or any type of values

//stores values on basis of index

ArrayList ar= **new** ArrayList();

ar.add(100); //0

ar.add(200); //1

ar.add(300); //2

System.***out***.println(ar.size());

ar.add(400); //3

ar.add(500); //4

System.***out***.println(ar.size());

ar.add("Tom"); //5

ar.add("Hello"); //6

ar.add(12.33); //7

ar.add('M'); //8

System.***out***.println(ar.size());

System.***out***.println(ar.get(4));

System.***out***.println(ar.get(8));

//to print all the values of an arrayList use the for loop

**for**( **int** i=0; i<=ar.size(); i++)

System.***out***.println(ar.get(i));

//To add only integer values to your arrayList, you need to add <Integer> on both left and right as below

ArrayList<Integer> ar1 = **new** ArrayList<Integer>();

ar1.add(100);

//ar1.add("Tom"); //This will throw an error if u try to add string value

//System.out.println(ar1.get(0));

//To add only String values to your arrayList, you need to add <String> on both left and right as below

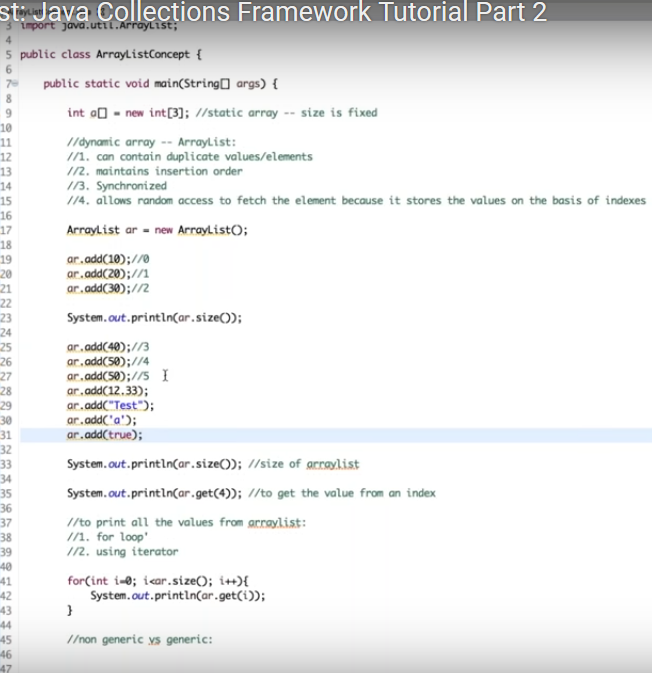
ArrayList<String> ar2= **new** ArrayList<String>();

ar2.add("Subash ");

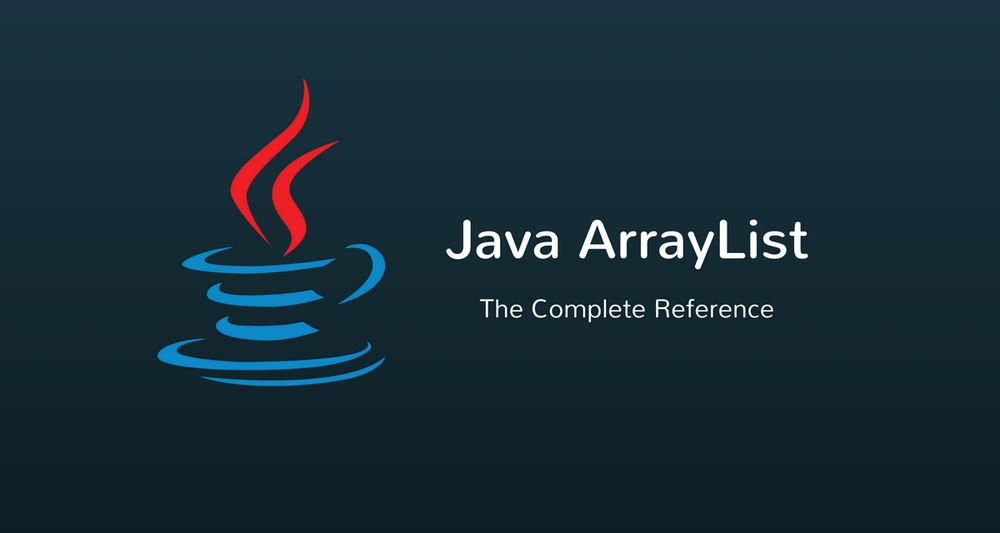
//System.out.println(ar2.get(0));

