**Collections in Java**

The Collection in Java is a framework that provides an architecture to store and manipulate the group of objects.

Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion.

Java Collection means a single unit of objects. Java Collection framework provides many interfaces (Set, List, Queue, Deque) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet).

**What is Collection in Java**

A Collection represents a single unit of objects, i.e., a group.

**What is a framework in Java**

1. It provides readymade architecture.
2. It represents a set of classes and interfaces.
3. It is optional.

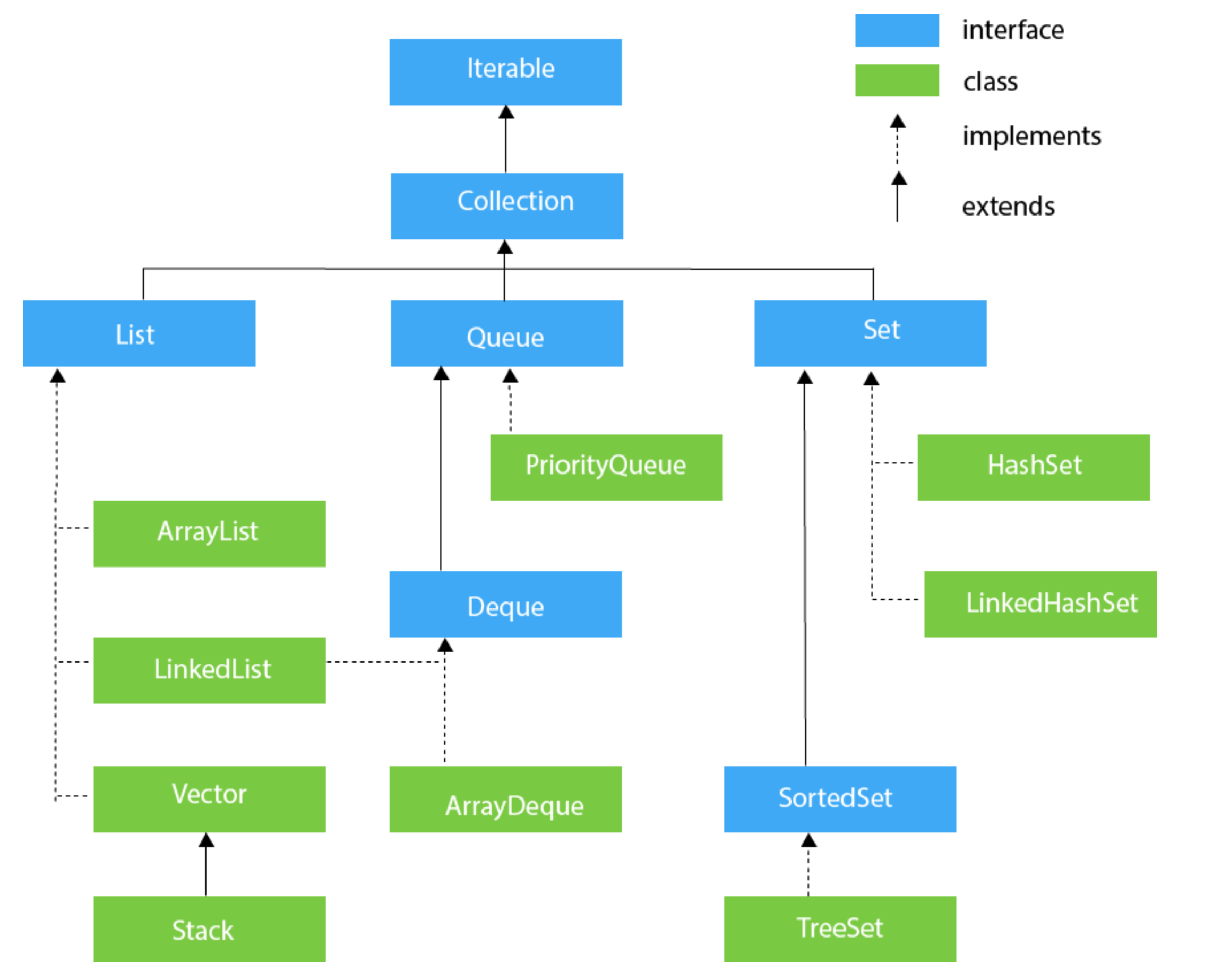
**What is Collection framework**

The Collection framework represents a unified architecture for storing and manipulating a group of objects. It has:

1. Interfaces and its implementations, i.e., classes
2. Algorithm

**Hierarchy of Collection Framework**

Let us see the hierarchy of Collection framework. The java.util package contains all the classes and interfaces for the Collection framework



**Iterable Interface**

The Iterable interface is the root interface for all the collection classes. The Collection interface extends the Iterable interface and therefore all the subclasses of Collection interface also implement the Iterable interface.

