VIPS
Technical Campus
योग: कर्मसु कौशलम्
IN PURSUIT OF PERFECTION

SCHOOL OF ENGINEERING AND TECHNOLOGY

Course : OBJECT ORIENTED PROGRAMMING

Paper Code: AIDS-202,AIML-202,IIOT-202

Faculty : Dr. Shivanka

Assistant Professor

VIPS

What are Java collections?

- Java collections refer to a collection of individual objects that are represented as a single unit. You can perform all operations such as searching, sorting, insertion, manipulation, deletion, etc., on Java collections just like you do it on data.
- Java collection it is a single entity object which can store multiple data.

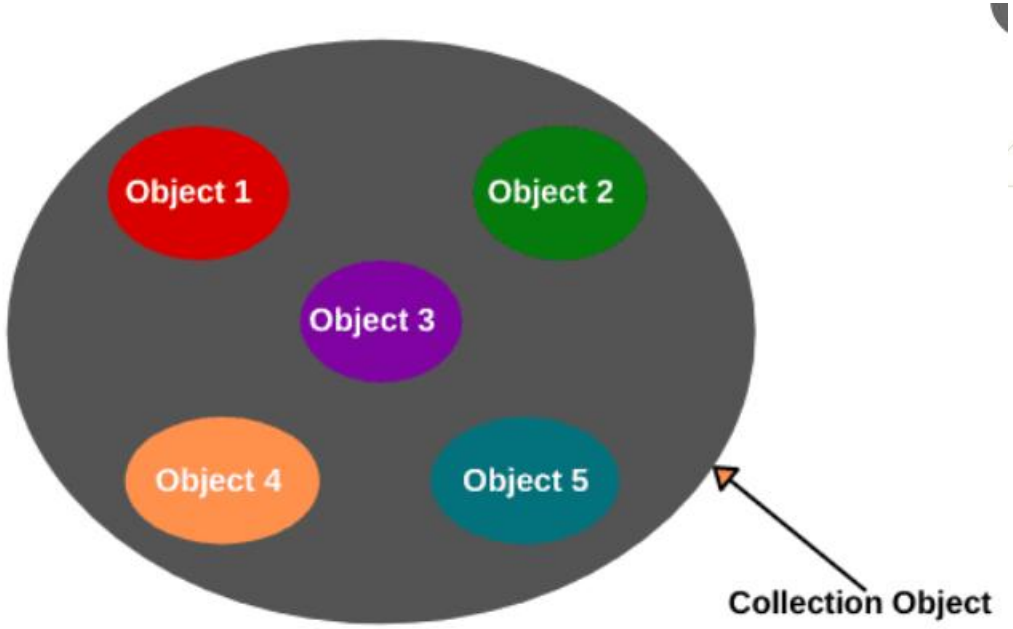


Fig: A group of objects stored in a collection object

Collections In Java and How to Implement Them?

- Java Collections are the one-stop solutions for all the data manipulation jobs such as storing data, searching, sorting, insertion, deletion, and updating of data. Java collection responds as a single object, and a Java Collection Framework provides various Interfaces and Classes.

What is a Java Collection?

- A Java Collection is a predefined architecture capable of storing a group of elements and behaving like a single unit such as an object or a group.
- And now that you're aware of what exactly is Java Collections, the next step is understanding the term Java Collections Framework.

What is the Java Collection Framework?

- Java Collection Framework repesents the library .It is set of predefined classes and interfaces which is used to store multiple data.
- It contains two parts
- i)java.util.Collection;
- ii)java.util.Map; (map contains Key value Pair for exp: roll no and names)

