Object Oriented Programming with Java

AKTU

Unit :- 4

BCS-403

Java Collections Framework

Java Collections Framework: Collection in Java, Collection Framework in Java, Hierarchy of Collection Framework, Iterator Interface, Collection Interface, List Interface, ArrayList, LinkedList, Vector, Stack, Queue Interface, Set Interface, HashSet, LinkedHashSet, SortedSet Interface, TreeSet, Map Interface, HashMap Class, LinkedHashMap Class, TreeMap Class, Hashtable Class, Sorting, Comparable Interface, Comparator Interface, Properties Class in Java.

Collections in Java

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

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.

In Java, "Collections" refers to a framework provided in the Java API to manage and manipulate groups of objects. These objects are commonly referred to as elements or items.

The Collections framework provides a unified architecture for working with collections of objects. It allows developers to easily store, retrieve, manipulate, and iterate over these collections.

Key features of Collections:

1. **Unified architecture:** The Collections framework provides a unified architecture for representing and manipulating collections.
2. **Generics support:** Most classes and interfaces in the Collections framework support generics, allowing developers to specify the type of elements a collection can hold.
3. **Dynamic resizing:** Many collection classes automatically resize themselves as needed to accommodate the addition or removal of elements.
4. **Algorithms:** The Collections framework includes various utility methods and algorithms for sorting, searching, and manipulating collections efficiently.
5. **Iterators:** Iterators provide a way to traverse the elements of a collection sequentially, enabling easy iteration over collection elements.

What is a framework in Java

- It provides readymade architecture.
- It represents a set of classes and interfaces.
- 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

The Java Collections framework is a set of classes and interfaces that provide implementations of commonly reusable collection data structures in Java.

The data structures can be lists, sets, maps, queues, etc.

It provides a unified architecture for manipulating and storing groups of objects.

Advantages of Java Collection Framework :-

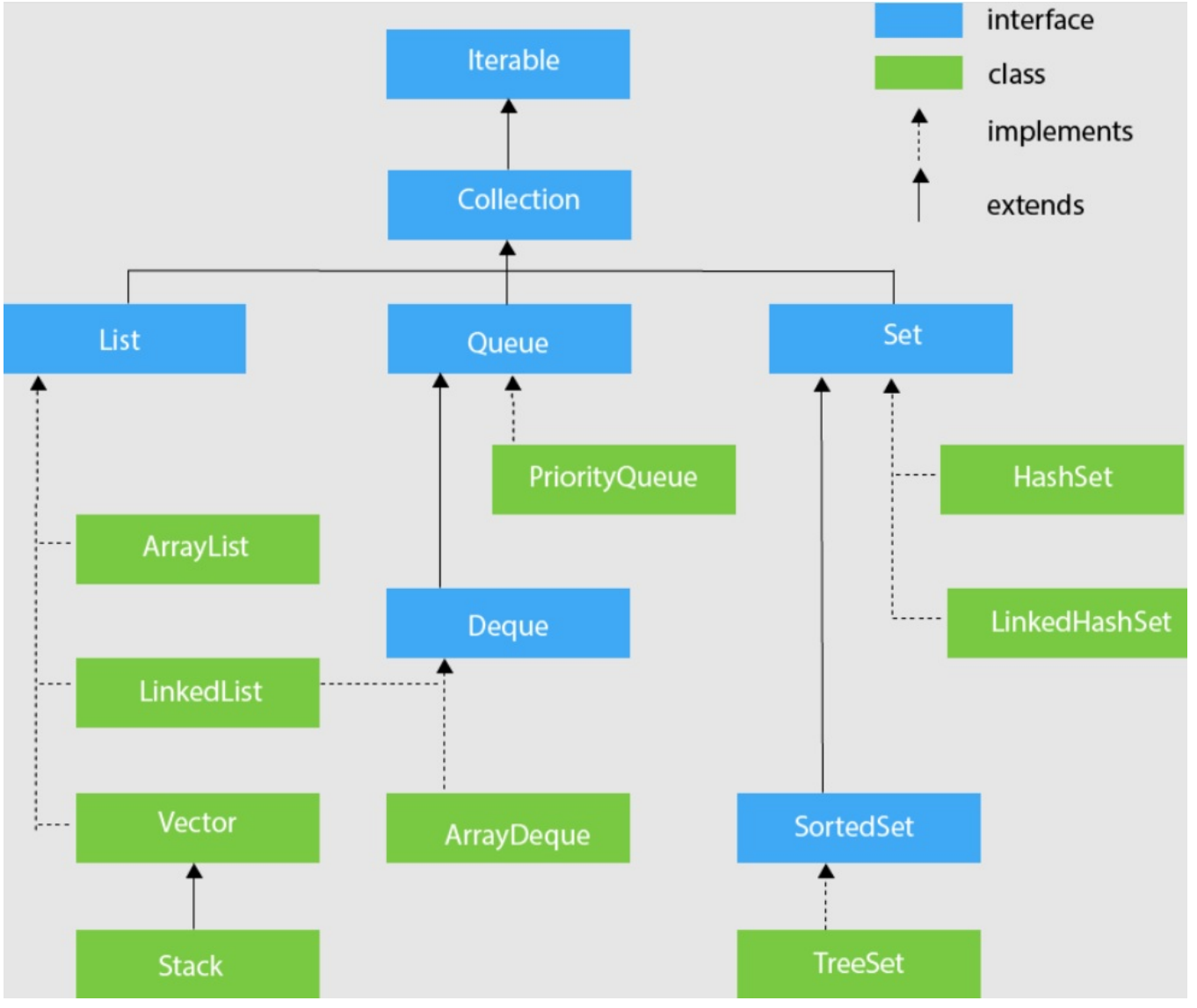
1. Reusable Implementations
2. Consistency
3. Efficiency
4. Type Safety
5. Scalability

Hierarchy of Collection Framework

1. The Utility Package (java.util) contains all the classes & interfaces that are required by the collection framework.

2. The collection framework contains an interface named an iterable interface which provides the iterator to iterate through all the collections. This interface is extended by the main collection interface which acts as a root for the collection framework.

3. All the collections extend this collection interface thereby extending the properties of the iterator and the methods of this interface.



