Dynamic Array Java Example

An [Array](https://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html) is a container object that holds a fixed number of values of a single type. The length of an array is established when the array is created. The length is fixed after creation.

A [dynamic array](https://en.wikipedia.org/wiki/Dynamic_array#targetText=In%20computer%20science%2C%20a%20dynamic,many%20modern%20mainstream%20programming%20languages.) is a variable-size list data structure that allows elements to be added or removed. Dynamic arrays overcome a limit of static arrays, which have a fixed capacity that needs to be specified at allocation.

In this article, we shall discuss the dynamic array and their implementations in Java (Vector, ArrayList, LinkedList, CopyOnWriteArrayList)

1. Dynamic array

A simple [dynamic array](https://en.wikipedia.org/wiki/Dynamic_array#targetText=In%20computer%20science%2C%20a%20dynamic,many%20modern%20mainstream%20programming%20languages.) can be constructed by allocating an array of fixed-size, typically larger than the number of elements immediately required. The elements of the dynamic array are stored at the start of the underlying array, and remaining positions towards the end of the array are reserved or unused. Elements can be added at the end of the dynamic array by using reserved space until the space is completely consumed. The underlying fixed-size array needs to be increased in size when further elements have to be added after all the space is consumed. Typically resizing is expensive as it involves allocating a new array and copying each element from the original array (costs *O(n)* time).

A fixed-size array will suffice in scenarios where the maximum logical size is fixed. A dynamic array will be needed when the maximum logical size is unknown initially, or likely to change.

2. Features of Dynamic Array

Key features of a dynamic array are adding, deleting and resizing an element. Let us now check these features.

2.1 Add an element to a dynamic array

As discussed in the previous section, elements are added at the end of an array. A new array (typically double the original array size) is created and data is copied from original array to the new one after the allocated space is consumed.

Fig 1. Adding elements

2.2 Delete an element from a dynamic array

The remove(i) removes the element at index location – *‘i’* and shifts all the elements at the right side of the index to left.

Fig 2. Removing element

2.3 Resize an array

An array’s size can be increased or decreased. Resizing is usually an expensive operation, as it would mean creating a new array and copying all the elements (costs *O(n)* time).

3. Dynamic array Java example

Let us now look at an example with the features discussed above. DynamicArray class provides operations to add and remove items from an array.

*DynamicArray.java*

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99 | import java.util.Arrays;  public class DynamicArray{      private int array[];      // holds the current size of array      private int size;      // holds the total capacity of array      private int capacity;        // default constructor to initialize the array and values      public DynamicArray(){          array = new int[2];          size=0;          capacity=2;      }        // to add an element at the end      public void addElement(int element){          // double the capacity if all the allocated space is utilized          if (size == capacity){              ensureCapacity(2);          }          array[size] = element;          size++;      }        // to add an element at a particular index      public void addElement(int index, int element){          // double the capacity if all the allocated space is utilized          if (size == capacity){              ensureCapacity(2);          }          // shift all elements from the given index to right          for(int i=size-1;i>=index;i--){              array[i+1] = array[i];          }          // insert the element at the specified index          array[index] = element;          size++;      }        // to get an element at an index      public int getElement(int index){          return array[index];      }        // to remove an element at a particular index      public void remove(int index){          if(index>=size || index<0){              System.out.println("No element at this index");          }else{              for(int i=index;i<size-1;i++){                  array[i] = array[i+1];              }              array[size-1]=0;              size--;          }      }        /\* method to increase the capacity, if necessary, to ensure it can hold at least the      \*  number of elements specified by minimum capacity arguement      \*/      public void ensureCapacity(int minCapacity){          int temp[] = new int[capacity\*minCapacity];          for (int i=0; i < capacity; i++){              temp[i] = array[i];          }          array = temp;          capacity = capacity \* minCapacity;      }        /\*      \*  Trim the capacity of dynamic array to the current size. i.e. remove unused space      \*/      public void trimToSize(){          System.out.println("Trimming the array");          int temp[] = new int[size];          for (int i=0; i < size; i++){              temp[i] = array[i];          }          array = temp;          capacity = array.length;        }        // to get the current size      public int size(){          return size;      }        // to get the current capacity      public int capacity(){          return capacity;      }        // method to print elements in array      public void printElements(){          System.out.println("elements in array are :"+Arrays.toString(array));      }  } |

DynamicArrayTest class has instructions to add and remove elements.