}

}



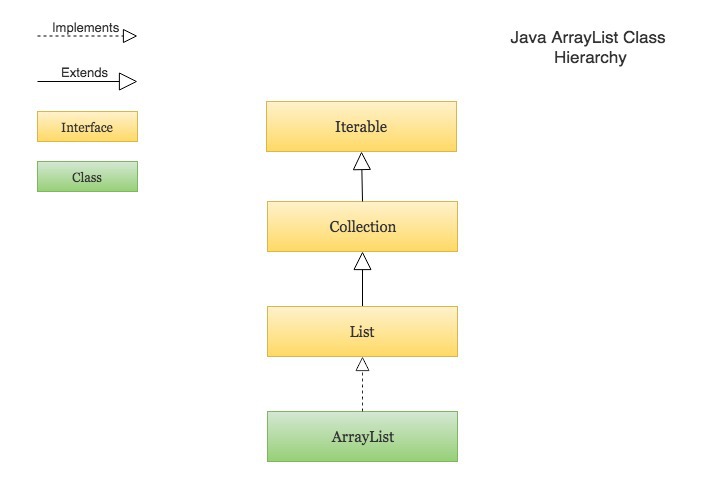
**Java ArrayList Tutorial with Examples**



//To an arrayList u can add any number of values or any types of values

ArrayList in Java is used to store dynamically sized collection of elements. Contrary to Arrays that are fixed in size, an ArrayList grows its size automatically when new elements are added to it.

ArrayList is part of Java’s collection framework and implements Java’s List interface.



Following are few key points to note about ArrayList in Java -

* An ArrayList is a re-sizable array, also called a dynamic array. It grows its size to accommodate new elements and shrinks the size when the elements are removed.
* ArrayList internally uses an array to store the elements. Just like arrays, It allows you to retrieve the elements by their index.
* Java ArrayList allows duplicate and null values.
* Java ArrayList is an ordered collection. It maintains the insertion order of the elements.
* You cannot create an ArrayList of primitive types like int, char etc. You need to use boxed types like Integer, Character, Boolean etc.
* Java ArrayList is not synchronized. If multiple threads try to modify an ArrayList at the same time, then the final outcome will be non-deterministic. You must explicitly synchronize access to an ArrayList if multiple threads are gonna modify it.
* The example at the bottom uses [Collections.synchronizedList()](https://docs.oracle.com/javase/8/docs/api/java/util/Collections.html" \l "synchronizedList-java.util.List-) method to get a synchronized view of the ArrayList.
* Moreover, the modifications to the ArrayList inside the incrementArrayList() method is wrapped inside a synchronized block. This ensures that no two threads can increment ArrayList elements at the same time.
* *You can also use a [CopyOnWriteArrayList](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CopyOnWriteArrayList.html) if you need thread safety. It is a thread-safe version of the ArrayList class. It implements all the mutating operations by making a fresh copy of the ArrayList.*

**Creating an ArrayList and adding new elements to it**

This example shows:

* How to create an ArrayList using the [ArrayList()](https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html" \l "ArrayList--) constructor.
* Add new elements to an ArrayList using the [add()](https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html#add-E-) method.

import java.util.ArrayList;

import java.util.List;

public class CreateArrayListExample {

public static void main(String[] args) {

// Creating an ArrayList of String

List<String> animals = new ArrayList<>();

// Adding new elements to the ArrayList

animals.add("Lion");

animals.add("Tiger");

animals.add("Cat");

animals.add("Dog");

System.out.println(animals);

// Adding an element at a particular index in an ArrayList

animals.add(2, "Elephant");

System.out.println(animals);

}

}

# Output

[Lion, Tiger, Cat, Dog]

[Lion, Tiger, Elephant, Cat, Dog]