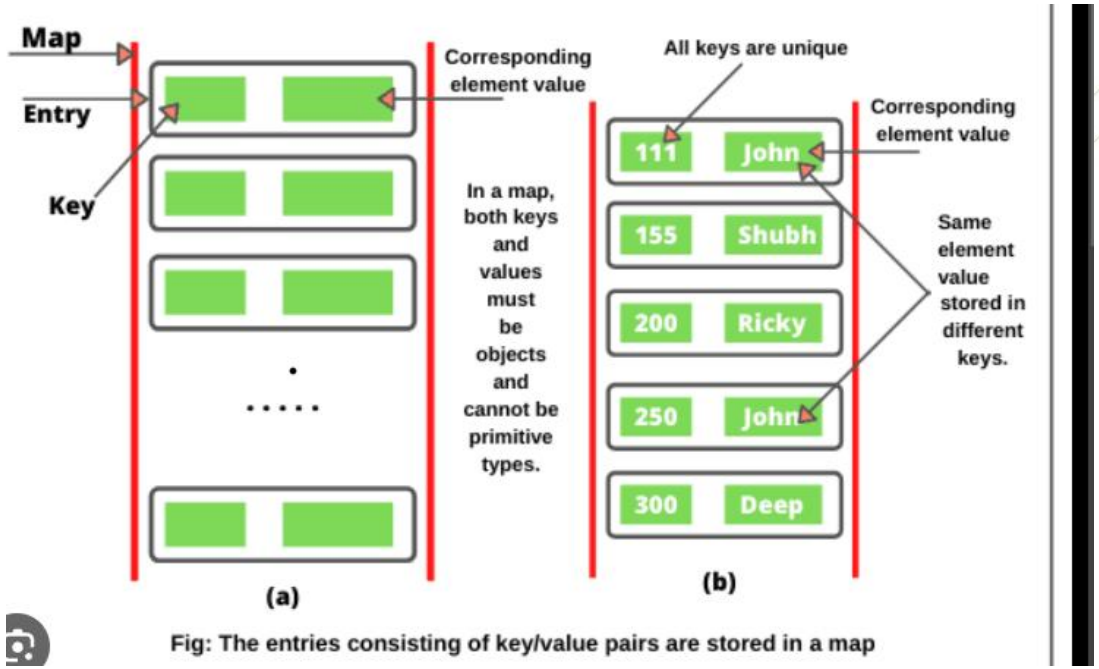


Fig: The entries consisting of key/value pairs are stored in a map

Java Collection Framework

- Java Collection Framework offers the capability to Java Collection to represent a group of elements in classes and Interfaces.
- Java Collection Framework enables the user to perform various data manipulation operations like storing data, searching, sorting, insertion, deletion, and updating of data on the group of elements. Followed by the Java Collections Framework,Hierarchy of Java collections and various descendants or classes and interfaces involved in the Java Collections.

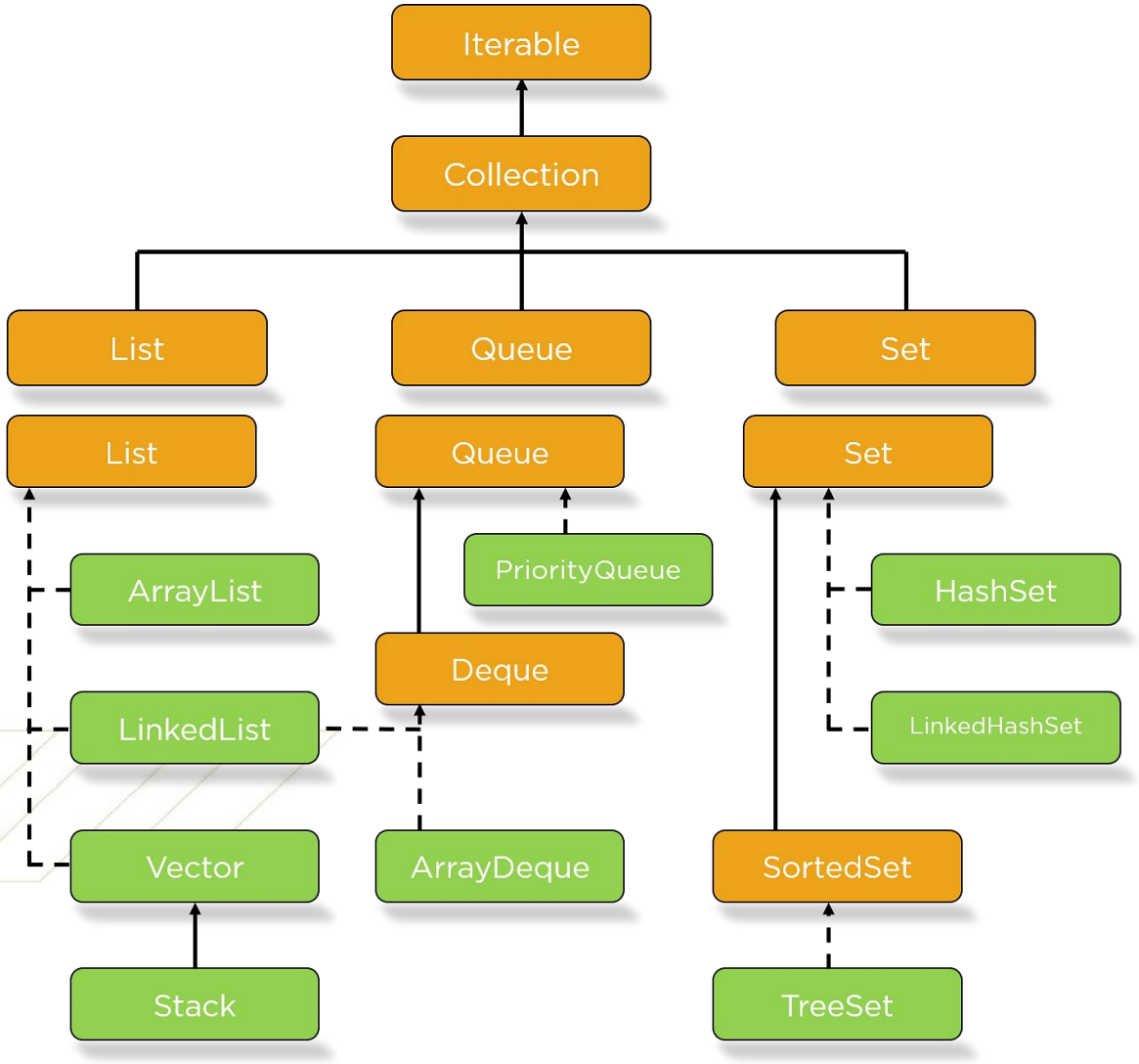
What is the Collection framework in Java?

