Collections Framework

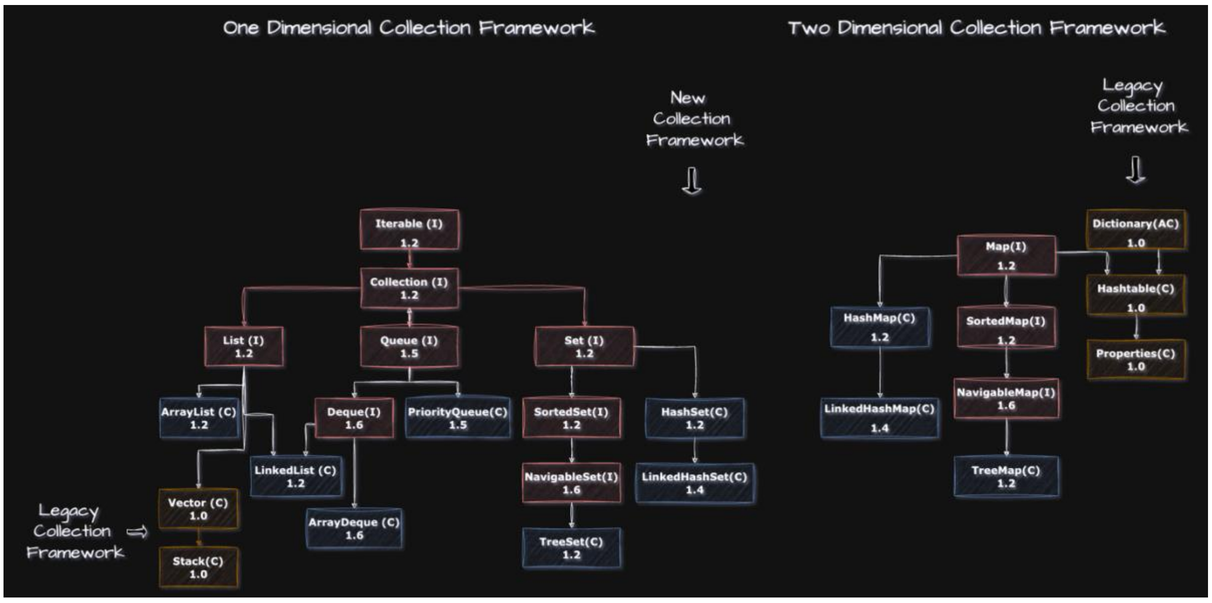
- The Collection Framework is an in-built facility in java provided by Sun Microsystems.
- By using Collection Framework, we can store different types of values either of same data type or different, in a single object.
- By using this, we can perform operations like adding elements, adding multiple elements at certain index, replacing elements, removing elements like either a particular element or all the elements at a time, we can perform searching and sorting operations easily by using predefined methods and by implementing less lines of code.
- It is one of the ways to store data, like database.
- In Collection Framework, we do not have any primitive data, we will have data in form of objects, and we only use wrapper classes in this framework.
- **Collection:** Collection is referred as a group of individual objects represented as a single entity. It is also considered as root interface in the framework.
- **Collections Framework:** Collection Framework defines several Classes and Interfaces which can be used to represent a Collection.

Differences Between Arrays and Collection

Arrays	Collection
Arrays are fixed in size.	Collection is growable in nature
We can store only homogeneous type of data in an array.	We can store both homogeneous and heterogeneous type of data in a Collection.
Arrays can deal with both primitive and Object type of data.	Collection deals with only Object type of data.
Arrays have poor memory management.	Collection has better memory management.
Arrays are better than a Collection in terms of Performance if size is known in advance.	Collection offers less performance than Arrays.
There is a limited predefined method support in arrays.	There is a huge library of predefined method support for Collection.

- Collection Framework is mainly classified into two types:
  - New Collection Framework (1.2 Version)
  - Legacy Collection Framework (1.0 Version)
- The classes that are present from the first version of Java are known as legacy classes (Stack, Vector, Dictionary, Hashtable, Properties)
- All these classes in Legacy Collection Framework are Thread-safe.
- Both New and Legacy Collection Framework are again sub classified into two types:
  - One-Dimensional Collection Framework.
  - Two-Dimensional Collection Framework.
- One-Dimensional Collection Framework stores the data in the form of values only whereas Two-Dimensional Collection Framework stores the data in terms of Key-Value Pairs.
- All predefined Classes and Interfaces of entire Collection Framework are available in a predefined package called “java.util”.
- The Interfaces in One-Dimensional Collection Framework are:
  - Iterable
  - Collection
  - List
  - Set
  - Queue
- The Interface in Two-Dimensional Collection Framework is:
  - Map

COLLECTION FRAMEWORK HIERARCHY



Cursors:

In Collection Framework, Cursor is used to iterate Objects, here we have three Cursors, they are:

- Iterator (I) also known as Universal Cursor
- ListIterator (I) Cursor only used for List Objects
- Enumeration (I) Legacy Cursor used for Legacy Classes.