**Creating an ArrayList from another collection**

This example shows:

* How to create an ArrayList from another collection using the [ArrayList(Collection c)](https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html" \l "ArrayList-java.util.Collection-) constructor.
* How to add all the elements from an existing collection to the new ArrayList using the [addAll()](https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html" \l "addAll-java.util.Collection-) method.

import java.util.ArrayList;

import java.util.List;

public class CreateArrayListFromCollectionExample {

public static void main(String[] args) {

List<Integer> firstFivePrimeNumbers = new ArrayList<>();

firstFivePrimeNumbers.add(2);

firstFivePrimeNumbers.add(3);

firstFivePrimeNumbers.add(5);

firstFivePrimeNumbers.add(7);

firstFivePrimeNumbers.add(11);

// Creating an ArrayList from another collection

List<Integer> firstTenPrimeNumbers = new ArrayList<>(firstFivePrimeNumbers);

List<Integer> nextFivePrimeNumbers = new ArrayList<>();

nextFivePrimeNumbers.add(13);

nextFivePrimeNumbers.add(17);

nextFivePrimeNumbers.add(19);

nextFivePrimeNumbers.add(23);

nextFivePrimeNumbers.add(29);

// Adding an entire collection to an ArrayList

firstTenPrimeNumbers.addAll(nextFivePrimeNumbers);

System.out.println(firstTenPrimeNumbers);

}

}

# Output

[2, 3, 5, 7, 11, 13, 17, 19, 23, 29]

**Accessing elements from an ArrayList**

This example shows:

* How to check if an ArrayList is empty using the [isEmpty()](https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html" \l "isEmpty--) method.
* How to find the size of an ArrayList using the [size()](https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html#size--) method.
* How to access the element at a particular index in an ArrayList using the [get()](https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html#get-int-) method.
* How to modify the element at a particular index in an ArrayList using the [set()](https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html#set-int-E-) method.

import java.util.ArrayList;

import java.util.List;

public class AccessElementsFromArrayListExample {

public static void main(String[] args) {

List<String> topCompanies = new ArrayList<>();

// Check if an ArrayList is empty

System.out.println("Is the topCompanies list empty? : " + topCompanies.isEmpty());

topCompanies.add("Google");

topCompanies.add("Apple");

topCompanies.add("Microsoft");

topCompanies.add("Amazon");

topCompanies.add("Facebook");

// Find the size of an ArrayList

System.out.println("Here are the top " + topCompanies.size() + " companies in the world");

System.out.println(topCompanies);

// Retrieve the element at a given index

String bestCompany = topCompanies.get(0);

String secondBestCompany = topCompanies.get(1);

String lastCompany = topCompanies.get(topCompanies.size() - 1);

System.out.println("Best Company: " + bestCompany);

System.out.println("Second Best Company: " + secondBestCompany);

System.out.println("Last Company in the list: " + lastCompany);

// Modify the element at a given index

topCompanies.set(4, "Walmart");

System.out.println("Modified top companies list: " + topCompanies);

}

}

# Output

Is the topCompanies list empty? : true

Here are the top 5 companies in the world

[Google, Apple, Microsoft, Amazon, Facebook]

Best Company: Google

Second Best Company: Apple

Last Company in the list: Facebook

Modified top companies list: [Google, Apple, Microsoft, Amazon, Walmart]

**Removing elements from an ArrayList**

This example shows:

1. How to remove the element at a given index in an ArrayList | [remove(int index)](https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html#remove-int-)
2. How to remove an element from an ArrayList | [remove(Object o)](https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html#remove-java.lang.Object-)
3. How to remove all the elements from an ArrayList that exist in a given collection | [removeAll()](https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html" \l "removeAll-java.util.Collection-)
4. How to remove all the elements matching a given predicate | [removeIf()](https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html" \l "removeIf-java.util.function.Predicate-)
5. How to clear an ArrayList | [clear()](https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html#clear--)

import java.util.ArrayList;

import java.util.List;

import java.util.function.Predicate;

public class RemoveElementsFromArrayListExample {

public static void main(String[] args) {

List<String> programmingLanguages = new ArrayList<>();

programmingLanguages.add("C");

programmingLanguages.add("C++");

programmingLanguages.add("Java");

programmingLanguages.add("Kotlin");

programmingLanguages.add("Python");

programmingLanguages.add("Perl");

programmingLanguages.add("Ruby");