Iterator Interface

1. This is the root interface for the entire collection framework.
2. The Iterator interface is used to traverse through elements of a collection.
3. It provides a uniform way to access elements of various collection types without exposing the underlying implementation details.

Purpose of the Iterator interface

1. **Traversal:** It allows sequential access to the elements of a collection. This enables iterating over the elements of a collection without needing to know its internal structure.
2. **Uniformity:** Iterator provides a common way to iterate over different types of collections. This uniformity simplifies the process of iterating through collections.
3. **Safe removal:** Iterator supports safe removal of elements from a collection during iteration.
4. **Enhanced for loop support:** The Iterator interface is utilized implicitly in the enhanced 'for' loop syntax. This syntax provides a concise and readable way to iterate over collections without explicitly dealing with iterators.

Collection Interfaces

The Collection interface is the root interface in the Java Collections framework hierarchy. It represents a group of objects, known as elements, and provides a unified way to work with collections of objects in Java.

The Collection interface defines a set of operations that can be performed on collections, regardless of their specific implementation. The Collection interface does not specify any particular ordering of elements.

The Collection interface allows duplicate elements.

Overall, the Collection interface serves as a fundamental building block for working with collections of objects in Java. It provides a common set of operations and behaviours that can be used across different types of collections.

Differentiate between Collection and Collections.

Collection is called interface in java whereas Collections is called a utility class in java and both of them can be found in java. util. package. Collection is used to represent a single unit with a group of individual objects whereas collections is used to operate on collection with several utility methods.

Collection	Collections
java.util.Collection is an interface	java.util.Collections is a class
Is used to represent a group of objects as a 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 structures of the Collection framework	It contains various static methods which help in data structure manipulation

List Interface

This is a child interface of the collection interface.
This interface is dedicated to the data of the list type in which we can store all the ordered collections of the objects.
This also allows duplicate data to be present in it. This list interface is implemented by various classes like ArrayList, Vector, Stack, etc.
Since all the subclasses implement the list, we can instantiate a list object with any of these classes.

Key Characteristics of List Interface :-

1. Ordered Collections
2. Indexed Access
3. Dynamic Size
4. Iterable
5. Search Operations

The classes that implement the List interface are given below.

ArrayList

The ArrayList class is a resizable array, which can be found in the java.util package. The difference between a built-in array and an ArrayList in Java, is that the size of an array cannot be modified (if you want to add or remove elements to/from an array, you have to create a new one).

How the ArrayList works

The `ArrayList` class has a regular array inside it. When an element is added, it is placed into the array. If the array is not big enough, a new, larger array is created to replace the old one and the old one is removed.

The ArrayList class implements the List interface. It uses a dynamic array to store the duplicate element of different data types. The ArrayList class maintains the insertion order and is non-synchronized. The elements stored in the ArrayList class can be randomly accessed. Consider the following example.

```
import java.util.*;
class TestJavaCollection1{
public static void main(String args[]){
ArrayList<String> list=new ArrayList<String>();//Creating arraylist
list.add("Ravi");//Adding object in arraylist
list.add("Vijay");
list.add("Ravi");
list.add("Ajay");
//Traversing list through Iterator
Iterator itr=list.iterator();
while(itr.hasNext()){
System.out.println(itr.next());
}
}
}
```

Output:

```
Ravi
Vijay
Ravi
Ajay
```

LinkedList

The `LinkedList` class is a collection which can contain many objects of the same type, just like the `ArrayList`.

The `LinkedList` class has all of the same methods as the `ArrayList` class because they both implement the `List` interface. This means that you can add items, change items, remove items and clear the list in the same way.

However, while the `ArrayList` class and the `LinkedList` class can be used in the same way, they are built very differently.

How the LinkedList works

The `LinkedList` stores its items in "containers." The list has a link to the first container and each container has a link to the next container in the list. To add an element to the list, the element is placed into a new container and that container is linked to one of the other containers in the list.

LinkedList implements the Collection interface. It uses a doubly linked list internally to store the elements. It can store the duplicate elements. It maintains the insertion order and is not synchronized. In LinkedList, the manipulation is fast because no shifting is required.

Consider the following example.

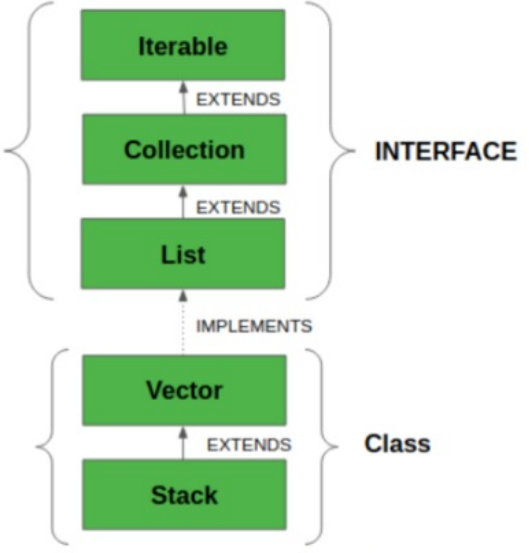
```
import java.util.*;
public class TestJavaCollection2{
public static void main(String args[]){
LinkedList<String> al=new LinkedList<String>();
al.add("Ravi");
al.add("Vijay");
al.add("Ravi");
al.add("Ajay");
Iterator<String> itr=al.iterator();
while(itr.hasNext()){
System.out.println(itr.next());
}
}
}
```

Output:

```
Ravi
Vijay
Ravi
Ajay
```

Vector

The Vector class implements a growable array of objects. Vectors fall in legacy classes, but now it is fully compatible with collections. It is found in java.util package and implement the List interface, so we can use all the methods of the List interface as shown below as follows:



Vector implements a dynamic array which means it can grow or shrink as required. Like an array, it contains components that can be accessed using an integer index.

- They are very similar to ArrayList, but Vector is synchronized and has some legacy methods that the collection framework does not contain.
- It also maintains an insertion order like an ArrayList. Still, it is rarely used in a non-thread environment as it is **synchronized**, and due to this, it gives a poor performance in adding, searching, deleting, and updating its elements.
- The Iterators returned by the Vector class are fail-fast. In the case of concurrent modification, it fails and throws the **ConcurrentModificationException.**

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 part of Collection framework.