- Collection Framework is a combination of classes and interface, which is used to store and manipulate the data in the form of objects. It provides various classes such as ArrayList, Vector, Stack, and HashSet, etc. and interfaces such as List, Queue, Set, etc. for this purpose.

QExplain various interfaces used in Collection framework?

- Collection framework implements various interfaces, Collection interface and Map interface (java.util.Map) are the mainly used interfaces of Java Collection Framework. List of interfaces of Collection Framework is given below:

Java Collection Framework Hierarchy



The following image depicts the Java Collections Hierarchy.

After the Hierarchy of Java collections; you should also get to know the various methods applied to the Collections in Java to perform the data manipulation operations.

Various interfaces used in Collection framework

1. Collection interface: Collection (java.util.Collection) is the primary interface, and every collection must implement this interface.

Syntax:public interface Collection<E>extends Iterable

Where <E> represents that this interface is of Generic type

2. List interface: List interface extends the Collection interface, and it is an ordered collection of objects. It contains duplicate elements. It also allows random access of elements.

Syntax:public interface List<E> extends Collection<E>

3. Set interface: Set (java.util.Set) interface is a collection which cannot contain duplicate elements. It can only include inherited methods of Collection interface

Syntax: public interface Set<E> extends Collection<E>

Queue interface: Queue (java.util.Queue) interface defines queue data structure, which stores the elements in the form FIFO (first in first out).

Syntax:public interface Queue<E> extends Collection<E>

Various interfaces used in Collection framework

- **4. Dequeue interface:** it is a double-ended-queue. It allows the insertion and removal of elements from both ends. It implants the properties of both Stack and queue so it can perform LIFO (Last in first out) stack and FIFO (first in first out) queue, operations.Syntax:
- Syntax
- **public interface Dequeue<E> extends Queue<E>**

- **5. Map interface:** A Map (java.util.Map) represents a **key, value pair** storage of elements. Map interface does not implement the Collection interface. It can only contain a unique key but can have duplicate elements. There are two interfaces which implement Map in java that are Map interface and Sorted Map.

What is the difference between ArrayList and Vector?

ArrayList	Vector
ArrayList is not synchronized.	Vector is synchronized.
ArrayList is not a legacy class.	Vector is a legacy class.
ArrayList increases its size by 50% of the array size.	Vector increases its size by doubling the array size.
ArrayList is not ,thread-safe, as it is not synchronized.	Vector list is ,thread-safe, as it,s every method is synchronized.

What is the difference between ArrayList and LinkedList?

ArrayList	LinkedList
ArrayList uses a dynamic array.	LinkedList uses a doubly linked list.
ArrayList is not efficient for manipulation because too much is required.	LinkedList is efficient for manipulation.
ArrayList is better to store and fetch data.	LinkedList is better to manipulate data.
ArrayList provides random access.	LinkedList does not provide random access.
ArrayList takes less memory overhead as it stores only object	LinkedList takes more memory overhead, as it stores the object as well as the address of that object.