System.out.println("Initial List: " + programmingLanguages);

// Remove the element at index `5`

programmingLanguages.remove(5);

System.out.println("After remove(5): " + programmingLanguages);

// Remove the first occurrence of the given element from the ArrayList

// (The remove() method returns false if the element does not exist in the ArrayList)

boolean isRemoved = programmingLanguages.remove("Kotlin");

System.out.println("After remove(\"Kotlin\"): " + programmingLanguages);

// Remove all the elements that exist in a given collection

List<String> scriptingLanguages = new ArrayList<>();

scriptingLanguages.add("Python");

scriptingLanguages.add("Ruby");

scriptingLanguages.add("Perl");

programmingLanguages.removeAll(scriptingLanguages);

System.out.println("After removeAll(scriptingLanguages): " + programmingLanguages);

// Remove all the elements that satisfy the given predicate

programmingLanguages.removeIf(new Predicate<String>() {

@Override

public boolean test(String s) {

return s.startsWith("C");

}

});

/\*

The above removeIf() call can also be written using lambda expression like this -

programmingLanguages.removeIf(s -> s.startsWith("C"))

\*/

System.out.println("After Removing all elements that start with \"C\": " + programmingLanguages);

// Remove all elements from the ArrayList

programmingLanguages.clear();

System.out.println("After clear(): " + programmingLanguages);

}

}

# Output

Initial List: [C, C++, Java, Kotlin, Python, Perl, Ruby]

After remove(5): [C, C++, Java, Kotlin, Python, Ruby]

After remove("Kotlin"): [C, C++, Java, Python, Ruby]

After removeAll(scriptingLanguages): [C, C++, Java]

After Removing all elements that start with "C": [Java]

After clear(): []

**Iterating over an ArrayList**

The following example shows how to iterate over an ArrayList using

1. Java 8 [forEach](https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html" \l "forEach-java.util.function.Consumer-) and lambda expression.
2. [iterator()](https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html#iterator--).
3. [iterator()](https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html#iterator--) and Java 8 [forEachRemaining()](https://docs.oracle.com/javase/8/docs/api/java/util/Iterator.html" \l "forEachRemaining-java.util.function.Consumer-) method.
4. [listIterator()](https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html#listIterator-int-).
5. Simple for-each loop.
6. for loop with index.

import java.util.ArrayList;

import java.util.Iterator;

import java.util.List;

import java.util.ListIterator;

public class IterateOverArrayListExample {

public static void main(String[] args) {

List<String> tvShows = new ArrayList<>();

tvShows.add("Breaking Bad");

tvShows.add("Game Of Thrones");

tvShows.add("Friends");

tvShows.add("Prison break");

System.out.println("=== Iterate using Java 8 forEach and lambda ===");

tvShows.forEach(tvShow -> {

System.out.println(tvShow);

});

System.out.println("\n=== Iterate using an iterator() ===");

Iterator<String> tvShowIterator = tvShows.iterator();

while (tvShowIterator.hasNext()) {

String tvShow = tvShowIterator.next();

System.out.println(tvShow);

}

System.out.println("\n=== Iterate using an iterator() and Java 8 forEachRemaining() method ===");

tvShowIterator = tvShows.iterator();

tvShowIterator.forEachRemaining(tvShow -> {

System.out.println(tvShow);

});

System.out.println("\n=== Iterate using a listIterator() to traverse in both directions ===");

// Here, we start from the end of the list and traverse backwards.