Consider the following example.

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

Output:

```
Ayush
Amit
Ashish
Garima
```

Stack

Java Collection framework provides a Stack class that models and implements a **Stack data structure**. The class is based on the basic principle of last-in-first-out. In addition to the basic push and pop operations, the class provides three more functions of empty, search, and peek. The class can also be said to extend Vector and treats the class as a stack with the five mentioned functions. The class can also be referred to as 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.

Consider the following example.

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

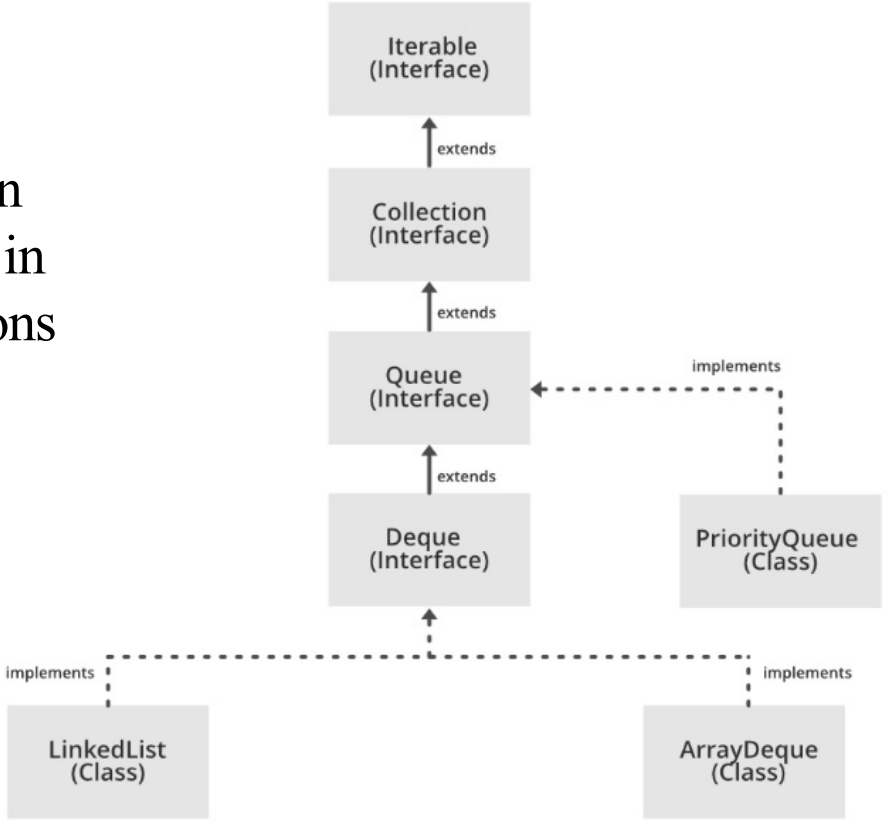
Output:

```
Ayush
Garvit
Amit
Ashish
```

Queue Interface

The Queue interface is present in java.util package and extends the Collection interface is used to hold the elements about to be processed in FIFO(First In First Out) order. It is an ordered list of objects with its use limited to inserting elements at the end of the list and deleting elements from the start of the list, (i.e.), it follows the **FIFO** or the First-In-First-Out principle.

Being an interface the queue needs a concrete class for the declaration and the most common classes are the PriorityQueue and LinkedList in Java. Note that neither of these implementations is thread-safe. PriorityBlockingQueue is one alternative implementation if the thread-safe implementation is needed



Queue interface maintains the first-in-first-out order.
It can be defined as an ordered list that is used to hold the elements which are about to be processed.
There are various classes like PriorityQueue, Deque, and ArrayDeque which implements the Queue interface.

Queue interface can be instantiated as:

```
. Queue<String> q1 = new PriorityQueue();

. Queue<String> q2 = new ArrayDeque();
```

There are various classes that implement the Queue interface, some of them are given below.

PriorityQueue

A PriorityQueue is used when the objects are supposed to be processed based on the priority. It is known that a Queue follows the First-In-First-Out algorithm, but sometimes the elements of the queue are needed to be processed according to the priority, that's when the PriorityQueue comes into play.

The PriorityQueue is based on the priority heap. The elements of the priority queue are ordered according to the natural ordering, or by a Comparator provided at queue construction time, depending on which constructor is used.

In the below priority queue, an element with a maximum ASCII value will have the highest priority.

The PriorityQueue class implements the Queue interface. It holds the elements or objects which are to be processed by their priorities. PriorityQueue doesn't allow null values to be stored in the queue.

Consider the following example.

```
import java.util.*;
public class TestJavaCollection5{
public static void main(String args[]){
PriorityQueue<String> queue=new PriorityQueue<String>();
queue.add("Amit Sharma");
queue.add("Vijay Raj");
queue.add("JaiShankar");
queue.add("Raj");
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 Sharma
head:Amit Sharma
iterating the queue elements:
Amit Sharma
Raj
JaiShankar
Vijay Raj
after removing two elements:
Raj
Vijay Raj
```

Deque Interface

The Deque (double-ended queue) interface in Java is a subinterface of the Queue interface and extends it to provide a double-ended queue, which is a queue that allows elements to be added and removed from both ends. The Deque interface is part of the Java Collections Framework and is used to provide a generic and flexible data structure that can be used to implement a variety of algorithms and data structures.

Deque interface extends the Queue interface. In Deque, we can remove and add the elements from both the side. Deque stands for a double-ended queue which enables us to perform the operations at both the ends.

Deque can be instantiated as:

```
Deque d = new ArrayDeque();
```

ArrayDeque

The ArrayDeque class in Java is an implementation of the Deque interface that uses a resizable array to store its elements. This class provides a more efficient alternative to the traditional Stack class, which was previously used for double-ended operations. The ArrayDeque class provides constant-time performance for inserting and removing elements from both ends of the queue, making it a good choice for scenarios where you need to perform many add and remove operations.

ArrayDeque class implements the Deque interface. It facilitates us to use the Deque. Unlike queue, we can add or delete the elements from both the ends.

ArrayDeque is faster than ArrayList and Stack and has no capacity restrictions.