*DynamicArrayTest.java*

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65 | public class DynamicArrayTest{      public static void main(String args[]){          DynamicArray array = new DynamicArray();          // adding elements at index 0 and 1          array.addElement(1);          array.addElement(2);          System.out.println("Size :"+array.size()+              " and Capacity :"+array.capacity());            array.addElement(3);          System.out.println("Size :"+array.size()+              " and Capacity :"+array.capacity());          array.printElements();            // add element at index 1          array.addElement(1,5);          System.out.println("Size :"+array.size()+              " and Capacity :"+array.capacity());          array.printElements();            // add element at index 2          array.addElement(2,6);          System.out.println("Size :"+array.size()+              " and Capacity :"+array.capacity());          array.printElements();            array.remove(2);          System.out.println("Size :"+array.size()+              " and Capacity :"+array.capacity());          array.printElements();            array.remove(2);          System.out.println("Size :"+array.size()+              " and Capacity :"+array.capacity());          array.printElements();            array.remove(1);          System.out.println("Size :"+array.size()+              " and Capacity :"+array.capacity());          array.printElements();            array.remove(2);          System.out.println("Size :"+array.size()+              " and Capacity :"+array.capacity());          array.printElements();          array.remove(1);          System.out.println("Size :"+array.size()+              " and Capacity :"+array.capacity());          array.printElements();            // Trim the array          array.trimToSize();          System.out.println("Size :"+array.size()+              " and Capacity :"+array.capacity());          array.printElements();          array.addElement(2);          System.out.println("Size :"+array.size()+              " and Capacity :"+array.capacity());          array.printElements();          array.addElement(3);          System.out.println("Size :"+array.size()+              " and Capacity :"+array.capacity());          array.printElements();      }  } |

Compile and execute the test class. The result would be as shown below. Note how the capacity increases after the allocated space is utilized. Trim operation removes all the unused space.

Fig 3. Test results

4. Built-in Dynamic arrays in Java

Java has built-in dynamic arrays. These are Vector, ArrayList, LinkedList and CopyOnWriteArrayList.

[ArrayList](https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html) is a resizable array implementation of the List interface. It implements all optional list operations and permits all elements, including null. In addition to implementing the List interface, this class provides methods to manipulate the size of the array that is used internally to store the list. Note that this implementation is not synchronized.

The [Vector](https://docs.oracle.com/javase/8/docs/api/java/util/Vector.html) class implements a growable array of objects. Like an array, it contains components that can be accessed using an integer index. The size of a Vector can grow or shrink as needed to accommodate adding or removing items after Vector has been created. Unlike the new collection implementations, Vector is synchronized. if a thread-safe implementation is not needed, it is recommended to use ArrayList in place of Vector.

The [LinkedList](https://docs.oracle.com/javase/8/docs/api/java/util/LinkedList.html) is a doubly-linked list implementation of the List and Deque interfaces. It implements all optional list operations and permits all elements (including null). Note that this implementation is not synchronized.

The [CopyOnWriteArrayList](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CopyOnWriteArrayList.html) is a thread-safe variant of ArrayList in which all mutative operations (add, set, and so on) are implemented by making a fresh copy of the underlying array. This is costly, but more efficient when traversals operations vastly outnumber mutations. All elements (including null) are permitted. The snapshot style iterator method uses a reference to the state of the array at the point that the iterator was created. This array never changes during the lifetime of the iterator, and hence interference is impossible.