ListIterator<String> tvShowListIterator = tvShows.listIterator(tvShows.size());

while (tvShowListIterator.hasPrevious()) {

String tvShow = tvShowListIterator.previous();

System.out.println(tvShow);

}

System.out.println("\n=== Iterate using simple for-each loop ===");

for(String tvShow: tvShows) {

System.out.println(tvShow);

}

System.out.println("\n=== Iterate using for loop with index ===");

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

System.out.println(tvShows.get(i));

}

}

}

# Output

=== Iterate using Java 8 forEach and lambda ===

Breaking Bad

Game Of Thrones

Friends

Prison break

=== Iterate using an iterator() ===

Breaking Bad

Game Of Thrones

Friends

Prison break

=== Iterate using an iterator() and Java 8 forEachRemaining() method ===

Breaking Bad

Game Of Thrones

Friends

Prison break

=== Iterate using a listIterator() to traverse in both directions ===

Prison break

Friends

Game Of Thrones

Breaking Bad

=== Iterate using simple for-each loop ===

Breaking Bad

Game Of Thrones

Friends

Prison break

=== Iterate using for loop with index ===

Breaking Bad

Game Of Thrones

Friends

Prison break

The iterator() and listIterator() methods are useful when you need to modify the ArrayList while traversing.

Consider the following example, where we remove elements from the ArrayList using iterator.remove() method while traversing through it -

import java.util.ArrayList;

import java.util.Iterator;

import java.util.List;

public class ArrayListIteratorRemoveExample {

public static void main(String[] args) {

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

numbers.add(13);

numbers.add(18);

numbers.add(25);

numbers.add(40);

Iterator<Integer> numbersIterator = numbers.iterator();

while (numbersIterator.hasNext()) {

Integer num = numbersIterator.next();

if(num % 2 != 0) {

numbersIterator.remove();

}

}

System.out.println(numbers);

}

}

# Output

[18, 40]

**Searching for elements in an ArrayList**

The example below shows how to:

* Check if an ArrayList contains a given element | [contains()](https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html#contains-java.lang.Object-)
* Find the index of the first occurrence of an element in an ArrayList | [indexOf()](https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html" \l "indexOf-java.lang.Object-)
* Find the index of the last occurrence of an element in an ArrayList | [lastIndexOf()](https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html" \l "lastIndexOf-java.lang.Object-)

import java.util.ArrayList;

import java.util.List;

public class SearchElementsInArrayListExample {

public static void main(String[] args) {

List<String> names = new ArrayList<>();

names.add("John");

names.add("Alice");

names.add("Bob");

names.add("Steve");

names.add("John");

names.add("Steve");

names.add("Maria");

// Check if an ArrayList contains a given element

System.out.println("Does names array contain \"Bob\"? : " + names.contains("Bob"));

// Find the index of the first occurrence of an element in an ArrayList

System.out.println("indexOf \"Steve\": " + names.indexOf("Steve"));

System.out.println("indexOf \"Mark\": " + names.indexOf("Mark"));

// Find the index of the last occurrence of an element in an ArrayList

System.out.println("lastIndexOf \"John\" : " + names.lastIndexOf("John"));

System.out.println("lastIndexOf \"Bill\" : " + names.lastIndexOf("Bill"));

}

}

# Output

Does names array contain "Bob"? : true

indexOf "Steve": 3

indexOf "Mark": -1

lastIndexOf "John" : 4

lastIndexOf "Bill" : -1

**ArrayList of user defined objects**

Since ArrayList supports generics, you can create an ArrayList of **any** type. It can be of simple types like Integer, String, Double or complex types like an ArrayList of ArrayLists, or an ArrayList of HashMaps or an ArrayList of any user defined objects.

In the following example, you’ll learn how to create an ArrayList of user defined objects.

import java.util.ArrayList;

import java.util.List;

class User {

private String name;

private int age;

public User(String name, int age) {

this.name = name;

this.age = age;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public int getAge() {

return age;

}

public void setAge(int age) {

this.age = age;

}

}

public class ArrayListUserDefinedObjectExample {

public static void main(String[] args) {

List<User> users = new ArrayList<>();

users.add(new User("Rajeev", 25));

users.add(new User("John", 34));

users.add(new User("Steve", 29));

users.forEach(user -> {

System.out.println("Name : " + user.getName() + ", Age : " + user.getAge());

});

}

}

# Output

Name : Rajeev, Age : 25

Name : John, Age : 34

Name : Steve, Age : 29

**Sorting an ArrayList**

Sorting an ArrayList is a very common task that you will encounter in your programs. In this section, I’ll show you how to -

* Sort an ArrayList using [Collections.sort()](https://docs.oracle.com/javase/8/docs/api/java/util/Collections.html" \l "sort-java.util.List-) method.
* Sort an ArrayList using [ArrayList.sort()](https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html" \l "sort-java.util.Comparator-) method.
* Sort an ArrayList of user defined objects with a custom comparator.

