**Java Vector**

The Vector class is an implementation of the List interface that allows us to create resizable-arrays similar to the [ArrayList](https://www.programiz.com/java-programming/arraylist" \o "Java ArrayList) class.

## Java Vector vs. ArrayList

In Java, both ArrayList and Vector implements the List interface and provides the same functionalities. However, there exist some differences between them.

The Vector class synchronizes each individual operation. This means whenever we want to perform some operation on vectors, the Vector class automatically applies a lock to that operation.

It is because when one thread is accessing a vector, and at the same time another thread tries to access it, an exception called ConcurrentModificationException is generated. Hence, this continuous use of lock for each operation makes vectors less efficient.

However, in array lists, methods are not synchronized. Instead, it uses the Collections.synchronizedList() method that synchronizes the list as a whole.

**Note:** It is recommended to use ArrayList in place of Vector because vectors are not threadsafe and are less efficient.

## Creating a Vector

Here is how we can create vectors in Java.

Vector<Type> vector = new Vector<>();

Here, Type indicates the type of a linked list. For example,

// create Integer type linked list

Vector<Integer> vector= new Vector<>();

// create String type linked list

Vector<String> vector= new Vector<>();

## Methods of Vector

## Add Elements to Vector

* add(element) - adds an element to vectors
* add(index, element) - adds an element to the specified position
* addAll(vector) - adds all elements of a vector to another vector

For example,

import java.util.Vector;

class Main {

public static void main(String[] args) {

Vector<String> mammals= new Vector<>();

// Using the add() method

mammals.add("Dog");

mammals.add("Horse");

// Using index number

mammals.add(2, "Cat");

System.out.println("Vector: " + mammals);

// Using addAll()

Vector<String> animals = new Vector<>();

animals.add("Crocodile");

animals.addAll(mammals);

System.out.println("New Vector: " + animals);

}

}

**Output**

Vector: [Dog, Horse, Cat]

New Vector: [Crocodile, Dog, Horse, Cat]

## Access Vector Elements

* get(index) - returns an element specified by the index
* iterator() - returns an iterator object to sequentially access vector elements

For example,

import java.util.Iterator;

import java.util.Vector;

class Main {

public static void main(String[] args) {

Vector<String> animals= new Vector<>();

animals.add("Dog");

animals.add("Horse");

animals.add("Cat");

// Using get()

String element = animals.get(2);

System.out.println("Element at index 2: " + element);

// Using iterator()

Iterator<String> iterate = animals.iterator();

System.out.print("Vector: ");

while(iterate.hasNext()) {

System.out.print(iterate.next());

System.out.print(", ");

}

}

}

**Output**

Element at index 2: Cat

Vector: Dog, Horse, Cat,

## Remove Vector Elements

* remove(index) - removes an element from specified position
* removeAll() - removes all the elements
* clear() - removes all elemets. It is more efficient than removeAll()

For example,

import java.util.Vector;

class Main {

public static void main(String[] args) {

Vector<String> animals= new Vector<>();

animals.add("Dog");

animals.add("Horse");

animals.add("Cat");

System.out.println("Initial Vector: " + animals);

// Using remove()

String element = animals.remove(1);

System.out.println("Removed Element: " + element);

System.out.println("New Vector: " + animals);

// Using clear()

animals.clear();

System.out.println("Vector after clear(): " + animals);

}

}

**Output**

Initial Vector: [Dog, Horse, Cat]

Removed Element: Horse

New Vector: [Dog, Cat]

Vector after clear(): []

## Others Vector Methods

|  |  |
| --- | --- |
| Methods | Descriptions |
| set() | changes an element of the vector |
| size() | returns the size of the vector |
| toArray() | converts the vector into an array |
| toString() | converts the vector into a String |
| contains() | searches the vector for specified element and returns a boolean result |