Collection:

Collection is a [interface](https://www.geeksforgeeks.org/interfaces-in-java/) present in java.util package. It is used to represent a group of individual objects as a single unit.

The collection is considered as the root interface of the collection framework. It provides several classes and interfaces to represent a group of individual objects as a single unit.

Collections:

Collections: Collections is a utility class present in java.util package. It defines several utility methods like sorting and searching which is used to operate on collection. It has all static methods.

Framework/API

Small volume of data: variables

Large volume of data: Arrays

Arrays:

Homogenous data

Indexed based Data Structure

Single var Name

Arrays Limitations :

we cannot store Heterogenous data.

Size is fixed

Contiguous Memory Location

Vector :Legacy Classes –Does not have Uniform Data Structures.

From Java 1.2 Collection Framework were introduced in Java.

**Collection vs Collections:**

|  |  |
| --- | --- |
| **Collection** | **Collections** |
| It is an interface. | It is a utility class. |
| It is used to represent a group of individual objects as a single unit. | It defines several utility methods that are used to operate on collection. |
| The Collection is an interface that contains a static method since java8. The Interface can also contain abstract and default methods. | It contains only static methods. |

Classes:All Classes are part of util package

ArrayList(Dynmaic Array DS)

LinkedList(Doubly Linked List DS)

ArrayDeque (Double Ended Queue)

PriorityQueue

Treeset

Hashset

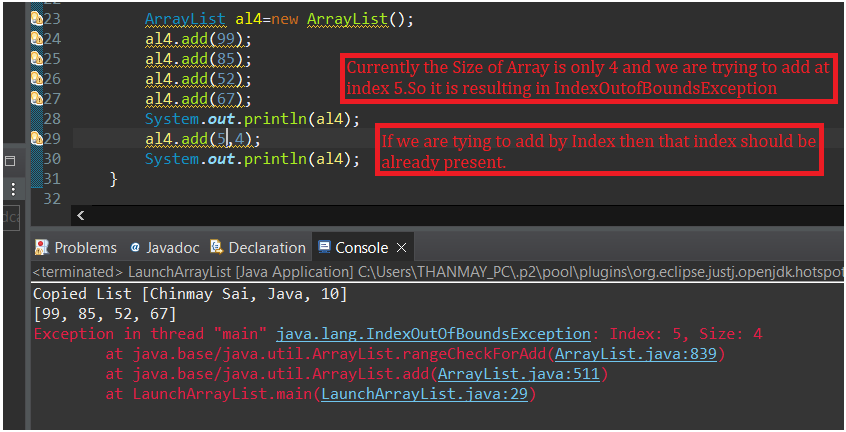
LinkedHashSet

Map: Legacy Classes

Data will be stored in collection as Object.

ArrayList --àList(Interface)

* Internally it follows Dynamic Data Structure.
* Data will be added at the rare end.
* Size will grow dynamically.
* We can store Heterogeneous data.
* Index based access is allowed
* The ArrayList's elements are stored contiguously in an underlying array.
* It allows Duplicates.



We can add the data at rear,front and middle

Rear

Front ---Not recommended

Middle --- Not recommended

When we add data in the front and middle of list shifting of elements of takes place which is time consuming.

import java.util.ArrayList;

public class LaunchArrayList {

public static void main(String[] args) {

ArrayList al=new ArrayList();

al.add(10);

al.add(20);

al.add(50);

System.***out***.println("al : "+al); // [10, 20, 50]

ArrayList al1=new ArrayList();

al1.add("Chinmay Sai");

al1.add("Java");

al1.add(10);

System.***out***.println("al1 : "+al1); //[Chinmay Sai, Java, 10]

//Copying one Array List to another

ArrayList al2=new ArrayList();

al2.addAll(al1);

System.***out***.println("Copied List "+al2);

ArrayList al4=new ArrayList();

al4.add(99);

al4.add(85);

al4.add(52);

al4.add(67);

System.***out***.println("al4 : "+al4);

al4.add(2, 28);

System.***out***.println("after adding in 2nd index :"+al4);

al4.add(0, 5);

System.***out***.println("after adding in 0th index : "+al4);

al4.add(55);

System.***out***.println("after adding in rear end : "+al4);

al4.addAll(3, al2);

System.***out***.println("New List after adding Other List : "+al4);

}

}

Array List Methods:

size() --returns how of the ArrayList is filled.

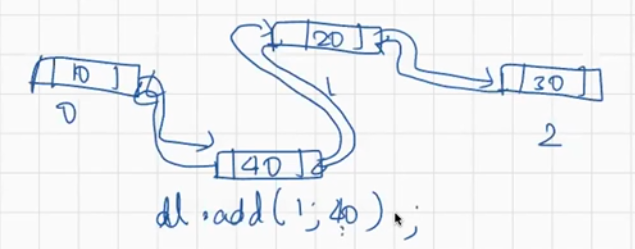
Capacity – How many elements we can have.

Linked List: (Follows Doubly Linked Data Structure)

* Homogenous Data
* Heterogeneous Data
* Stored as Object
* Indexed based access is possible.
* Linked List is more efficient when compared with Array List.
* Allows duplicates
* Implements both List Interface and Dequeue interface.