Consider the following example.

```
import java.util.*;
public class TestJavaCollection6{
public static void main(String[] args) {
//Creating Deque and adding elements
Deque<String> deque = new ArrayDeque<String>();
deque.add("Gautam");
deque.add("Karan");
deque.add("Ajay");
//Traversing elements
for (String str : deque) {
System.out.println(str);
}
}
}
```

Output:

```
Gautam
Karan
Ajay
```

Set Interface

The set interface is present in java.util package and extends the Collection interface. It is an unordered collection of objects in which duplicate values cannot be stored. It is an interface that implements the mathematical set. This interface contains the methods inherited from the Collection interface and adds a feature that restricts the insertion of the duplicate elements. There are two interfaces that extend the set implementation namely SortedSet and NavigableSet.

In the above image, the navigable set extends the sorted set interface. Since a set doesn't retain the insertion order, the navigable set interface provides the implementation to navigate through the Set. The class which implements the navigable set is a TreeSet which is an implementation of a self-balancing tree. Therefore, this interface provides us with a way to navigate through this tree.

Set Interface in Java is present in java.util package. It extends the Collection interface It represents the unordered set of elements which doesn't allow us to store the duplicate items. We can store at most one null value in Set. Set is implemented by HashSet, LinkedHashSet, and TreeSet. Set can be instantiated as:

```
Set<data-type> s1 = new HashSet<data-type>();
Set<data-type> s2 = new LinkedHashSet<data-type>();
Set<data-type> s3 = new TreeSet<data-type>();
```

HashSet

Java HashSet class implements the Set interface, backed by a hash table which is actually a HashMap instance. No guarantee is made as to the iteration order of the hash sets which means that the class does not guarantee the constant order of elements over time. This class permits the null element. The class also offers constant time performance for the basic operations like add, remove, contains, and size assuming the hash function disperses the elements properly among the buckets, which we shall see further.

Java HashSet Features

A few important features of HashSet are mentioned below:

- Implements Set Interface.
- The underlying data structure for HashSet is Hashtable.
- As it implements the Set Interface, duplicate values are not allowed.
- Objects that you insert in HashSet are not guaranteed to be inserted in the same order. Objects are inserted based on their hash code.
- NULL elements are allowed in HashSet.
- HashSet also implements **Serializable** and **Cloneable** interfaces.

Declaration of HashSet

```
public class HashSet<E> extends AbstractSet<E> implements Set<E>,
Cloneable, Serializable
```

where **E** is the type of elements stored in a HashSet

Consider the following example.

```
import java.util.*;
public class TestJavaCollection7{
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());
}
}
}
```

Output:

```
Vijay
Ravi
Ajay
```

Must SUBSCRIBE

I Tech World (AKTU)

Ritik ThAkur

B.Tech (IT)

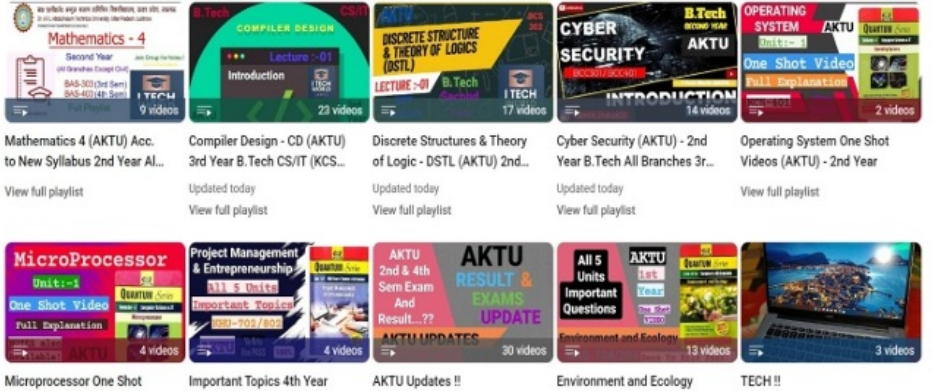
Instagram

@ritik_thakur00

HOME VIDEOS SHORTS PLAYLISTS COMMUNITY MEMBERSHIP CHANNELS ABOUT

Created playlists

Sort by



Mathematics 4 (AKTU) Acc. to New Syllabus 2nd Year Al...

View full playlist

Compiler Design - CD (AKTU) 3rd Year B.Tech CS/IT (KCS...

Updated today

View full playlist

Discrete Structures & Theory of Logic - DSTL (AKTU) 2nd...

Updated today

View full playlist

Cyber Security (AKTU) - 2nd Year B.Tech All Branches 3r...

Updated today

View full playlist

Operating System One Shot Videos (AKTU) - 2nd Year

View full playlist

Microprocessor One Shot

Important Topics 4th Year

AKTU Updates !!

