Java Collection

The java collections framework is a collection of interfaces and classes which helps in sorting and processing the data efficiently. The framework has several useful classes.

Collections are like containers that group multiple items in a single unit. Collections are used in every programming language and when java arrived, it also came with few collection classes – **Vector, stack, hashtable Array etc.**

Java Collections framework consists of the following parts:

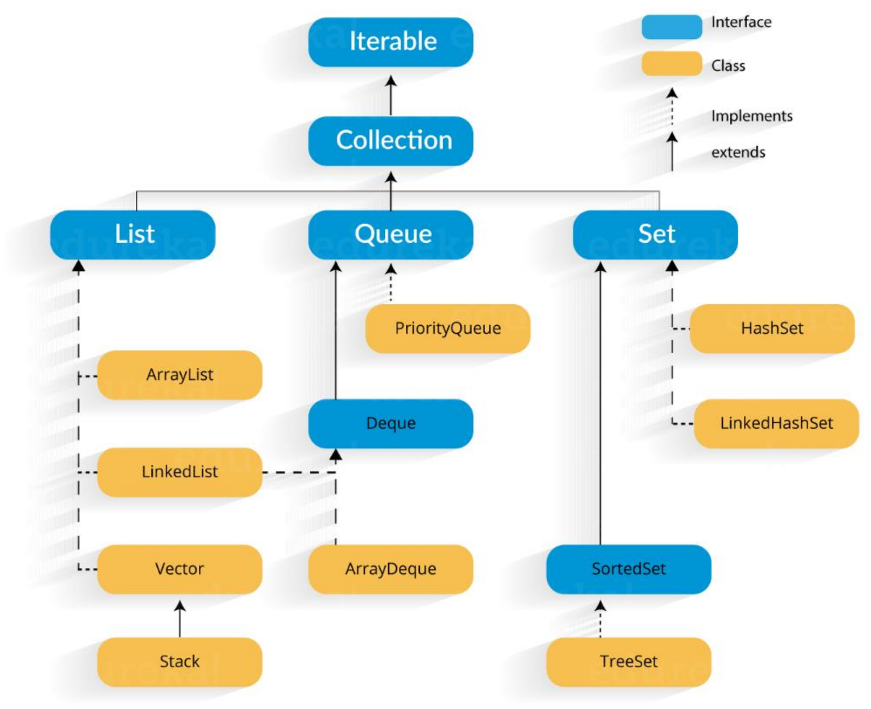
- Interface: Java collection Framework interface provides the abstract data types to represent collection. Java.util.Collection is the root interface of collections framework.
  It contain some important method such as **size(), iterator(), add(), remove(), clear()** that every collection class must implement. All the collections framework interfaces are present in **java.util** package.
- Implementation class: Java collection framework provides core implementation class for collections. We can use them to create different type of collection in java programs such as **ArrayList, LinkedList, HashMap, TreeMap, HashSet, TreeSet.**
- Algorithms: Algorithms are useful methods to provide some common functionalities for example searching, sorting and shuffling. Algorithm are polymorphic in nature as the same time method can be used to take many forms.

Why use collection in java?

There are several benefits of using java collections such as:

1. Reducing the effort required to write the code by providing useful data structure and algorithms.
2. Java Collections provides high-performance and high-quality data structure and algorithms thereby increasing the speed and quality.
3. Unrelated API's can pass collection interfaces back and forth.
4. Decrees extra efforts required to learn and design new API's
5. Support reusability of standard data structure and algorithm.

Java Collections framework hierarchy:



Collection interface is at the root of the hierarchy. Collection interface provides all general purpose methods which all collections class must support or throw UnsupportedOperationException. It extends iterable interface which add support iterating over collection elements using the for each loop statement.

Java Collection: Interface

Iterator Interface:

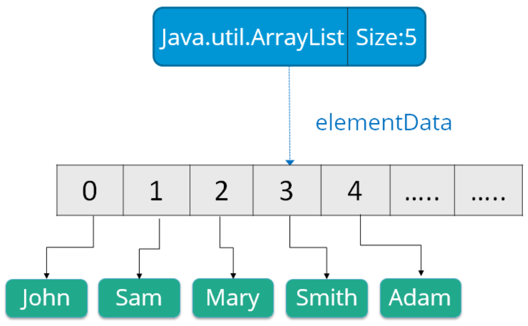
Iterator interface that iterates the elements. It is used to traverse the list and modify the elements. Iterator interface has three methods which are mentioned below:

1. public boolean hasNext(): This method returns true if the iterator has more elements.
2. public object next(): It return the element and move the cursor pointer to the next element.
3. public void remove(): This method remove the last element returned by the iterator.