It contains only one abstract method. i.e.,

Iterator<T> iterator()

It returns the iterator over the elements of type T.

**Collection Interface**

The Collection interface is the interface which is implemented by all the classes in the collection framework. It declares the methods that every collection will have. In other words, we can say that the Collection interface builds the foundation on which the collection framework depends.

Some of the methods of Collection interface are Boolean add ( Object obj), Boolean addAll ( Collection c), void clear(), etc. which are implemented by all the subclasses of Collection interface.

**Java ArrayList**

Java ArrayList class uses a dynamic array for storing the elements. It is like an array, but there is no size limit. We can add or remove elements anytime. So, it is much more flexible than the traditional array. It is found in the java.util package. It is like the Vector in C++.

The ArrayList in Java can have the duplicate elements also. It implements the List interface so we can use all the methods of List interface here. The ArrayList maintains the insertion order internally.

**The important points about Java ArrayList class are:**

1. Java ArrayList class can contain duplicate elements.
2. Java ArrayList class maintains insertion order.
3. Java ArrayList class is non synchronized.
4. Java ArrayList allows random access because array works at the index basis.
5. In ArrayList, manipulation is little bit slower than the LinkedList in Java because a lot of shifting needs to occur if any element is removed from the array list.

**Constructors of ArrayList**

1. **ArrayList()** : It is used to build an empty array list.
2. **ArrayList(Collection<? extends E> c)** It is used to build an array list that is initialized with the elements of the collection c.
3. **ArrayList(int capacity)** It is used to build an array list that has the specified initial capacity.

**Java ArrayList Example**

import java.util.\*;

public class ArrayListExample1{

public static void main(String args[]){

ArrayList<String> list=new ArrayList<String>();//Creating arraylist

list.add("Mango");//Adding object in arraylist

list.add("Apple");

list.add("Banana");

list.add("Grapes");

//Printing the arraylist object

System.out.println(list);

}

}

**Iterating ArrayList using Iterator**

import java.util.\*;

public class ArrayListExample2{

public static void main(String args[]){

ArrayList<String> list=new ArrayList<String>();//Creating arraylist

list.add("Mango");//Adding object in arraylist

list.add("Apple");

list.add("Banana");

list.add("Grapes");

//Traversing list through Iterator

Iterator itr=list.iterator();//getting the Iterator

while(itr.hasNext()){//check if iterator has the elements

System.out.println(itr.next());//printing the element and move to next

}

}

}

**Iterating ArrayList using For-each loop**

import java.util.\*;

public class ArrayListExample3{

public static void main(String args[]){

ArrayList<String> list=new ArrayList<String>();//Creating arraylist

list.add("Mango");//Adding object in arraylist

list.add("Apple");

list.add("Banana");

list.add("Grapes");

//Traversing list through for-each loop

for(String fruit:list)

System.out.println(fruit);

}

}

**Java LinkedList class**

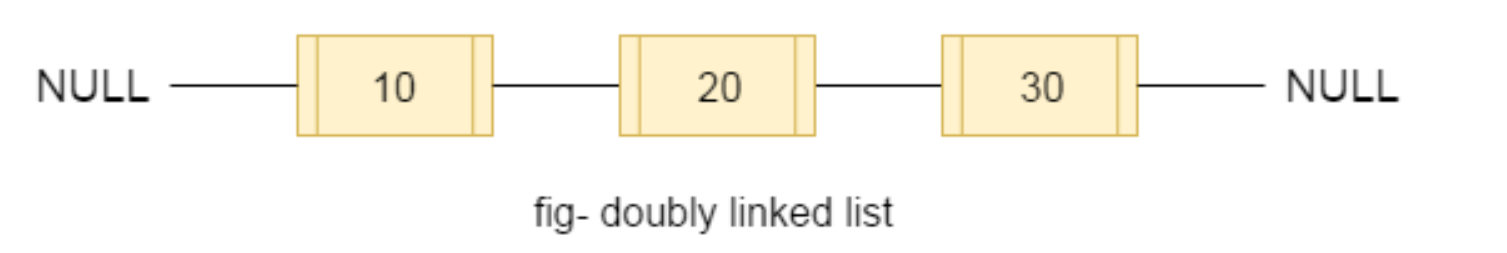
Java LinkedList class uses a doubly linked list to store the elements. It provides a linked-list data structure.