Environment and Ecology

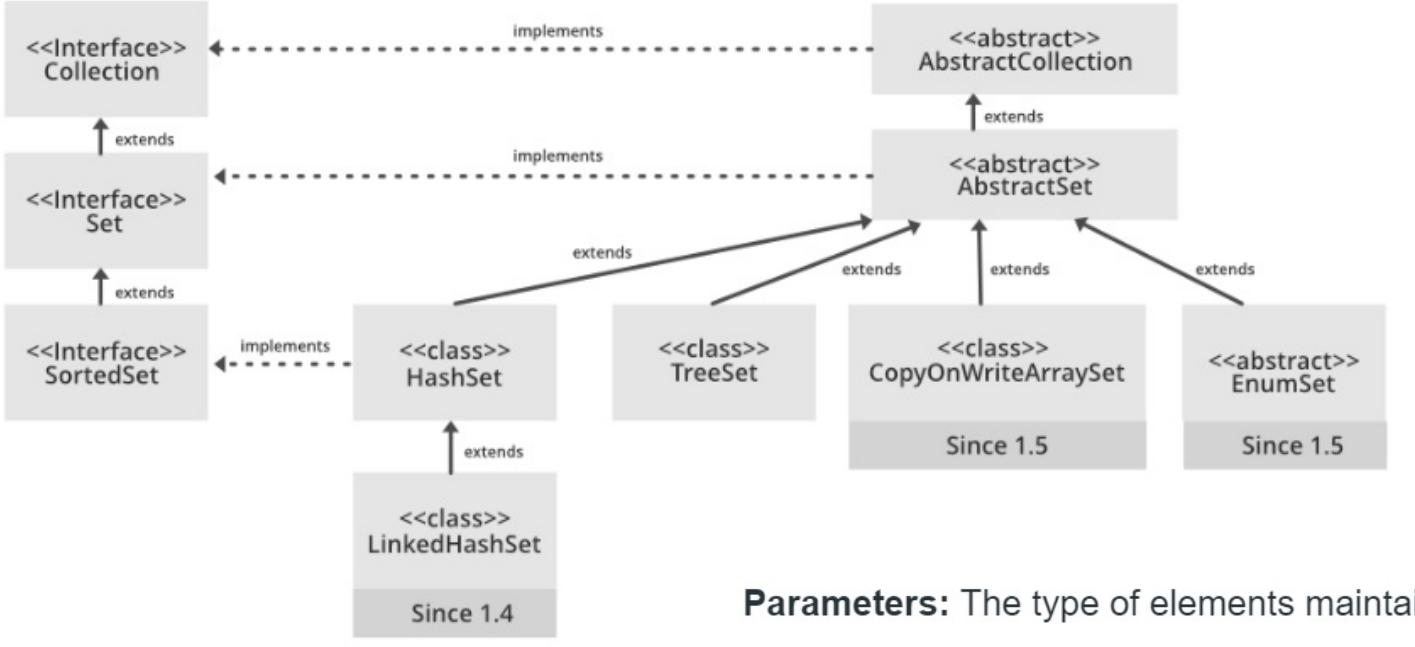
TECH !!

SUBSCRIBE

LinkedHashSet

The **LinkedHashSet** is an ordered version of HashSet that maintains a doubly-linked List across all elements. When the iteration order is needed to be maintained this class is used. When iterating through a HashSet the order is unpredictable, while a LinkedHashSet lets us iterate through the elements in the order in which they were inserted. When cycling through LinkedHashSet using an iterator, the elements will be returned in the order in which they were inserted.

The Hierarchy of LinkedHashSet is as follows:



Parameters: The type of elements maintained by this set

LinkedHashSet class represents the LinkedList implementation of Set Interface. It extends the HashSet class and implements Set interface. Like HashSet, It also contains unique elements. It maintains the insertion order and permits null elements.

Consider the following example.

```
import java.util.*;
public class TestJavaCollection8{
public static void main(String args[]){
LinkedHashSet<String> set=new LinkedHashSet<String>();
set.add("Ravi");
set.add("Vijay");
set.add("Ravi");
set.add("Ajay");
Iterator<String> itr=set.iterator();
while(itr.hasNext()){
System.out.println(itr.next());
}
}
}
```

Output:

```
Ravi
Vijay
```

SortedSet Interface

The SortedSet interface present in java.util package extends the Set interface present in the collection framework. It is an interface that implements the mathematical set. This interface contains the methods inherited from the Set interface and adds a feature that stores all the elements in this interface to be stored in a sorted manner.

Key Characteristics :-

1. Sorted order: Elements in a SortedSet are stored in sorted order.
2. Uniqueness: Like other Set implementations, a SortedSet does not allow duplicate elements.
3. Efficient retrieval: SortedSet provides efficient methods for retrieving elements based on their sorted order.
4. No index-based access: Unlike List implementations, SortedSet does not support index-based access to elements

TreeSet

TreeSet is one of the most important implementations of the SortedSet interface in Java that uses a Tree for storage. The ordering of the elements is maintained by a set using their natural ordering whether or not an explicit comparator is provided. This must be consistent with equals if it is to correctly implement the Set interface.

Java TreeSet class implements the Set interface that uses a tree for storage. Like HashSet, TreeSet also contains unique elements. However, the access and retrieval time of TreeSet is quite fast. The elements in TreeSet stored in ascending order.

Key Characteristics :-

1. Sorted order
2. Unique elements
3. Efficient operations
4. Iterating in sorted order
5. Use of Red-Black tree

Map Interface

1. The Map interface in Java represents a collection of key-value pairs, where each key is associated with a corresponding value.
2. It provides a way to store and retrieve elements based on their keys.
3. Keys are unique within a map, meaning that each key can map to at most one value.
4. The Map interface provides an efficient retrieval and manipulation of values based on their keys.
5. This makes maps suitable for scenarios where quick access to values based on unique identifiers is required.

Key Characteristics :-

1. Key - Value Pairs :- Map stores Elements as key-value pairs, where each key is associated with a corresponding value.
2. Uniqueness of Keys :- Each Key in a map is unique.

3. No Ordering :- Maps do not maintain any inherent ordering of their Elements.

4. Effiecient Retrieval :- Maps provide effiecient method for retrieval values based on their corresponding keys.

Role of the Map Interface in Java Collection Framework :-

1. Associative data structure: Maps provide an associative data structure that allows elements to be stored and retrieved based on unique keys.

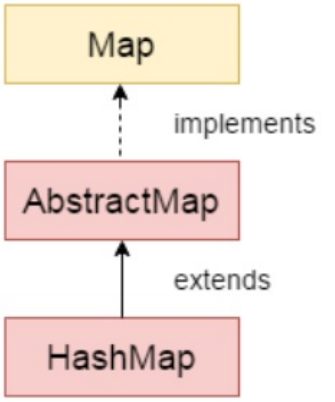
2. Flexible storage: Maps offer flexibility in storing and organizing data.

3. Key uniqueness: Maps enforce the uniqueness of keys within the collection.

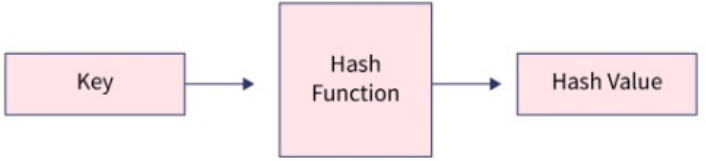
4. Integration with Collections framework: The Map interface integrates seamlessly with other interfaces and classes in the Java Collections framework.