What is the difference between Iterator and ListIterator?

Iterator	ListIterator
The Iterator traverses the elements in the forward direction only.	ListIterator traverses the elements in backward and forward directions both.
2)The Iterator can be used in List, Set, and Queue.	ListIterator can be used in List only.
3)The Iterator can only perform remove operation while traversing the collection.	ListIterator can perform add,removeand set operation while traversing the collection.

Java Iterator

- Java Iterator

An Iterator is an object that can be used to loop through collections, like ArrayList and HashSet. It is called an "iterator" because "iterating" is the technical term for looping.

- To use an Iterator, you must import it from the java.util package.

- Getting an Iterator
- The iterator() method can be used to get an Iterator for any collection:

Java Iterator and Advantage

How to use Java Iterator?

- When a user needs to use the Java Iterator, then it's compulsory for them to make an instance of the Iterator interface from the collection of objects they desire to traverse over. After that, the received Iterator maintains the trail of the components in the underlying collection to make sure that the user will traverse over each of the elements of the collection of objects.

- If the user modifies the underlying collection while traversing over an Iterator leading to that collection, then the Iterator will typically acknowledge it and will throw an exception in the next time when the user will attempt to get the next component from the Iterator.

Advantages of Java Iterator

- Iterator in Java became very prevalent due to its numerous advantages. The advantages of Java Iterator are given as follows -
- The user can apply these iterators to any of the classes of the Collection framework.
- In Java Iterator, we can use both of the read and remove operations.
- If a user is working with a for loop, they cannot modernize(add/remove) the Collection, whereas, if they use the Java Iterator, they can simply update the Collection.

Disadvantages of Java Iterator

Despite the numerous advantages, the Java Iterator has various disadvantages also. The disadvantages of the Java Iterator are given below -

- The Java Iterator only preserves the iteration in the forward direction. In simple words, the Java Iterator is a uni-directional Iterator.
- The replacement and extension of a new component are not approved by the Java Iterator.
- In CRUD Operations, the Java Iterator does not hold the various operations like CREATE and UPDATE.
- In comparison with the Spliterator, Java Iterator does not support traversing elements in the parallel pattern which implies that Java Iterator supports only Sequential iteration.
- In comparison with the Spliterator, Java Iterator does not support more reliable execution to traverse the bulk volume of data.

- Java Iterator Methods
- The following figure perfectly displays the class diagram of the Java Iterator interface. It contains a total of four methods that are:

- hasNext()
- next()
- remove()
- forEachRemaining()

Java Iterator Methods

- The forEachRemaining() method was added in the Java 8. Let's discuss each method in detail.

- boolean hasNext(): The method does not accept any parameter. It returns true if there are more elements left in the iteration. If there are no more elements left, then it will return false.

- **E next():** It is similar to hasNext() method. It also does not accept any parameter. It returns E, i.e., the next element in the traversal. If the iteration or collection of objects has no more elements left to iterate, then it throws the NoSuchElementException.
- **default void remove():** This method also does not require any parameters. There is no return type of this method. The main function of this method is to remove the last element returned by the iterator traversing through the underlying collection.

Java Iterator Methods

- **default void forEachRemaining(Consumer action):** It is the only method of Java Iterator that takes a parameter. It accepts action as a parameter. Action is nothing but that is to be performed. There is no return type of the method. This method performs the particularized operation on all of the left components of the collection until all the components are consumed or the action throws an exception. Exceptions thrown by action are delivered to the caller. If the action is null, then it throws a NullPointerException.

Iterator Example


```
import java.io.*;
import java.util.*;
    public class JavaIteratorExample {
    public static void main(String[] args)
    {
      ArrayList<String> cityNames = new ArrayList<String>();
     cityNames.add("Delhi");
       cityNames.add("Mumbai");
       cityNames.add("Kolkata");
       cityNames.add("Chandigarh");
       cityNames.add("Noida");
     // Iterator to iterate the cityNames
       Iterator iterator = cityNames.iterator();
System.out.println("CityNames elements : ");

        while (iterator.hasNext())
            System.out.print(iterator.next() + " ");
   System.out.println();
    }
}
```

Output:

CityNames elements:

Delhi Mumbai Kolkata Chandigarh Noida

What is the difference between Iterator and Enumeration?

