**1)DIFFERENCE BETWEEN ARRAY LIST AND LINKED LIST:**

ArrayList and LinkedList both implements List interface and maintains insertion order. Both are non synchronized classes.

However, there are many differences between ArrayList and LinkedList classes that are given below.

**Array List**

1) ArrayList internally uses a dynamic array to store the elements.

2)Manipulation with ArrayList is slow because it internally uses an array. If any element is removed from the array,

all the bits are shifted in memory.

3)An ArrayList class can act as a list only because it implements List only.

4)ArrayList is better for storing and accessing data.

**Linked List**

1)LinkedList internally uses a doubly linked list to store the elements.

2)Manipulation with LinkedList is faster than ArrayList because it uses a doubly linked list,

so no bit shifting is required in memory.

3)LinkedList class can act as a list and queue both because it implements List and Deque interfaces.

4)LinkedList is better for manipulating data.

Example of ArrayList and LinkedList in Java

import java.util.\*;

class TestArrayLinked{

public static void main(String args[]){

List<String> al=new ArrayList<String>();//creating arraylist

al.add("Ravi");//adding object in arraylist

al.add("Vijay");

al.add("Ravi");

al.add("Ajay");

List<String> al2=new LinkedList<String>();//creating linkedlist

al2.add("James");//adding object in linkedlist

al2.add("Serena");

al2.add("Swati");

al2.add("Junaid");

System.out.println("arraylist: "+al);

System.out.println("linkedlist: "+al2);

}

}

Output:

arraylist: [Ravi,Vijay,Ravi,Ajay]

linkedlist: [James,Serena,Swati,Junaid]

**2)SET:**

* Set does not allow duplicate elements.
* Set do not maintain any insertion order.
* But in set almost only one null value.
* Set implementation classes are [HashSet](http://www.geeksforgeeks.org/hashset-in-java/), [LinkedHashSet](https://www.geeksforgeeks.org/linkedhashset-in-java-with-examples/), and [TreeSet](https://www.geeksforgeeks.org/treeset-in-java-with-examples/).
* Set does not provide get method to get the elements at a specified index
* If you want to create a collection of unique elements then we can use set
* Iterator can be used traverse the set elements

**EXAMPLE:**

import java.util.\*;

// Main class

public class CollectinosInJava2 {

// Main driver method

public static void main(String[] args)

{

// Demonstrating Set using HashSet

// Declaring object of type String

Set<String> hash\_Set = new HashSet<String>();

// Adding elements to the Set

// using add() method

hash\_Set.add("appi");

hash\_Set.add("appu");

hash\_Set.add("gopi");

hash\_Set.add("tofil");

hash\_Set.add("select");

// Printing elements of HashSet object

System.out.println(hash\_Set);

}

}

**ARRAY LIST:**

* The list interface allows duplicate elements
* The list maintains insertion order.
* We can add any number of null values.
* List implementation classes are [Array List](https://www.geeksforgeeks.org/arraylist-in-java/), [LinkedList](https://www.geeksforgeeks.org/linked-list-in-java/)
* The list provides get() method to get the element at a specified index.
* If you need to access the elements frequently by using the index then we can use the list
* To traverse the list elements by using Listlterator.

**EXAMPLE:**

import java.util.\*;

public class CollectionsInJava1

{

public static void main(String[] args)

{

List<String> l = new ArrayList<>(); //List Implementation

l.add("Sam"); //adding objects to list

l.add("Sandy");

l.add("Joe");

l.add("Arya");

l.add("Nik");

System.out.println("List objects are: " +l); // printing the list

l.remove("Nik"); //removing objects from list

System.out.println("After Removing Nik, List Objects are" +l);

}

}

**MAP:**

* The map does not allow duplicate elements
* The map also does not maintain any insertion order.
* The map allows a single null key at most and any number of null values.
* Map implementation classes are [HashMap](http://www.geeksforgeeks.org/java-util-hashmap-in-java/), [HashTable](https://www.geeksforgeeks.org/hashtable-in-java/), [TreeMap](https://www.geeksforgeeks.org/treemap-in-java/), [ConcurrentHashMap](https://www.geeksforgeeks.org/concurrenthashmap-in-java/), and [LinkedHashMap](https://www.geeksforgeeks.org/linkedhashmap-class-java-examples/).
* The map does not  provide get method to get the elements at a specified index
* If you want to store the data in the form of key/value pair then we can use the map.
* Through keyset, value, and entry set.