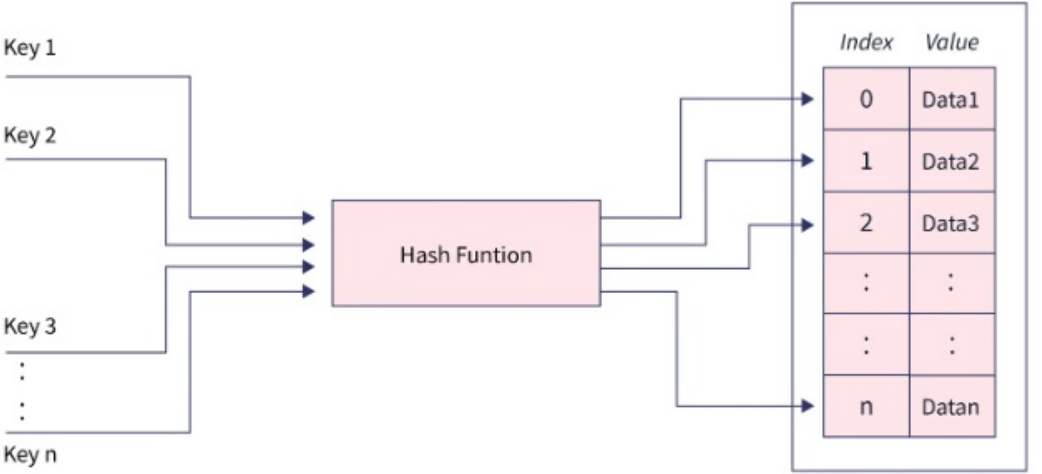
5. Multiple implementations: The Map interface allows multiple implementations with different performance characteristics and usage patterns

Java HashMap



Java **HashMap** class implements the Map interface which allows us *to store key and value pair*, where keys should be unique. If you try to insert the duplicate key, it will replace the element of the corresponding key. It is easy to perform operations using the key index like updation, deletion, etc. HashMap class is found in the `java.util` package.



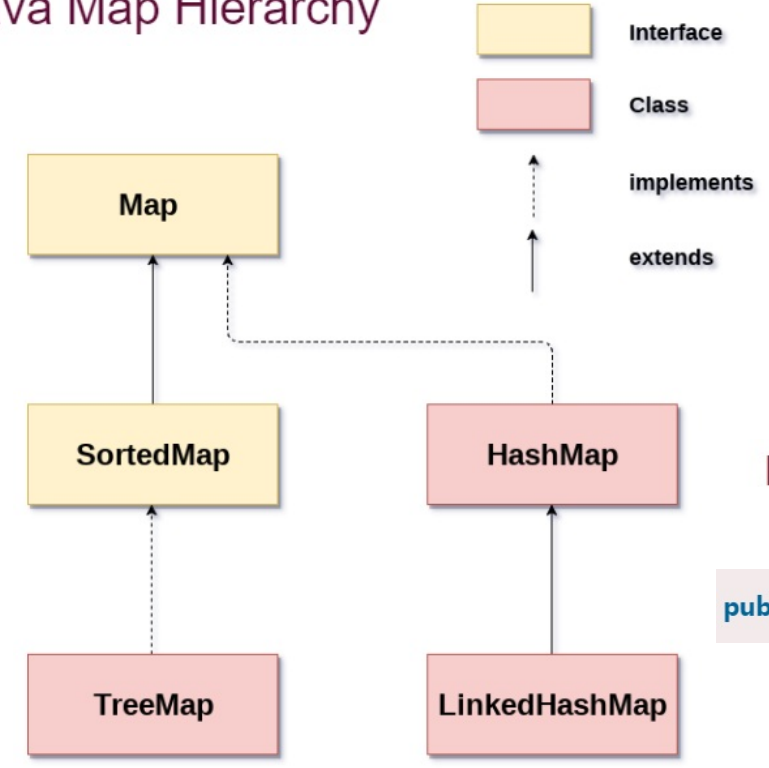


HashMap in Java is like the legacy Hashtable class, but it is not synchronized. It allows us to store the null elements as well, but there should be only one null key. Since Java 5, it is denoted as `HashMap<K,V>`, where K stands for key and V for value. It inherits the AbstractMap class and implements the Map interface.

Points to remember

- Java HashMap contains values based on the key.
- Java HashMap contains only unique keys.
- Java HashMap may have one null key and multiple null values.
- Java HashMap is non synchronized.
- Java HashMap maintains no order.

Java Map Hierarchy



HashMap class Parameters

Let's see the Parameters for java.util.HashMap class.

- **K**: It is the type of keys maintained by this map.
- **V**: It is the type of mapped values.

HashMap class declaration

```
public class HashMap<K,V> extends AbstractMap<K,V> implements Map<K,V>, Cloneable, Serializa
```

LinkedHashMap

The **LinkedHashMap Class** is just like HashMap with an additional feature of maintaining an order of elements inserted into it. HashMap provided the advantage of quick insertion, search, and deletion but it never maintained the track and order of insertion, which the LinkedHashMap provides where the elements can be accessed in their insertion order.

Important Features of a LinkedHashMap are listed as

- A LinkedHashMap contains values based on the key. It implements the Map interface and extends the HashMap class.
- It contains only unique elements.
- It may have one null key and multiple null values.
- It is non-synchronized.
- It is the same as HashMap with an additional feature that it maintains insertion order. For example, when we run the code with a HashMap, we get a different order of elements.

Declaration:

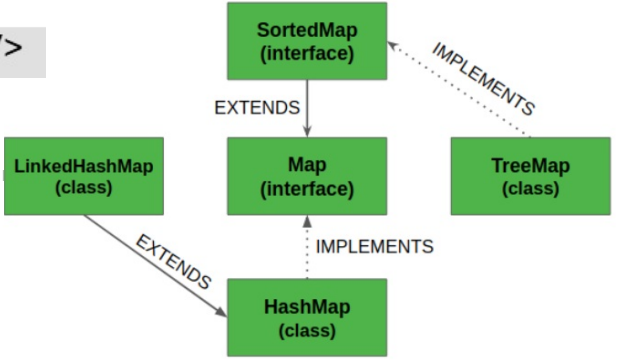
```
public class LinkedHashMap<K,V> extends HashMap<K,V> implements Map<K,V>
```

Here, **K** is the key Object type and **V** is the value Object type

- **K** – The type of the keys in the map.
- **V** – The type of values mapped in the map.