	Iterator	Enumeration
1)	The Iterator can traverse legacy and non-legacy elements.	Enumeration can traverse only legacy elements.
2)	The Iterator is fail-fast.	Enumeration is not fail-fast.
3)	The Iterator is slower than Enumeration.	Enumeration is faster than Iterator.
4)	The Iterator can perform remove operation while traversing the collection.	The Enumeration can perform only traverse operation on the collection.

What is the difference between List and Set?

8) What is the difference between List and Set?

The List and Set both extend the collection interface. However, there are some differences between the both which are listed below.

The **List** can contain duplicate elements whereas **Set** includes unique items.
The **List** is an ordered collection which maintains the insertion order whereas **Set** is an unordered collection which does not preserve the insertion order.
The **List** interface contains a single legacy class which is Vector class whereas **Set** interface does not have any legacy class.
The **List** interface can allow n number of null values whereas **Set** interface only allows a single null value.

Difference between List and Set

List(Interface)

1.List is index based data structure .

2.List can store Duplicate element.

3.List can store any numbers of null values(because it allow duplicate value).

4.List follows the insertion orders.

5.We can iterate(get) the List element by Iterator(Forward direction)&ListIterator(Forward &backward element can get).

Set (Interface)

1.It is not Indexed based data structure.It store the data accordingly to the hashcode values.

2.Set does not allow to store duplicate element.

3.Set can store only one null values(because it don't allow duplicate value).

4.Set does not follows the insertion order.

5.We can iterate(get) the List element by Iterator(Forward direction)

Example of ArrayList(Interface)


```
import java.util.ArrayList;
import java.util.List;
public class Listdemo {
    public static void main(String[] args){
        List l=new ArrayList();
        l.add(20);//add(index,value)by default it take 0
index value
        l.add(1,30);
        l.add(2,40);
I.add(3,60);
//l.add(5,60);//error
        System.out.println("List of element");
              System.out.println(l); }
```

Output

List of element

[20, 30, 40]

Example of Arraylist and Hashset


```
package demo;
 import java.util.*;
public class arraylist {
  public static void main(String args[]){
ArrayList a1=new ArrayList();
a1.add(20);//arraylist obj contain different type of data
a1.add("parul");
a1.add('w');
System.out.println(a1);
List l=new ArrayList();//reference obj of Interface List
HashSet h1=new HashSet();
Set s1=new HashSet();//reference obj of Interface Hashset
h1.add("Arjun");//Hashset obj contain different type of data
h1.add(30);
h1.add(true);
System.out.println(h1);
}}
```

output:

[20, parul, w]

[Arjun, 30, true]

How to remove duplicates from ArrayList in Java?


```
public class RemoveDuplicateArrayList {
    public static void main(String[] args) {
        List<String> l = new ArrayList<String>();
        l.add("Mango");
        l.add("Banana");
        l.add("Mango");
        l.add("Apple");
        System.out.println(l.toString());
        Set<String> s = new LinkedHashSet<String>(l);
        System.out.println(s);
    }
}
```

Output:

Before converting to set

[Mango, Banana, Mango, Apple]

After converting to set

[Mango, Banana, Apple]