**1. Sort an ArrayList using Collections.sort() method**

import java.util.ArrayList;

import java.util.Collections;

import java.util.List;

public class ArrayListCollectionsSortExample {

public static void main(String[] args) {

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

numbers.add(13);

numbers.add(7);

numbers.add(18);

numbers.add(5);

numbers.add(2);

System.out.println("Before : " + numbers);

// Sorting an ArrayList using Collections.sort() method

Collections.sort(numbers);

System.out.println("After : " + numbers);

}

}

# Output

Before : [13, 7, 18, 5, 2]

After : [2, 5, 7, 13, 18]

**2. Sort an ArrayList using ArrayList.sort() method**

import java.util.ArrayList;

import java.util.Comparator;

import java.util.List;

public class ArrayListSortExample {

public static void main(String[] args) {

List<String> names = new ArrayList<>();

names.add("Lisa");

names.add("Jennifer");

names.add("Mark");

names.add("David");

System.out.println("Before soft Names : "+ names);

// Sort an ArrayList using its sort() method. You must pass a Comparator to the ArrayList.sort() method.

names.sort(new Comparator<String>() {

@Override

public int compare(String name1, String name2) {

return name1.compareTo(name2);

}

});

// The above `sort()` method call can also be written simply using lambda expression

names.sort((name1, name2) -> name1.compareTo(name2));

// Following is an even more concise solution

names.sort(Comparator.naturalOrder());

System.out.println("Sorted Names : " + names);

}

}

# Output

Names : [Lisa, Jennifer, Mark, David]

Sorted Names : [David, Jennifer, Lisa, Mark]

**3. Sort an ArrayList of Objects using custom Comparator**

import java.util.ArrayList;

import java.util.Collections;

import java.util.Comparator;

import java.util.List;

class Person {

private String name;

private Integer age;

public Person(String name, Integer age) {

this.name = name;

this.age = age;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public Integer getAge() {

return age;

}

public void setAge(Integer age) {

this.age = age;

}

@Override

public String toString() {

return "{" +

"name='" + name + '\'' +

", age=" + age +

'}';

}

}

public class ArrayListObjectSortExample {

public static void main(String[] args) {

List<Person> people = new ArrayList<>();

people.add(new Person("Sachin", 47));

people.add(new Person("Chris", 34));

people.add(new Person("Rajeev", 25));

people.add(new Person("David", 31));

System.out.println("Person List : " + people);

// Sort People by their Age

people.sort((person1, person2) -> {

return person1.getAge() - person2.getAge();

});

// A more concise way of writing the above sorting function

people.sort(Comparator.comparingInt(Person::getAge));

System.out.println("Sorted Person List by Age : " + people);

// You can also sort using Collections.sort() method by passing the custom Comparator

Collections.sort(people, Comparator.comparing(Person::getName));

System.out.println("Sorted Person List by Name : " + people);

}

}

# Output

Person List : [{name='Sachin', age=47}, {name='Chris', age=34}, {name='Rajeev', age=25}, {name='David', age=31}]

Sorted Person List by Age : [{name='Rajeev', age=25}, {name='David', age=31}, {name='Chris', age=34}, {name='Sachin', age=47}]

Sorted Person List by Name : [{name='Chris', age=34}, {name='David', age=31}, {name='Rajeev', age=25}, {name='Sachin', age=47}]

**Synchronizing Access to an ArrayList**

The ArrayList class is not synchronized. If multiple threads try to modify an ArrayList at the same time then the final result becomes not-deterministic because one thread might override the changes done by another thread.