**The important points about Java LinkedList are:**

1. Java LinkedList class can contain duplicate elements.
2. Java LinkedList class maintains insertion order.
3. Java LinkedList class is non synchronized.
4. In Java LinkedList class, manipulation is fast because no shifting needs to occur.
5. Java LinkedList class can be used as a list or queue.

**Doubly Linked List**

In the case of a doubly linked list, we can add or remove elements from both sides.



**Constructors of Java LinkedList**

1. **LinkedList()** It is used to construct an empty list.
2. **LinkedList(Collection<? extends E> c)** It is used to construct a list containing the elements of the specified collection, in the order, they are returned by the collection's iterator.

**Java LinkedList Example**

import java.util.\*;

public class LinkedList1{

public static void main(String args[]){

LinkedList<String> al=new LinkedList<String>();

al.add("Ravi");

al.add("Vijay");

al.add("Ravi");

al.add("Ajay");

Iterator<String> itr=al.iterator();

while(itr.hasNext()){

System.out.println(itr.next());

}

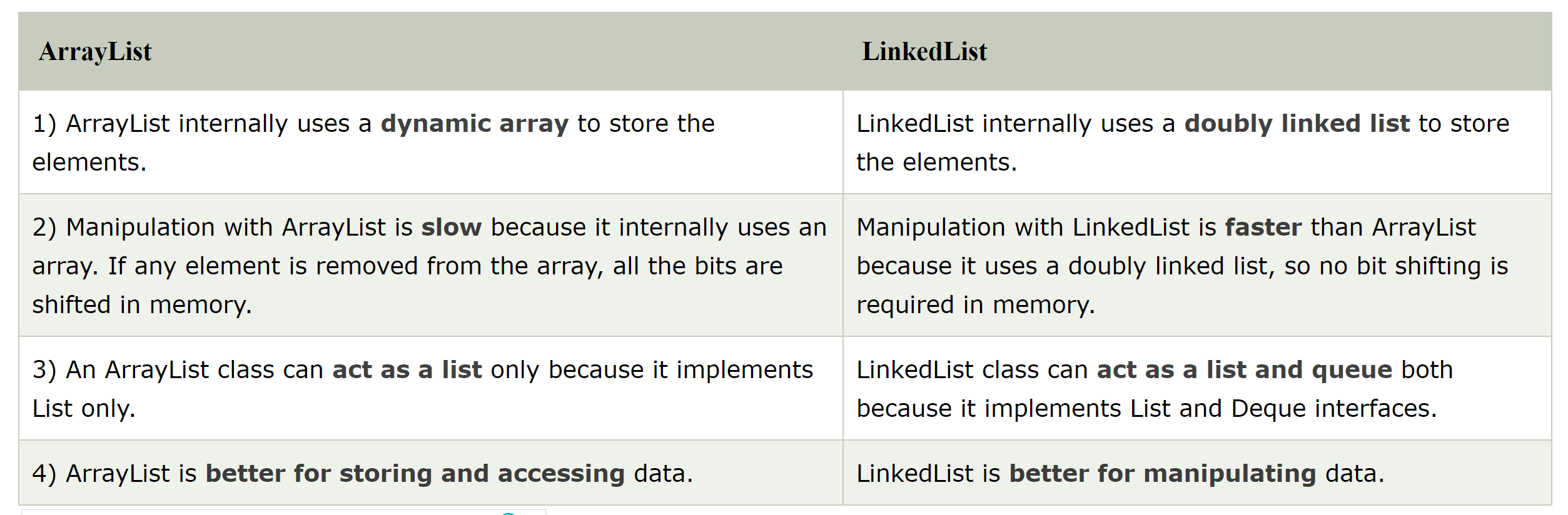
}

}

**Difference between ArrayList and LinkedList**

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.