|  |
| --- |
| // Java program deals with all operation of a dynamic array  // add, remove, resize memory of array is the main feature  public class DynamicArray {        // create three variable array[] is a array,      // count will deal with no of element add by you and      // size will with size of array[]      private int array[];      private int count;      private int size;      // constructor initialize value to variable        public DynamicArray()      {          array = new int[1];          count = 0;          size = 1;      }      // function add an element at the end of array        public void add(int data)      {            // check no of element is equql to size of array          if (count == size) {              growSize(); // make array size double          } // insert element at end of array          array[count] = data;          count++;      }        // function makes size double of array      public void growSize()      {            int temp[] = null;          if (count == size) {                // temp is a double size array of array              // and store array elements              temp = new int[size \* 2];              {                  for (int i = 0; i < size; i++) {                      // copy all array value into temp                      temp[i] = array[i];                  }              }          }            // double size array temp initialize          // into variable array again          array = temp;            // and make size is double also of array          size = size \* 2;      }        // function shrink size of array      // which block unnecessary remove them      public void shrinkSize()      {          int temp[] = null;          if (count > 0) {                // temp is a count size array              // and store array elements              temp = new int[count];              for (int i = 0; i < count; i++) {                    // copy all array value into temp                  temp[i] = array[i];              }                size = count;                // count size array temp initialize              // into variable array again              array = temp;          }      }      // function add an element at given index        public void addAt(int index, int data)      {          // if size is not enough make size double          if (count == size) {              growSize();          }            for (int i = count - 1; i >= index; i--) {                // shift all element right              // from given index              array[i + 1] = array[i];          }            // insert data at given index          array[index] = data;          count++;      }        // function remove last element or put      // zero at last index      public void remove()      {          if (count > 0) {              array[count - 1] = 0;              count--;          }      }        // function shift all element of right      // side from given index in left      public void removeAt(int index)      {          if (count > 0) {              for (int i = index; i < count - 1; i++) {                    // shift all element of right                  // side from given index in left                  array[i] = array[i + 1];              }              array[count - 1] = 0;              count--;          }      }        public static void main(String[] args)      {          DynamicArray da = new DynamicArray();            // add 9 elements in array          da.add(1);          da.add(2);          da.add(3);          da.add(4);          da.add(5);          da.add(6);          da.add(7);          da.add(8);          da.add(9);            // print all array elements after add 9 elements          System.out.println("Elements of array:");          for (int i = 0; i < da.size; i++) {              System.out.print(da.array[i] + " ");          }            System.out.println();            // print size of array and no of element          System.out.println("Size of array: " + da.size);          System.out.println("No of elements in array: " +                                                da.count);            // shrinkSize of array          da.shrinkSize();            // print all array elements          System.out.println("Elements of array "+                     "after shrinkSize of array:");          for (int i = 0; i < da.size; i++) {              System.out.print(da.array[i] + " ");          }          System.out.println();            // print size of array and no of element          System.out.println("Size of array: " + da.size);          System.out.println("No of elements in array: " +                                                 da.count);            // add an element at index 1          da.addAt(1, 22);            // print Elements of array after adding an          // element at index 1          System.out.println("Elements of array after" +                        " add an element at index 1:");          for (int i = 0; i < da.size; i++) {              System.out.print(da.array[i] + " ");          }            System.out.println();            // print size of array and no of element          System.out.println("Size of array: " + da.size);          System.out.println("No of elements in array: " +                                                 da.count);            // delete last element          da.remove();            // print Elements of array after delete last          // element          System.out.println("Elements of array after" +                                " delete last element:");          for (int i = 0; i < da.size; i++) {              System.out.print(da.array[i] + " ");          }            System.out.println();            // print size of array and no of element          System.out.println("Size of array: " + da.size);          System.out.println("No of elements in array: " +                                                da.count);            // delete element at index 1          da.removeAt(1);            // print Elements of array after delete          // an element index 1          System.out.println("Elements of array after"+                        " delete element at index 1:");          for (int i = 0; i < da.size; i++) {              System.out.print(da.array[i] + " ");          }          System.out.println();            // print size of array and no of element          System.out.println("Size of array: " + da.size);          System.out.println("No of elements in array: " +                                                 da.count);      }  } |

**Output:**

Elements of array:

1 2 3 4 5 6 7 8 9 0 0 0 0 0 0 0

Size of array: 16

No of elements in array: 9

Elements of array after shrinkSize of array:

1 2 3 4 5 6 7 8 9

Size of array: 9

No of elements in array: 9

Elements of array after add an element at index 1:

1 22 2 3 4 5 6 7 8 9 0 0 0 0 0 0 0 0

Size of array: 18

No of elements in array: 10

Elements of array after delete last element:

1 22 2 3 4 5 6 7 8 0 0 0 0 0 0 0 0 0

Size of array: 18

No of elements in array: 9

Elements of array after delete element at index 1:

1 2 3 4 5 6 7 8 0 0 0 0 0 0 0 0 0 0

Size of array: 18

No of elements in array: 8

|  | **Average Case** | **Worst Case** |
| --- | --- | --- |
| **space** | O(n)*O*(*n*) | O(n)*O*(*n*) |
| **lookup** | O(1)*O*(1) | O(1)*O*(1) |
| **append** | O(1)*O*(1) | O(n)*O*(*n*) |
| **insert** | O(n)*O*(*n*) | O(n)*O*(*n*) |
| **delete** | O(n)*O*(*n*) | O(n)*O*(*n*) |