Elements will be stored at disperse locations rather than contiguous locations.

Each insertion will create a node with pointer to previous node and next node.



Ex: package org.example;  
  
import java.util.LinkedList;  
  
public class LinkedListExample {  
 public static void main(String[] args) {  
 LinkedList ll=new LinkedList();  
 ll.add(10);  
 ll.add("Chinmay Sai");  
 ll.add("Y");  
 System.*out*.println();  
 System.*out*.println("LinkedList Data "+ll);  
 ll.addLast("Last");  
 ll.addFirst("First");  
 System.*out*.println("Linked List Data "+ll);  
 ll.add(2,"Middle");  
 System.*out*.println(ll);  
 }  
}

If we use offerLast()/offerFirst() ----It is not guarnteed that the element will be added to the list.

ll.offerFirst(10);  
ll.offerLast(20);

Using the below way it is guarnteed that the addition of the elements will happen .

ll.addFirst(5);  
ll.addLast(40);

Linked List In Built Methods :

package org.example;  
  
*/\*\**  
 *\**  
 *\* Java Program for Linked List In-Built Methods*  
 *\*/*  
  
import java.util.LinkedList;  
  
public class LinkedListInBuiltMethods {  
 public static void main(String[] args) {  
 LinkedList ll=new LinkedList();  
 ll.add(15);  
 ll.add(30);  
 ll.add(45);  
 ll.add(60);  
 ll.add(75);  
 System.*out*.println("Linked List : "+ll);  
 System.*out*.println("First Element : "+ll.getFirst());  
 System.*out*.println("Last Element : "+ll.getLast());  
 System.*out*.println("Index of 75 : "+ll.indexOf(75));  
 System.*out*.println("Last Index of 15 : "+ll.lastIndexOf(15));  
  
 ll.offerFirst(10);  
 ll.offerLast(20);  
 System.*out*.println("Updated Linked List "+ll);  
 ll.addFirst(5);  
 ll.addLast(40);  
 //peekFirst() -->Retrieves, but does not remove, the first element of this list, or returns null if this list is empty.  
 System.*out*.println(ll.peekFirst());  
 System.*out*.println(ll);  
 //pollFirst() --Retrieves and removes the first element of this list, or returns null if this list is empty.  
 System.*out*.println(ll.pollFirst());  
 System.*out*.println(ll);  
 //Removes First Element of the List  
 System.*out*.println(ll.remove());  
 System.*out*.println("Final List "+ll);  
 }  
}

|  |  |  |
| --- | --- | --- |
|  | **ArrayList** | **LinkedList** |
| **1.** | This class uses a dynamic array to store the elements in it. With the introduction of [generics](https://www.geeksforgeeks.org/generics-in-java/), this class supports the storage of all types of objects. | This class uses a [doubly linked list](https://www.geeksforgeeks.org/doubly-linked-list/) to store the elements in it. Similar to the ArrayList, this class also supports the storage of all types of objects. |
| **2.** | Manipulating ArrayList takes more time due to the internal implementation. Whenever we remove an element, internally, the array is traversed and the memory bits are shifted. | Manipulating LinkedList takes less time compared to ArrayList because, in a doubly-linked list, there is no concept of shifting the memory bits. The list is traversed and the reference link is changed. |
| **3.** | Inefficient memory utilization. | Good memory utilization. |
| **4.** | It can be one, two or multi-dimensional. | It can either be single, double or circular LinkedList. |
| **5.** | Insertion operation is slow. | Insertion operation is fast. |
| **6.** | This class implements a [List interface](https://www.geeksforgeeks.org/list-interface-java-examples/). Therefore, this acts as a list. | This class implements both the List interface and the [Deque interface](https://www.geeksforgeeks.org/deque-interface-java-example/). Therefore, it can act as a list and a deque. |
| **7.** | This class works better when the application demands storing the data and accessing it. | This class works better when the application demands manipulation of the stored data. |
| **8.** | Data access and storage is very efficient as it stores the elements according to the indexes. | Data access and storage is slow in LinkedList. |
| **9.** | Deletion operation is not very efficient. | Deletion operation is very efficient. |

When to use Array and when to use ArrayList:

When the size of data is know if all the data are of same type then we can go with Arrays.

Arrays are faster when compared with ArrayList as in case of ArrayList we need to convert Primitive to Objects and then store as Object.

Array List:

If the size is List is varying

Data are not od similar type then we can go with Array List.

Array Dequeue :

* It follows Double Ended Queue
* Index based access is not allowed
* Insertion and deletion can happen at both Front end and rear end.
* Duplicates are allowed

package org.example;  
  
import java.util.ArrayDeque;  
  
public class ArrayDequeueExample {  
 public static void main(String[] args) {  
 ArrayDeque ard=new ArrayDeque();  
 ard.add(2);  
 ard.add(4);  
 ard.add(6);  
 ard.add(8);  
 ard.add(10);  
 System.*out*.println(ard);  
 ard.addFirst(1);  
 System.*out*.println(ard);  
 ard.addLast(23);  
 System.*out*.println(ard);  
 }  
}