

# ArrayList

## Definition

An ArrayList 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.

<https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html>

This link describes all the methods of the ArrayList, **you should often refer to the Java Documentation to see what methods are available for a given class.**

# Methods

Method name	Description
<code>add(<b>value</b>)</code>	Adds the given value to the end of the list
<code>add(<b>index</b>, <b>value</b>)</code>	Inserts the given value before the given index
<code>clear()</code>	Removes all elements
<code>contains(<b>value</b>)</code>	Returns true if the given element is in the list
<code>get(<b>index</b>)</code>	Returns the value at the given index
<code>indexOf(<b>value</b>)</code>	Returns the first index at which the given element appears in the list (or -1 if not found)
<code>lastIndexOf(<b>value</b>)</code>	Returns the last index at which the given element appears in the list (or -1 if not found)
<code>remove(<b>value</b>)</code>	Removes value at given index, sliding others back
<code>size()</code>	Returns the number of elements in the list

# Adding elements

- **Elements are added dynamically to the end of the list**

```
ArrayList<String> list = new ArrayList<String>();  
list.add("John");  
list.add("Paul");  
list.add("Denise");  
printList(list);
```

- **Elements can be added in a given position in the list!**

```
list.add(2, "Anne-Marie");  
printList(list);
```

# Removing elements

- **One element can be removed by index**

```
list.remove(1);
```

- **One element can also be removed by value**

```
list.remove("John");
```

- **All elements can be removed in one operation**

```
list.clear();
```

## Retrieving one element

- The value of one element can be obtained without removing it
- The value of the element at the specified position (index)

```
String val = list.get(1);
```

## Printing elements

- Elements are added dynamically to the end of the list

```
ArrayList<String> list = new ArrayList<String>();
```

```
System.out.println("Elements of the list are: " + list.toString());
```

- Printing process can be personalised using an iterator
  - We will talk about **iterator**, later on

# Searching for elements

- You can search the ArrayList for particular elements

```
if (list.contains("John")) {  
    System.out.println("John is in the list");  
} else {  
    System.out.println("John is not found.");  
}  
  
int index;  
if (list.contains("John")) {  
    index = list.indexOf("John");  
    System.out.println(index + " " + list.get(index));  
}
```

- `contains` tells you whether an element is in the list or not, and `indexOf` tells you at which index you can find it.

# Iteration over Collection

- When we want to process the elements from a collection ( e.g. ArrayList) we have to access each element, one by one
  - We iterate through the collection
  - An Iterator object can be attached to the ArrayList to access the elements one by one and to perform an operation on each element
    - Example of operations: *printing / changing the value / setting a value of each element*
  - **Iterator** type object is assigned to the collection
- The iterator supports `hasNext()` and `next()` methods



# Iteration over Collection

- **Code Example for ArrayList type collection**

```
ArrayList<String> fruits;  
fruits = new ArrayList<String>();  
fruits.add("Strawberry");  
fruits.add("Banana");  
//. . .  
Iterator i;  
i = fruits.iterator();  
while (i.hasNext())  
{  
    System.out.println("I like " +  
        i.next());  
}
```

Assign the iterator "i" to the  
**ArrayList "fruits"**

Defines an iterator "i"

Check if the iteration has more  
elements to parse (true/false)

Returns the next element from the  
**ArrayList "fruits"** in the iteration

# Iteration over Collection

## Main Steps

### 1. Associate an iterator to the collection

```
Iterator iteratorName = collectionName.iterator();
```

### 2. Iterate over (parse) the collection using the while loop

```
while (iteratorName.hasNext())  
{  
    //access the current element of the collection  
}
```

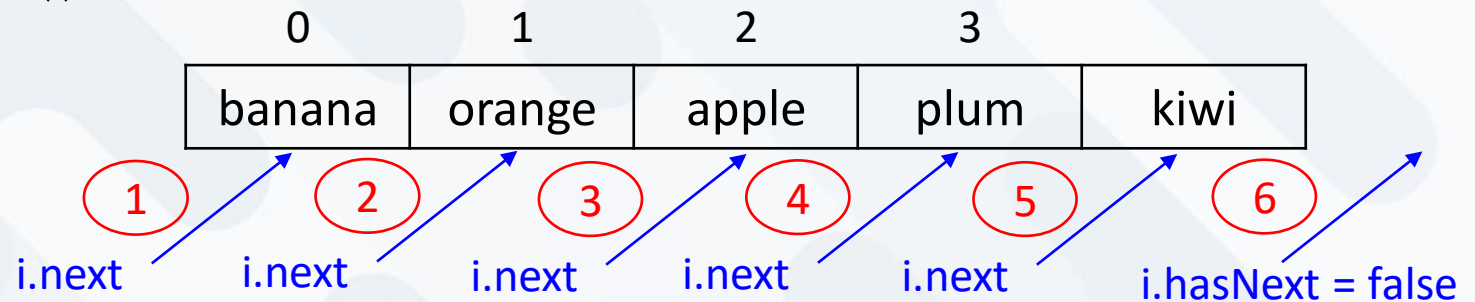
### 3. Access the current element of the collection (iterator)

- `iteratorName.next()`

Vector fruits

While loop steps

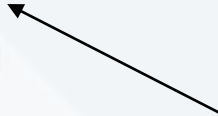
Iterator i



# Iteration over Collection

- A for loop may also be used to parse (iterate) through the collection

```
ArrayList<String> fruits;  
fruits = new ArrayList<String>();  
fruits.add("Strawberry");  
//. . .  
int j;  
for(j = 0; j<fruits.size(); j++)  
    System.out.println("Element " + fruits.get(j));
```



Returns the element on  
position j