**EXAMPLE:**

import java.util.\*;

class Collections4 {

int id;

String name,author,publisher;

int quantity;

public Collections4(int id, String name, String author, String publisher, int quantity) {

this.id = id;

this.name = name;

this.author = author;

this.publisher = publisher;

this.quantity = quantity;

}

}

public class MapExample {

public static void main(String[] args) {

//Creating map of Books

Map<Integer,Collections4> map=new LinkedHashMap<Integer,Collections4>();

//Creating Books

Collections4 b1=new Collections4(101,"Let us C","Yashwant Kanetkar","BPB",8);

Collections4 b2=new Collections4(102,"Data Communications & Networking","Forouzan","Mc Graw Hill",4);

Collections4 b3=new Collections4(103,"Operating System","Galvin","Wiley",6);

//Adding Books to map

map.put(2,b2);

map.put(1,b1);

map.put(3,b3);

//Traversing map

for(Map.Entry<Integer, Collections4> entry:map.entrySet()){

int key=entry.getKey();

Collections4 b=entry.getValue();

System.out.println(key+" Details:");

System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity);

}

}

}

**3)equals() method**

* When comparing two objects together, Java calls their equals() method which returns true if the two objects are equal,or false otherwise. Note that this comparison using equals()method is very different than using the == operator.
* The equals()method is designed to compare two objects semantically (by comparing the data members of the class), whereas the == operator compares two objects technically (by comparing their references i.e. memory addresses).

A typical example is String comparison in Java. Let’s see the following code:

String s1 = new String("This is a string");

String s2 = new String("This is a string");

boolean refEqual = (s1 == s2);

boolean secEqual = (s1.equals(s2));

System.out.println("s1 == s2: " + refEqual);

System.out.println("s1.equals(s2): " + secEqual);

output:

s1 == s2: false

s1.equals(s2): true

hashCode()

The hashCode() method of objects is used when you insert them into a HashTable, HashMap or HashSet.

When inserting an object into a hastable you use a key. The hash code of this key is calculated,

and used to determine where to store the object internally. When you need to lookup an object in a

hashtable you also use a key. The hash code of this key is calculated and used to determine where to search

for the object.

Here are two rules that are good to know about implementing the hashCode() method in your own classes,

if the hashtables in the Java Collections API are to work correctly:

If object1 and object2 are equal according to their equals() method, they must also have the same hash code.

If object1 and object2 have the same hash code, they do NOT have to be equal too.

In shorter words:

If equal, then same hash codes too.

Same hash codes no guarantee of being equal.

Here are two example implementation of the hashCode() method matching the equals() methods shown earlier:

public class Employee {

protected long employeeId;

protected String firstName;

protected String lastName;

public int hashCode(){

return (int) employeeId;

}

}

public class Employee {

protected long employeeId;

protected String firstName;

protected String lastName;

public int hashCode(){

return (int) employeeId \*

firstName.hashCode() \*

lastName.hashCode();

}

}

**FAIL FAST DEMO:**

import java.util.HashMap;

import java.util.Iterator;

import java.util.Map;

public class FailFastDemo {

public static void main(String[] args)

{

Map<String, String> empName = new HashMap<String, String>();

empName.put("Sam Hanks", "New york");

empName.put("Will Smith", "LA");

empName.put("Scarlett", "Chicago");

Iterator iterator = empName.keySet().iterator();

while (iterator.hasNext()) {

System.out.println(empName.get(iterator.next()));

// adding an element to Map

// exception will be thrown on next call

// of next() method.

empName.put("Istanbul", "Turkey");

}

}

}

**FAIL SAFE DEMO:**

import java.util.concurrent.CopyOnWriteArrayList;

import java.util.Iterator;

class FailSafeDemo {

public static void main(String args[])

{

CopyOnWriteArrayList<Integer> list

= new CopyOnWriteArrayList<Integer>(new Integer[] { 1, 7, 9, 11 });

Iterator itr = list.iterator();

while (itr.hasNext()) {

Integer i = (Integer)itr.next();

System.out.println(i);

if (i == 7)

list.add(15); // It will not be printed

//This means it has created a separate copy of the collection

}

}

}