**Vector**

Vector uses a dynamic array to store the data elements. It is similar to ArrayList. However, It is synchronized and contains many methods that are not the part of Collection framework.

import java.util.\*;

public class TestJavaCollection3{

public static void main(String args[]){

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

v.add("Ayush");

v.add("Amit");

v.add("Ashish");

v.add("Garima");

Iterator<String> itr=v.iterator();

while(itr.hasNext()){

System.out.println(itr.next());

}

}

}

**Stack**

The stack is the subclass of Vector. It implements the last-in-first-out data structure, i.e., Stack. The stack contains all of the methods of Vector class and also provides its methods like boolean push(), boolean peek(), boolean push(object o), which defines its properties.

import java.util.\*;

public class TestJavaCollection4{

public static void main(String args[]){

Stack<String> stack = new Stack<String>();

stack.push("Ayush");

stack.push("Garvit");

stack.push("Amit");

stack.push("Ashish");

stack.push("Garima");

stack.pop();

Iterator<String> itr=stack.iterator();

while(itr.hasNext()){

System.out.println(itr.next());

}

}

}

**Java HashSet**

Java HashSet class is used to create a collection that uses a hash table for storage. It implements Set interface.

The important points about Java HashSet class are:

1. HashSet stores the elements by using a mechanism called hashing.
2. HashSet contains unique elements only.
3. HashSet allows null value.
4. HashSet class is non synchronized.
5. HashSet doesn't maintain the insertion order. Here, elements are inserted on the basis of their hashcode.
6. HashSet is the best approach for search operations.

The initial default capacity of HashSet is 16, and the load factor is 0.75.

**Difference between List and Set**

A list can contain duplicate elements whereas Set contains unique elements only.

**Constructors of Java HashSet class**

**1) HashSet()** It is used to construct a default HashSet.

**2) HashSet(int capacity)** It is used to initialize the capacity of the hash set to the given integer value capacity. The capacity grows automatically as elements are added to the HashSet.

**3) HashSet(int capacity, float loadFactor)** It is used to initialize the capacity of the hash set to the given integer value capacity and the specified load factor.

**4) HashSet(Collection<? extends E> c)** It is used to initialize the hash set by using the elements of the collection c.

**Java HashSet Example**

import java.util.\*;

class HashSet1{

public static void main(String args[]){

//Creating HashSet and adding elements

HashSet<String> set=new HashSet();

set.add("One");

set.add("Two");

set.add("Three");

set.add("Four");

set.add("Five");

Iterator<String> i=set.iterator();

while(i.hasNext())

{

System.out.println(i.next());

}

}

}

**Java HashSet example ignoring duplicate elements**

import java.util.\*;

class HashSet2{

public static void main(String args[]){

//Creating HashSet and adding elements

HashSet<String> set=new HashSet<String>();

set.add("Ravi");

set.add("Vijay");

set.add("Ravi");

set.add("Ajay");

//Traversing elements

Iterator<String> itr=set.iterator();

while(itr.hasNext()){

System.out.println(itr.next());

}

}

}

**Java LinkedHashSet class**

Java LinkedHashSet class is a Hashtable and Linked list implementation of the set interface. It inherits HashSet class and implements Set interface.

The important points about Java LinkedHashSet class are:

1. Java LinkedHashSet class contains unique elements only like HashSet.
2. Java LinkedHashSet class provides all optional set operation and permits null elements.
3. Java LinkedHashSet class is non synchronized.
4. Java LinkedHashSet class maintains insertion order.

**Constructors of Java LinkedHashSet class**

**HashSet()** It is used to construct a default HashSet.

**HashSet(Collection c)** It is used to initialize the hash set by using the elements of the collection c.

**LinkedHashSet(int capacity)** It is used initialize the capacity of the linked hash set to the given integer value capacity.

**LinkedHashSet(int capacity, float fillRatio)** It is used to initialize both the capacity and the fill ratio (also called load capacity) of the hash set from its argument.

**Java LinkedHashSet Example**

import java.util.\*;

class LinkedHashSet1{

public static void main(String args[]){

//Creating HashSet and adding elements

LinkedHashSet<String> set=new LinkedHashSet();

set.add("One");

set.add("Two");

set.add("Three");

set.add("Four");

set.add("Five");

Iterator<String> i=set.iterator();

while(i.hasNext())

{

System.out.println(i.next());

}

}

}

**Java TreeSet class**

Java TreeSet class implements the Set interface that uses a tree for storage. It inherits AbstractSet class and implements the NavigableSet interface. The objects of the TreeSet class are stored in ascending order.