Java Collection List

A list is an ordered collection of element which may contain duplicates. it is an interface that extends the collection interface. Lists are further classified into the following:

1. ArrayList
   ArrayList is the implementation of list interface where the elements can be dynamically added or removed from the list. Also, the size of the list is increase dynamically it the element are add more than the initial size



Synatax:

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

Method	Description
boolean add(Collection c)	Appends the specified element to the end of a list.
void add(int index, Object element)	Inserts the specified element at the specified position.
void clear()	Removes all the elements from this list.
int lastIndexOf(Object o)	Return the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this element.
Object clone()	Return a shallow copy of an ArrayList.
Object[] toArray()	Returns an array containing all the elements in the list.
void trimToSize()	Trims the capacity of this ArrayList instance to be the list's current size.

Example:

```
import java.util.*;
class ArrayListExample{
      public static void main(String args[]){

            ArrayList al=new ArrayList(); // creating array list
            al.add("Jack");               // adding elements 1
            al.add("Tyler");              // adding elements 2
            Iterator itr=al.iterator();
            while(itr.hasNext()){
                  System.out.println(itr.next());
            }
             }
      }
```

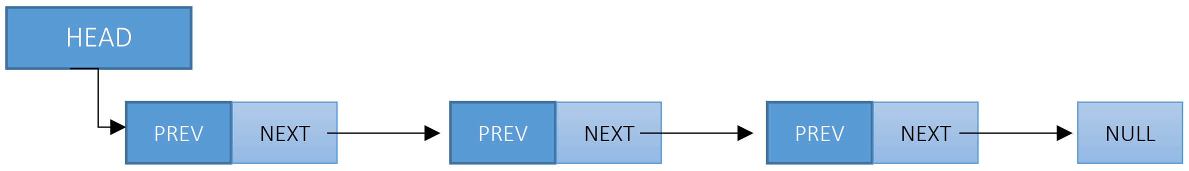
2. Linked List

Linked List is a sequence of links which contains items. Each Link contains a connection to another link.

Java Linked List class uses two of linked list to store the element.

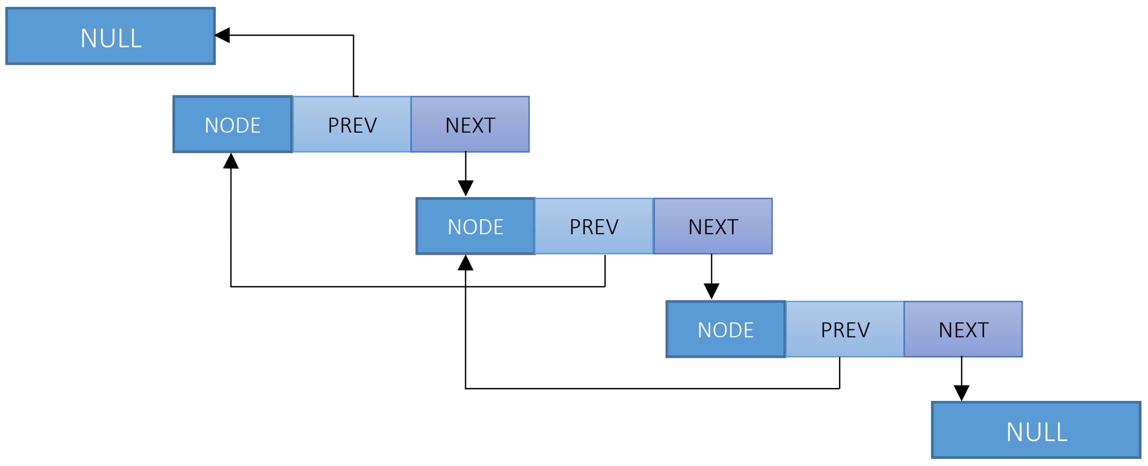
- Singly Linked List

  In a single linked list each node in this list stores the data of the node and a pointer or reference to the next node in the list.