Set Interface

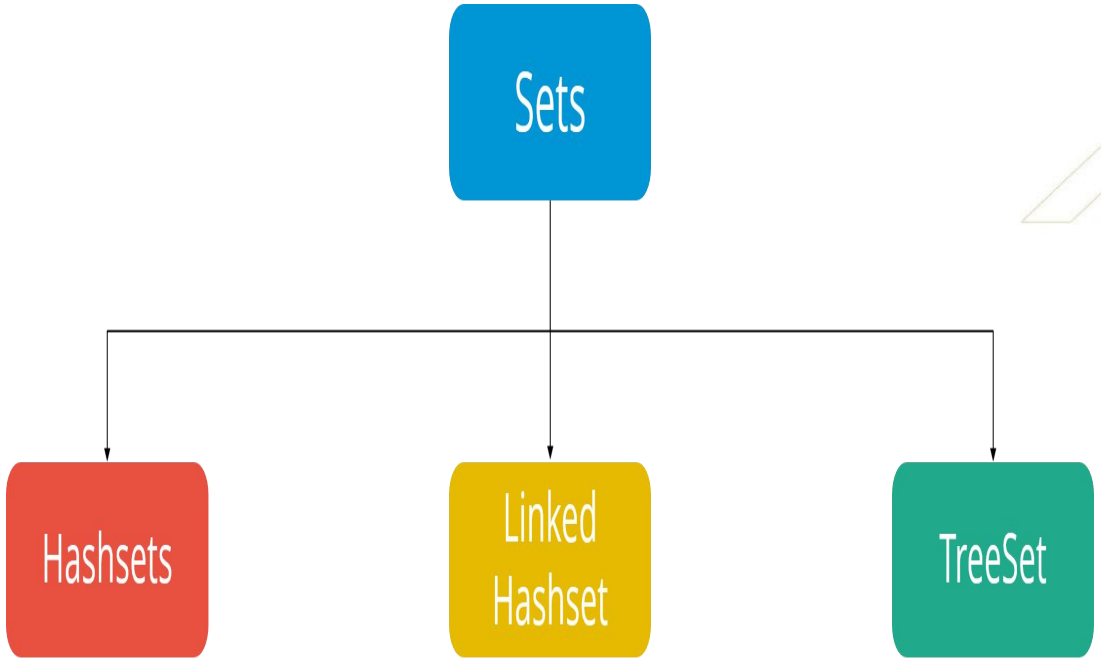
- The set interface is inherited from the Java collections Interface A Set interface cannot store duplicate/redundant elements in it.It store unique values Here's an example based on a set interface.

```
package VIPS;
import java.util.*;
public class SetExample {
public static void main(String args[]) {
int count[] = { 21, 23, 43, 53, 22, 65 };
Set<Integer> set = new HashSet<Integer>();
try {
for (int i = 0; i <= 5; i++) {
set.add(count[i]);}
System.out.println(set);
TreeSet<Integer> sortedSet =
new TreeSet<Integer>(set);
System.out.println("The sorted list is:");
System.out.println(sortedSet);
System.out.println("First element of the set is: " + (Integer)
sortedSet.first());
System.out.println("last element of the set is: " + (Integer)
sortedSet.last());
} catch (Exception e) {}}}
```

Output :[65, 21, 53, 22, 23, 43]
The sorted list is: [21, 22, 23, 43, 53, 65]
First element of the set is: 21
last element of the set is: 65

HashSet:

- Java **HashSet class** creates a collection that use a hash table for storage. Hashset only contain unique elements and it inherits the Abstract Set class and implements Set interface



Map Interface

- A Map is an object that maps keys to values. A map cannot contain duplicate keys: Each key can map to at most one value. It models the mathematical function abstraction.
- The Java platform contains three general-purpose Map implementations: HashMap, TreeMap, and LinkedHashMap. Their behavior and performance are precisely analogous to HashSet, TreeSet, and LinkedHashSet, as

```
import java.util.HashMap;
public class hashmap {
    public static void main(String[] args){
        HashMap hm=new HashMap();//HasMap store value in Key-value pair
        hm.put(67, "Shruti");
        hm.put(68, "Komal");
        hm.put(69, "Yashdeep");
        System.out.println(hm);
    }}
```

OUTPUT:{67=Shruti, 68=Komal, 69=Yashdeep}

How to reverse ArrayList?


```
import java.util.ArrayList;
import java.util.Collection;
import java.util.Collections;
import java.util.Iterator;
import java.util.List;
public class ReverseArrayList {
public static void main(String[] args) {
    List list = new ArrayList<>();
    list.add(10);
    list.add(50);
    list.add(30);
    Iterator i = list.iterator();
    System.out.println("printing the list....");
    while(i.hasNext())
    {
        System.out.println(i.next());
    }
    Iterator i2 = list.iterator();
    Collections.reverse(list);
    System.out.println("printing list in reverse order....");
    while(i2.hasNext())
    {
        System.out.println(i2.next());
    } } }
```

Outputprinting the list....

10

50

30

printing list in reverse order....

30 50 10

Hashset and Treeset example


```
package VIPS;
import java.util.HashSet;
public class HashSetClass {
public static void main(String args[]) {
HashSet<String> hset = new HashSet<String>();
hset.add("Suzuki");
hset.add("Kawasaki");
hset.add("Honda");
hset.add("Ducati");
hset.add("Yamaha");
hset.add("Yamaha");
hset.add("Suzuki");
hset.add(null);
hset.add(null);
// Displaying HashSet elements
System.out.println(hset);}}
```

OUTPUT:

[null, Suzuki, Ducati, Yamaha, Kawasaki, Honda]

TreeSet Class is a NavigableSet implementation based on TreeMap. Here, the elements are ordered using comparators.