**The important points about Java TreeSet class are:**

1. Java TreeSet class contains unique elements only like HashSet.
2. Java TreeSet class access and retrieval times are quiet fast.
3. Java TreeSet class doesn't allow null element.
4. Java TreeSet class is non synchronized.
5. Java TreeSet class maintains ascending order.

**Constructors of Java TreeSet class**

**TreeSet()** It is used to construct an empty tree set that will be sorted in ascending order according to the natural order of the tree set.

**TreeSet(Collection<? extends E> c)** It is used to build a new tree set that contains the elements of the collection c.

**TreeSet(Comparator<? super E> comparator)** It is used to construct an empty tree set that will be sorted according to given comparator.

**TreeSet(SortedSet<E> s)** It is used to build a TreeSet that contains the elements of the given SortedSet.

**Java TreeSet Example**

import java.util.\*;

class TreeSet1{

public static void main(String args[]){

//Creating and adding elements

TreeSet<String> al=new TreeSet<String>();

al.add("Ravi");

al.add("Vijay");

al.add("Ravi");

al.add("Ajay");

//Traversing elements

Iterator<String> itr=al.iterator();

while(itr.hasNext()){

System.out.println(itr.next());

}

}

}

**Java Queue Interface**

Java Queue interface orders the element in FIFO(First In First Out) manner. In FIFO, first element is removed first and last element is removed at last.

**Methods of Java Queue Interface**

**boolean add(object)** It is used to insert the specified element into this queue and return true upon success.

**boolean offer(object)** It is used to insert the specified element into this queue.

**Object remove()** It is used to retrieves and removes the head of this queue.

**Object poll()** It is used to retrieves and removes the head of this queue, or returns null if this queue is empty.

**Object element()** It is used to retrieves, but does not remove, the head of this queue.

**Object peek()** It is used to retrieves, but does not remove, the head of this queue, or returns null if this queue is empty.

**Java PriorityQueue Example**

import java.util.\*;

class TestCollection12{

public static void main(String args[]){

PriorityQueue<String> queue=new PriorityQueue<String>();

queue.add("Amit");

queue.add("Vijay");

queue.add("Karan");

queue.add("Jai");

queue.add("Rahul");

System.out.println("head:"+queue.element());

System.out.println("head:"+queue.peek());

System.out.println("iterating the queue elements:");

Iterator itr=queue.iterator();

while(itr.hasNext()){

System.out.println(itr.next());

}

queue.remove();

queue.poll();

System.out.println("after removing two elements:");

Iterator<String> itr2=queue.iterator();

while(itr2.hasNext()){

System.out.println(itr2.next());

}

}

}

**Java Deque Interface**

Java Deque Interface is a linear collection that supports element insertion and removal at both ends. Deque is an acronym for "double ended queue".

**Methods of Java Deque Interface**

**boolean add(object)** It is used to insert the specified element into this deque and return true upon success.

**boolean offer(object)** It is used to insert the specified element into this deque.

**Object remove()** It is used to retrieves and removes the head of this deque.

**Object poll()** It is used to retrieves and removes the head of this deque, or returns null if this deque is empty.

**Object element()** It is used to retrieves, but does not remove, the head of this deque.

**Object peek()** It is used to retrieves, but does not remove, the head of this deque, or returns null if this deque is empty.

import java.util.\*;

public class ArrayDequeExample {

public static void main(String[] args) {

//Creating Deque and adding elements

Deque<String> deque = new ArrayDeque<String>();

deque.add("Ravi");

deque.add("Vijay");

deque.add("Ajay");

//Traversing elements

for (String str : deque) {

System.out.println(str);

}

}

}

**Methods to know**

**void add(int index, E element)** It is used to insert the specified element at the specified position in a list.

**boolean add(E e)** It is used to append the specified element at the end of a list.

**boolean addAll(Collection<? extends E> c)** It is used to append all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator.

**boolean addAll(int index, Collection<? extends E> c)** It is used to append all the elements in the specified collection, starting at the specified position of the list.

**E get(int index)** It is used to fetch the element from the particular position of the list.

**boolean isEmpty()** It returns true if the list is empty, otherwise false.

**boolean contains(Object o)** It returns true if the list contains the specified element