**Example Demonstrating ArrayList’s unpredictable behavior in multi-threaded environments**

The following example shows what happens when multiple threads try to modify an ArrayList at the same time.

import java.util.ArrayList;

import java.util.List;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

import java.util.concurrent.TimeUnit;

public class UnsafeArrayListExample {

public static void main(String[] args) throws InterruptedException {

List<Integer> unsafeArrayList = new ArrayList<>();

unsafeArrayList.add(1);

unsafeArrayList.add(2);

unsafeArrayList.add(3);

// Create a thread pool of size 10

ExecutorService executorService = Executors.newFixedThreadPool(10);

// Create a Runnable task that increments each element of the ArrayList by one

Runnable task = () -> {

incrementArrayList(unsafeArrayList);

};

// Submit the task to the executor service 100 times.

// All the tasks will modify the ArrayList concurrently

for(int i = 0; i < 100; i++) {

executorService.submit(task);

}

executorService.shutdown();

executorService.awaitTermination(60, TimeUnit.SECONDS);

System.out.println(unsafeArrayList);

}

// Increment all the values in the ArrayList by one

private static void incrementArrayList(List<Integer> unsafeArrayList) {

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

Integer value = unsafeArrayList.get(i);

unsafeArrayList.set(i, value + 1);

}

}

}

The final output of the above program should be equal to [101, 102, 103] because we’re incrementing the values in the ArrayList 100 times. But if you run the program, it will produce different output every time it is run -

# Output

[96, 96, 98]

Try running the above program multiple times and see how it produces different outputs. To learn more about such issues in multi-threaded programs, check out my article on [Java Concurrency Issues and Thread Synchronization](https://www.callicoder.com/java-concurrency-issues-and-thread-synchronization/).

**Example demonstrating how to synchronize concurrent modifications to an ArrayList**

All right! Now let’s see how we can synchronize access to the ArrayList in multi-threaded environments.

The following example shows the synchronized version of the previous example. Unlike the previous program, the output of this program is deterministic and will always be the same.

import java.util.ArrayList;

import java.util.Collections;

import java.util.List;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

import java.util.concurrent.TimeUnit;

public class SynchronizedArrayListExample {

public static void main(String[] args) throws InterruptedException {

List<Integer> safeArrayList = Collections.synchronizedList(new ArrayList<>());

safeArrayList.add(1);

safeArrayList.add(2);

safeArrayList.add(3);

// Create a thread pool of size 10

ExecutorService executorService = Executors.newFixedThreadPool(10);

// Create a Runnable task that increments each element of the ArrayList by one

Runnable task = () -> {

incrementArrayList(safeArrayList);

};

// Submit the task to the executor service 100 times.

// All the tasks will modify the ArrayList concurrently

for(int i = 0; i < 100; i++) {

executorService.submit(task);

}

executorService.shutdown();

executorService.awaitTermination(60, TimeUnit.SECONDS);

System.out.println(safeArrayList);

}

// Increment all the values in the ArrayList by one

private static void incrementArrayList(List<Integer> safeArrayList) {

synchronized (safeArrayList) {

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

Integer value = safeArrayList.get(i);

safeArrayList.set(i, value + 1);

}

}

}

}

# Output

[101, 102, 103]

The above example uses [Collections.synchronizedList()](https://docs.oracle.com/javase/8/docs/api/java/util/Collections.html" \l "synchronizedList-java.util.List-) method to get a synchronized view of the ArrayList.

Moreover, the modifications to the ArrayList inside the incrementArrayList() method is wrapped inside a synchronized block. This ensures that no two threads can increment ArrayList elements at the same time.

*You can also use a [CopyOnWriteArrayList](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CopyOnWriteArrayList.html) if you need thread safety. It is a thread-safe version of the ArrayList class. It implements all the mutating operations by making a fresh copy of the ArrayList.*

**Conclusion**

That’s all folks. In this article, you learned what is an ArrayList, how to create an ArrayList, how to add, modify and remove elements from an ArrayList, how to iterate over an ArrayList, how to sort an ArrayList, and how to synchronize access to an ArrayList.

Thank you for reading. See you in the next post.

<https://www.youtube.com/watch?v=Plob9dBGyEY> VIDEO

25 MIN ONWARDS