Let us check an example based on TreeSet Class.

```
//TreeSet
package VIPS;
import java.util.TreeSet;
public class TreeSetClass {
    public static void main(String args[]) {
        TreeSet<Integer> treeset = new TreeSet<Integer>();
        treeset.add(8476);
        treeset.add(748);
        treeset.add(88);
        treeset.add(983);
        treeset.add(18);
        treeset.add(0);
        System.out.println(treeset);                                    }
 }
```

OUTPUT:

[0, 18, 88, 748, 983, 8476]

How to synchronize List, Set and Map elements?

- Yes, Collections class provides methods to make List, Set or Map elements as synchronized:

- public static List synchronizedList(List l){}
- public static Set synchronizedSet(Set s){}
- public static SortedSet synchronizedSortedSet(SortedSet s){}
- public static Map synchronizedMap(Map m){}
- public static SortedMap synchronizedSortedMap(SortedMap m){}

Q What is the advantage of the generic collection?

- There are three main advantages of using the generic collection.

- If we use the generic class, we don't need typecasting.
- It is type-safe and checked at compile time.
- Generic confirms the stability of the code by making it bug detectable at compile time.

Q What is hash-collision in Hashtable and how it is handled in Java?

- Two different keys with the same hash value are known as hash-collision. Two separate entries will be kept in a single hash bucket to avoid the collision. There are two ways to avoid hash-collision.
- Separate Chaining
- Open Addressing

Q What is the Dictionary class?

- The Dictionary class provides the capability to store key-value pairs.

- **QWhat is the default size of load factor in hashing based collection?**
- The default size of load factor is 0.75. The default capacity is computed as initial capacity * load factor. For example, 16 * 0.75 = 12. So, 12 is the default capacity of Map.

- **Q What do you understand by fail-fast?**
- The Iterator in java which immediately throws ConcurrentmodificationException, if any structural modification occurs in, is called as a Fail-fast iterator. Fail-fats iterator does not require any extra space in memory.

What is the difference between HashSet and TreeSet and HashMap?

- The HashSet and TreeSet, both classes, implement Set interface. The differences between the both are listed below.

- HashSet maintains no order whereas TreeSet maintains ascending order.
- HashSet impended by hash table whereas TreeSet implemented by a Tree structure.
- HashSet performs faster than TreeSet.
- HashSet is backed by HashMap whereas TreeSet is backed by TreeMap.

What is the difference between HashSet and HashMap?

- The differences between the HashSet and HashMap are listed below.
- HashSet contains only values whereas HashMap includes the entry (key, value). HashSet can be iterated, but HashMap needs to convert into Set to be iterated.
- HashSet implements Set interface whereas HashMap implements the Map interface
- HashSet cannot have any duplicate value whereas HashMap can contain duplicate values with unique keys.
- HashSet contains the only single number of null value whereas HashMap can hold a single null key with n number of null values.

What is the difference between Comparable and Comparator and Collection and Collections ?

The differences between the Collection and Collections are given below.

- The Collection is an interface whereas Collections is a class.
- The Collection interface provides the standard functionality of data structure to List, Set, and Queue. However, Collections class is to sort and synchronize the collection elements.
- The Collection interface provides the methods that can be used for data structure whereas Collections class provides the static methods which can be used for various operation on a collection.

Comparable	Comparator
Comparable provides only one sort of sequence.	The Comparator provides multiple sorts of sequences.
It provides one method named compareTo().	It provides one method named compare().
It is found in java.lang package	It is located in java.util package.
If we implement the Comparable interface, The actual class is modified.	The actual class is not changed. 1

List Interface

- The List interface is derived from the java util package. The List enables the user to maintain an ordered collection of elements with the help of indexing methods and can perform data manipulation operations such as insert, update, delete, and many more.

```
//List Interface
package VIPS;
import java.util.*;
public class ListInterface {
public static void main(String args[]) {
List<String> list = new ArrayList<String>();
list.add("David");
list.add("Jhon");
list.add("Stacy");
//list.add("Stacy");
for (String Students : list)
System.out.println(Students);
}
}
```

Java collections: Queue

- Queue in Java follows a **FIFO approach** i.e. it orders the elements in First 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 throws an exception if the operation fails, the other returns a special value.Queue is in java is a collection that is capable to store and apply operations on elements
- Priority queue implements Queue interface. The elements of the priority queue are ordered according to their natural ordering, or by a Comparator provided at the queue construction time. The head of this queue is the least element with respect to the specified ordering.