- Doubly Linked List

  In a doubly Linked list, it has two reference, one to the next node and another to previous node.



Some of the methods in the linked list are listed below:

Method	Description
boolean add(Object o)	It is used to append the specified element to the end of the vector.
boolean contains(Object o)	Returns true if this list contains the specified element.
void add (int index, Object element)	Inserts the element at the specified element in the vector.
void addFirst(Object o)	It is used to insert the given element at the beginning.
void addLast(Object o)	It is used to append the given element to the end.
int size()	It is used to return the number of elements in a list
boolean remove(Object o)	Removes the first occurrence of the specified element from this list.
int indexOf(Object element)	Returns the index of the first occurrence of the specified element in this list, or -1.
int lastIndexOf(Object element)	Returns the index of the last occurrence of the specified element in this list, or -1.

```
import java.util.*;
public class LinkedlistExample{
        public static void main(String args[]){
                LinkedList<String> al=new LinkedList<String>();// creating linked list
                al.add("Rachit"); // adding elements
                al.add("Rahul");
                al.add("Rajat");
                Iterator<String> itr = al.iterator();
                while(itr.hasNext()){
                        System.out.println(itr.next());
                }
        }
}
```

3. Vectors
   Vectors are similar to arrays, where the elements of the vector object can be accessed via an index into the vector. Vector implements a dynamic array. Also, the vector is not limited to a specific size, it can shrink or grow automatically whenever required. It is similar to ArrayList, but with two differences:
   - Vector is synchronized.
   - Vector contains many legacy methods that are not part of the collection's framework.
   - Below are some of the methods of the Vector class:

Method	Description
boolean add(Object o)	Appends the specified element to the end of the list.
void clear()	Removes all of the elements from this list.
void add(int index, Object element)	Inserts the specified element at the specified position.
boolean remove(Object o)	Removes the first occurrence of the specified element from this list.
boolean contains(Object element)	Returns true if this list contains the specified element.
int indexOfObject (Object element)	Returns the index of the first occurrence of the specified element in the list, or -1.
int size()	Returns the number of elements in this list.
int lastIndexOf(Object o)	Return the index of the last occurrence of the specified element in the list, or -1 if the list does not contain any element.

Java Collection queue.

Queue in java a FIFO (First in First Out) approach. It orders the elements. In first out manner. In a queue, the first element is removed first and last element is removed in the end. Each basic method exists in two forms.

One throw exception if the operation fails, the other returns a special values.

	Throw Exception	Returns Special Values
Insert	Add(e)	Offer(e)
Remove	Remove()	Poll()
Examine	Element()	Peek()

Also, Priority queue implement Queue interface. The elements of the priority queue are ordered according to their natural ordering, or by a comparator provide at the queue construction time.

Below are some of the methods of Java Queue interface:

Method	Description
boolean add(object)	Inserts the specified element into the queue and returns true if it is a success.
boolean offer(object)	Inserts the specified element into this queue.
Object remove()	Retrieves and removes the head of the queue.
Object poll()	Retrieves and removes the head of the queue, or returns null if the queue is empty.

Object element()	Retrieves, but does not remove the head of the queue.
Object peek()	Retrieves, but does not remove the head of this queue, or returns null if the queue is empty.

Example:

```
import java.util.*;
class QueueExample {
    public static void main(String args[]){
        PriorityQueue<String> queue=new PriorityQueue<String>();
        // creating priority queue
        queue.add("Amit");
        // adding elements
        queue.add("Rachit");
        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 Collection Sets

A set refers to a collection that cannot contain duplicate element. It is mainly used to model the mathematical set abstraction. Set has its implementation in various classes such as **HashSet, TreeSet** and **LinkedHashSet.**

1. HashSet: Java HashSet class creates a collection that use a hashtable for storage. HashSet only contain unique elements and it inherits the Abstract Set class and implements Set interface. Also it use a mechanism hashing to store the element

   Below are some of the methods of Java HashSet class:

Method	Description
boolean add(Object o)	Adds the specified element to this set if it is not already present.
boolean contains(Object o)	Returns true if the set contains the specified element.
void clear()	Removes all the elements from the set.
boolean isEmpty()	Returns true if the set contains no elements.
boolean remove(Object o)	Remove the specified element from the set.
Object clone()	Returns a shallow copy of the HashSet instance: the elements themselves are not cloned.
Iterator iterator()	Returns an iterator over the elements in this set.
int size()	Return the number of elements in the set.