It implements Map<K, V> interface, and extends HashMap<K, V> class.

Though the Hierarchy of LinkedHashMap is as depicted as follows:



MAP Hierarchy in Java

How LinkedHashMap Work Internally?

A LinkedHashMap is an extension of the **HashMap** class and it implements the **Map** interface.Therefore, the class is declared as:

```
public class LinkedHashMap
extends HashMap
implements Map
```

In this class, the data is stored in the form of nodes. The implementation of the LinkedHashMap is very similar to a doubly-linked list. Therefore, each node of the LinkedHashMap is represented as:

- **Hash:** All the input keys are converted into a hash which is a shorter form of the key so that the search and insertion are faster.
- **Key:** Since this class extends HashMap, the data is stored in the form of a key-value pair. Therefore, this parameter is the key to the data.
- **Value:** For every key, there is a value associated with it. This parameter stores the value of the keys. Due to generics, this value can be of any form.
- **Next:** Since the LinkedHashMap stores the insertion order, this contains the address to the next node of the LinkedHashMap.
- **Previous:** This parameter contains the address to the previous node of the LinkedHashMap.

Property	HashMap	TreeMap	LinkedHashMap	HashTable
Iteration Order	Random	Sorted according to natural **order of keys**	Sorted according to the **insertion order**.	Random
Efficiency: Get, Put, Remove, ContainsKey	$O(1)$	$O(\log(n))$	$O(1)$	$O(1)$
Null keys/values	allowed	Not-allowed*	allowed	Not-allowed
Interfaces	Map	Map, SortedMap, NavigableMap	Map	Map
Synchronized	Not instead use Collection.synchronizedMap(new HashMap())			Yes but prefer to use ConcurrentHashMap
Implementation	Buckets	Red-Black tree	HashTable and LinkedList using doubly linked list of buckets	Buckets
Comments	Efficient	Extra cost of maintaining TreeMap	Advantage of TreeMap without extra cost.	Obsolete

Hash table in Java

The **Hashtable** class implements a hash table, which maps keys to values. Any non-null object can be used as a key or as a value. To successfully store and retrieve objects from a hashtable, the objects used as keys must implement the hashCode method and the equals method.

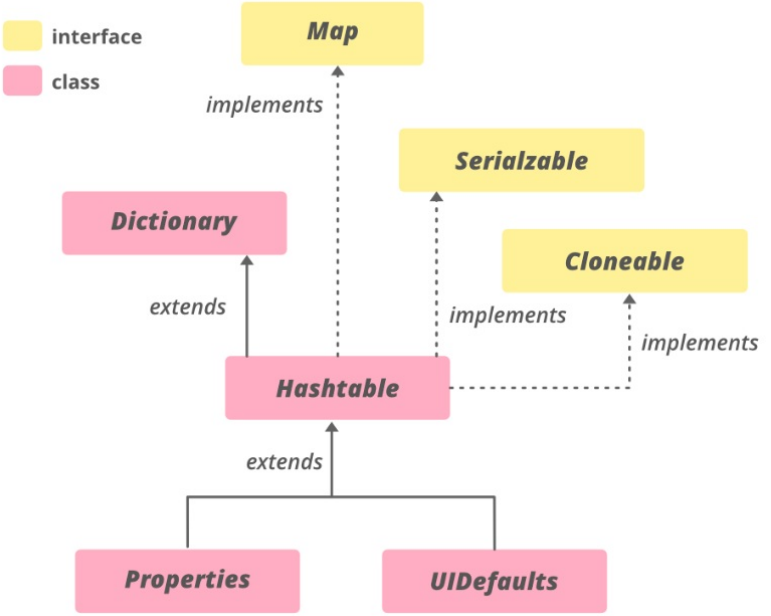
The java.util.Hashtable class is a class in Java that provides a key-value data structure, similar to the Map interface. It was part of the original Java Collections framework and was introduced in Java 1.0.

However, the Hashtable class has since been considered obsolete and its use is generally discouraged. This is because it was designed prior to the introduction of the Collections framework and does not implement the Map interface, which makes it difficult to use in conjunction with other parts of the framework. In addition, the Hashtable class is synchronized, which can result in slower performance compared to other implementations of the Map interface.

In general, it's recommended to use the Map interface or one of its implementations (such as HashMap or ConcurrentHashMap) instead of the Hashtable class.

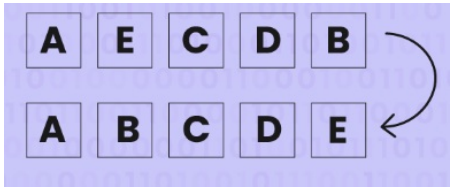
Features of Hashtable

- It is similar to HashMap, but is synchronized.
- Hashtable stores key/value pair in hash table.
- In Hashtable we specify an object that is used as a key, and the value we want to associate to that key.The key is then hashed, and the resulting hash code is used as the index at which the value is stored within the table.
- HashMap doesn’t provide any Enumeration, while Hashtable provides not fail-fast Enumeration.