Example Queue


```
package VIPS;
import java.util.*;
public class QueueInterface {
public static void main(String args[]) {
Queue<String> queue = new
LinkedList<String>();
queue.add("Rose");
queue.add("Lotus");
queue.add("Habiscus");
queue.add("Lily");
System.out.println(queue);
queue.remove("Lily");

System.out.println("Queue total size: "
+queue.size());
System.out.println("Queue includes
flowers 'Lily': ?" +
queue.contains("Lily"));
 queue.clear();}
}
```

Output

[Rose, Lotus, Habiscus, Lily]
Queue total size: 3
Queue includes flowers 'Lily': ?false

What do you understand by BlockingQueue ?

- BlockingQueue is an interface which extends the Queue interface. It provides concurrency in the operations like retrieval, insertion, deletion. While retrieval of any element, it waits for the queue to be non-empty. While storing the elements, it waits for the available space. BlockingQueue cannot contain null elements, and implementation of BlockingQueue is thread-safe.
- Syntax
- public interface BlockingQueue<E> extends Queue <E>

What does the hashCode() method?

- The hashCode() method returns a hash code value (an integer number).
- The hashCode() method returns the same integer number if two keys (by calling equals() method) are identical.
- However, it is possible that two hash code numbers can have different or the same keys.
- If two objects do not produce an equal result by using the equals() method, then the hashcode() method will provide the different integer result for both the objects.

Why we override equals() method?

- The equals method is used to check whether two objects are the same or not. It needs to be overridden if we want to check the objects based on the property.

- For example, Employee is a class that has 3 data members: id, name, and salary. However, we want to check the equality of employee object by the salary. Then, we need to override the equals() method.

How to convert ArrayList to Array and Array to ArrayList?

- We can convert an Array to ArrayList by using the asList() method of Arrays class. asList() method is the static method of Arrays class and accepts the List object. Consider the following syntax:

- Arrays.asList(item)
- We can convert an ArrayList to Array using toArray() method of the ArrayList class. Consider the following syntax to convert the ArrayList to the List object.

- List_object.toArray(new String[List_object.size()])

How to make Java ArrayList Read-Only?

- We can obtain java ArrayList Read-only by calling the Collections.unmodifiableCollection() method. When we define an ArrayList as Read-only then we cannot perform any modification in the collection through add(), remove() or set() method.

Collection package Question

Q When to use ArrayList and LinkedList?

- LinkedLists are better to use for the update operations whereas ArrayLists are better to use for the search operations.

Q How to synchronize ArrayList?

- We can synchronize ArrayList in two ways.
- Using Collections.synchronizedList() method
- Using CopyOnWriteArrayList<T>

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.

PriorityQueue

	Throws Exception	Returns Special Value
Insert	Add(e)	Offer(e)
Remove	Remove()	Poll()
Examine	Element()	Peek()

PriorityQueue


```
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());
      }
   }
}
```

Output:

```
head:Amit
head:Amit
iterating the queue elements:
Amit
Rachit
Rahul
after removing two elements:
Rahul
```

Queue Interface

- A Queue interface is inherited from the Java Collections interface. The Queue is a linear Collection that offers data manipulation operations on the collection elements, and follows the FIFO(First In First Out) principle. For example:

```
Example://Queue Interface
package VIPS;
import java.util.*;
public class QueueInterface {public static void main(String[] args) {
Queue<String> queue = new LinkedList<>();
queue.add("Apple");queue.add("Mango");
queue.add("Grapes");
queue.add("Banana");
System.out.println(queue);
queue.remove("Grapes");
System.out.println(queue);
System.out.println("Queue total Size: " + queue.size());
System.out.println("Queue includes fruit 'Apple'? : " +
queue.contains("Apple"));
queue.clear();
}}
```

Deque interface

- A Deque interface is inherited from the Java Collections Interface. The term Deque stands for Double-Ended Queue. The Deque supports insertion and deletion operations on both sides of the Queue.

```
Example://Deque Interface

package VIPS;

import java.util.ArrayDeque;

import java.util.Deque;

public class DequeInterface {public static void main(String[] args) {

Deque<Integer> num = new ArrayDeque<>();

num.offer(10);num.offerLast(21);num.offerFirst(52);

System.out.println("Deque elements: " + num);

int first = num.peekFirst();

System.out.println("First Element is: " + first);

int lastElement = num.peekLast();System.out.println("Last Element: "
+ lastElement);

int removed = num.pollFirst();System.out.println("Removed First
Element: " + removed);System.out.println("Updated Deque is: " +
num);

}}
```

Thank You