```
import java.util.*;
      class HashsetExample{
            public static void main(String args[]){

            HashSet<String> al= new HashSet(); // creating hashSet
            al.add("Rachit");                 // adding elements
            al.add("Amit");
            al.add("jack");
            Iterator<String>   itr=al.iterator();
```

```
                while(itr.hasNext()){
                        System.out.println(itr.next());
                }
        }
 }
```

2. Linked HashSet: Java LinkedHashSet class is a Hash table and Linked list implementation of the set interface. It contains only unique elements like HashSet. Linked HashSet also provides all optional set operations and maintains insertion order.

   Example:

```
        import java.util.*;
class LinkedHashsetExample{
        public static void main(String args[]){
                LinkedHashSet<String> al=new LinkedHashSet<String>();
                al.add("Mariana");
                al.add("Rick");
                al.add("Sam");
                Iterator<String> itr=al.iterator();
                while(itr.hasNext()){
                        System.out.println(itr.next());
                }
        }
}
```

3. TreeSet: TreeSet class implement the set interface that uses a tree for storage. The object of this class is stored in the ascending order. also, it inherits abstractSet class and implements NavigableSet interface. It contains only unique element like HashSet. In TreeSet class, access and retrieval time are faster.

   Below are some method of java TreeSet Class:

Method	Description
boolean addAll(Collection c)	Add all the elements in the specified collection to this set.
boolean contains(Object o)	Returns true if the set contains the specified element.
boolean isEmpty()	Returns true if this set contains no elements.
boolean remove(Object o)	Remove the specified element from the set.
void add(Object o)	Add the specified element to the set.
void clear()	Removes all the elements from the set.
Object clone()	Return a shallow copy of this TreeSet instance.
Object first()	Return the first element currently in the sorted set.
Object last()	Return the last element currently in the sorted set.
int size()	Return the number of elements in the set.

```
        import java.util.*;
        class TreeSetExample{
                public static void main(String args[]){
                        TreeSet<String> al=new TreeSet<String>();
                        al.add("John");
                        al.add("Sam");
                        al.add("Rick");
                        Iterator<String> itr=al.iterator();
                        while(itr.hasNext()){
                                System.out.println(itr.next());
                        }
                }
        }
```

Differences Between Collections and Collections

Collection	Collections
java.util.Collection is an interface	java.util.Collections is a class
Is used to represent a group of object as single entity	It is used to define various utility method for collection objects.
It is the root interface of the collection framework	It is a utility class
It is used to derive the data structure of the collection framework.	It contain various static method which help in data structure manipulation.

Differences Between Array and ArrayList

Array	ArrayList
java.util.Array is a class	java.util.ArrayList is a class
It is strongly typed	It is loosely types
Cannot be dynamically resized	Can be dynamically resize
No need to box and unbox the element	Needs to box and unbox element

Differences Between Iterable and Iterator

Iterable	Iterator
Iterable is an interface	It is also a interface
Belongs to java.lang.package	Belong to java.util.package
Provide one single abstract method called iterator()	Provide two abstract method called hasNext() and next()
It is represent of a sears of element that can be traversed	It represent the object with iteration state.

Differences Between ArrayList and LinkedList

ArrayList	LinkedList
Implements dynamic array internally to store elements	Implements doubly linked list internally to store element
Manipulation of element is slower	Manipulation of element is faster.
Can act like a list	Can act as a list and queue
Effective for data storage and access	Effective for data manipulation.

Differences Between Comparable and comparator

Comparable	Comparator
java.lang.package	java.util.package
Element are sorted based on natural ordering	Element are sorted based on user-customized ordering
Provide a single method called compareTo()	Provide to method equals() and compare()
Modifies the actual class	Doesn't modifies the actual class.

Differences Between List and Set

List	Set
An Ordered Collection of element	An unordered collection of element
Preserves the insertion order	Doesn't preserves the insert order
Duplicate values are allowed	Duplicate values are not allowed
Any number of null values can be stored	Only one null values can be stored
List Iterator can be used to traverse the list in any direction	ListIterator cannot be used to traverse a Set
Contain a legacy class called vector	Doesn't contains anu legacy class.