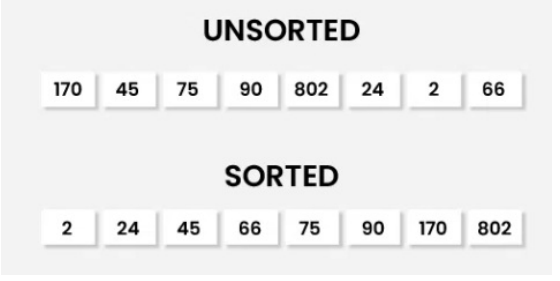
The Hierarchy of Hashtable

Sorting in Java



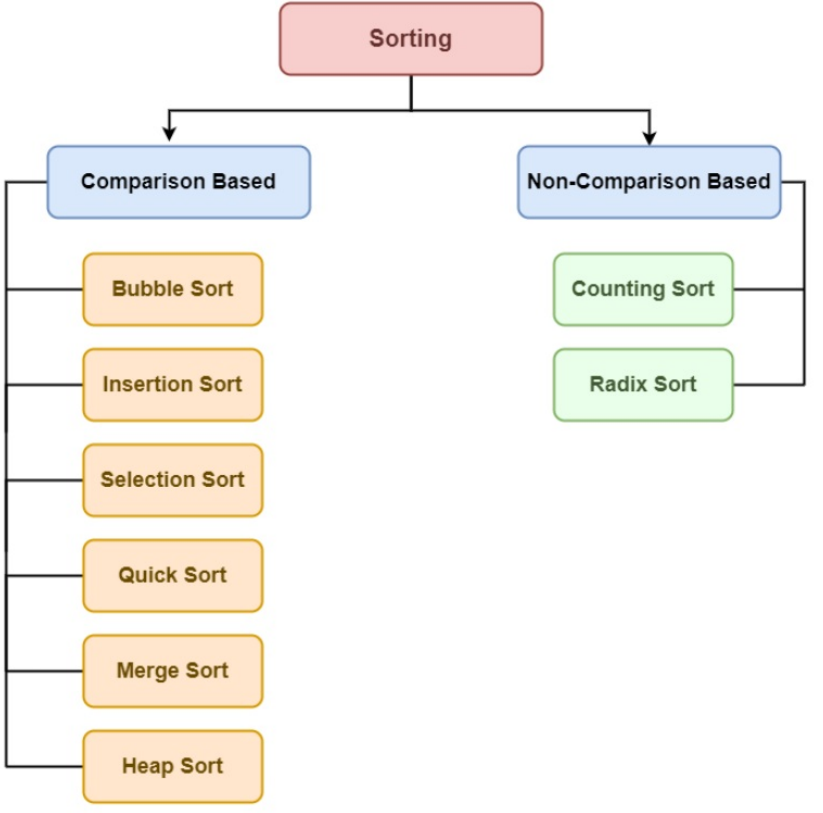
Whenever we do hear sorting algorithms come into play such as selection sort, bubble sort, insertion sort, radix sort, bucket sort, etc but if we look closer here we are not asked to use any kind of algorithms.

It is as simple sorting with the help of linear and non-linear data structures present within java.



Ways of sorting in Java

1. Using loops
2. Using sort() method of Arrays class
3. Using sort method of Collections class
4. Sorting on a subarray



Java Comparable interface

Java Comparable interface is used to order the objects of the user-defined class. This interface is found in java.lang package and contains only one method named compareTo(Object). It provides a single sorting sequence only, i.e., you can sort the elements on the basis of single data member only. For example, it may be rollno, name, age or anything else.

compareTo(Object obj) method

public int compareTo(Object obj): It is used to compare the current object with the specified object. It returns

- positive integer, if the current object is greater than the specified object.
- negative integer, if the current object is less than the specified object.
- zero, if the current object is equal to the specified object.

We can sort the elements of:

1. String objects
2. Wrapper class objects
3. User-defined class objects

Collections class

Collections class provides static methods for sorting the elements of collections. If collection elements are of Set or Map, we can use TreeSet or TreeMap. However, we cannot sort the elements of List. Collections class provides methods for sorting the elements of List type elements.

Method of Collections class for sorting List elements

public void sort(List list): It is used to sort the elements of List. List elements must be of the Comparable type.

Note: String class and Wrapper classes implement the Comparable interface by default. So if you store the objects of string or wrapper classes in a list, set or map, it will be Comparable by default.

Comparator Interface in Java with Examples

A comparator interface is used to order the objects of user-defined classes. A comparator object is capable of comparing two objects of the same class**.** Following function compare obj1 with obj2.

Syntax:

```
public int compare(Object obj1, Object obj2):
```

Suppose we have an Array/ArrayList of our own class type, containing fields like roll no, name, address, DOB, etc, and we need to sort the array based on Roll no or name?

Method 1: One obvious approach is to write our own sort() function using one of the standard algorithms. This solution requires rewriting the whole sorting code for different criteria like Roll No. and Name.

Method 2: Using comparator interface- Comparator interface is used to order the objects of a user-defined class. This interface is present in java.util package and contains 2 methods compare(Object obj1, Object obj2) and equals(Object element).
Using a comparator, we can sort the elements based on data members. For instance, it may be on roll no, name, age, or anything else.

Method of Collections class for sorting List elements is used to sort the elements of List by the given comparator.

```
public void sort(List list, ComparatorClass c)
```

To sort a given List, ComparatorClass must implement a Comparator interface.

How do the sort() method of Collections class work?

Internally the Sort method does call Compare method of the classes it is sorting. To compare two elements, it asks “Which is greater?”

Compare method returns -1, 0, or 1 to say if it is less than, equal, or greater to the other. It uses this result to then determine if they should be swapped for their sort.

Properties Class in Java

The Properties class represents a persistent set of properties. The Properties can be saved to a stream or loaded from a stream.It belongs to **java.util** package. **Properties** define the following instance variable. This variable holds a default property list associated with a **Properties** object.

Properties defaults: *This variable holds a default property list associated with a Properties object.*

Features of Properties class:

- **Properties** is a subclass of Hashtable.
- It is used to maintain a list of values in which the key is a string and the value is also a string i.e; it can be used to store and retrieve string type data from the properties file.
- Properties class can specify other properties list as it's the default. If a particular key property is not present in the original Properties list, the default properties will be searched.
- Properties object does not require external synchronization and Multiple threads can share a single Properties object.
- Also, it can be used to retrieve the properties of the system.

Advantage of a Properties file

In the event that any data is changed from the properties record, you don't have to recompile the java class. It is utilized to store data that is to be changed habitually.

Note: The Properties class does not inherit the concept of a load factor from its superclass,

Hashtable Declaration

```
public class Properties extends Hashtable<Object,Object>
```

Constructors of Properties

1. Properties(): This creates a **Properties** object that has no default values.

Properties p = new Properties();

2. Properties(Properties propDefault): The second creates an object that uses **propDefault** for its default value.

Properties p = new Properties(Properties propDefault);

THANK YOU!!

I-TECH WORLD (AKTU)

JOIN TELEGRAM GROUP FOR HAND WRITTEN NOTES , QUANTUMS AND OTHER STUDY MATERIAL.

LINK GIVEN IN DESCRIPTION BOX