Differences between Collection List Queue and Set

COLLECTION	LIST	QUEUE	SET
It is super interface for List, Queue, Set Interfaces.	It is a sub interface of Collection Interface	It is a sub interface of Collection Interface	It is a sub interface of Collection Interface
Here, we can store Duplicate Elements	Here, we can store Duplicate Elements	Here, we can store Duplicate Elements	Here, we cannot store Duplicate Elements
Here, there will be no order of Elements	Here, there will be order of Elements (Insertion Order)	Here, there will be no order of Elements	Here, there will be no order of Elements
Here, we can retrieve elements only in forward direction	Here, we can retrieve elements in both forward and backward direction	Here, we can retrieve elements in both forward and backward direction	Here, we can retrieve elements only in forward direction
Here, we can add elements only at the ending of the index	Here, we can add elements anywhere based on the requirement.	Here, we can add elements at either ending or starting of the Index.	Here, we can add elements only at the ending of the index.
Here, we can store null values.	Here, we can store null values.	Here, we cannot store null values.	Here we can store only one null value.

COLLECTION (1.2V):

- It is a Predefined Interface present in the “UTIL” package, and the fully qualified name of the Collection Interface is “java.util.Collection”
- Here, it doesn’t follow any order of elements and Collection interface will also allow duplicates and Null values.
- If you want to represent a group of individual objects as a single entity, then you should prefer Collection Interface.
- Collection Interface defines the most common methods which are applicable for any Collection Object.
- There is no concrete class which implements the Collection interface directly.
- Collection interface is introduced in Java version 1.2.

Methods of Collection Interface

- public int size() Returns the number of elements in this collection.
- public boolean isEmpty() Checks whether the collection contains any elements.
- public boolean add(Object) Adds an element at the end of the collection.
- public boolean addAll(Collection) Adds an entire collection to this collection
- public boolean remove(Object) Removes a specified element from this collection
- public boolean removeAll(Collection) Removes an entire collection from this collection
- public boolean equals(Object) Compares two objects for equality
- public void clear() removes all of the elements from this collection
- public Object[] toArray() Returns an array containing elements in this collection.
- public boolean contains(Object) checks if the collection contains a specified element
- public boolean containsAll(Collection) Checks if the collection contains an entire collection
- public Iterator iterator() Returns an iterator over the elements in this collection
- public int hashcode() Returns the hash code value for this collection
- public boolean retainAll(Collection) Retains only the elements in this collection that are contained in the specified collection

ITERATOR (1.2V):

- It is a Predefined Interface available in the “java.util.” package. It works as a Cursor.
- By using this, we can retrieve the values only in forward direction.
- By using this, we can also remove elemets from a collection.

Methods:

- public boolean hasNext()
- public Object next()
- public void remove()

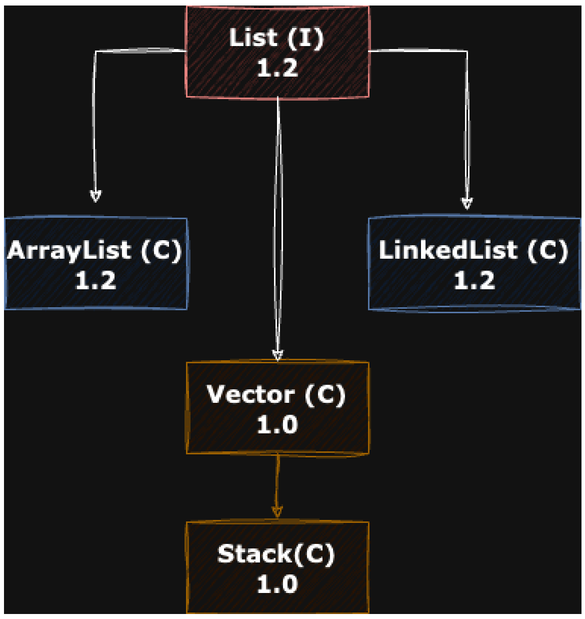
Example of Iterations through Iterator Interface:

```
Iterator itr = collection.iterator();
while(itr.hasNext()){
        System.out.println(itr.next());
}
```

LIST (1.2V) :

- It is a Predefined Interface present in “java.util” Package and the fully qualified name of the List is “java.util.List”.
- List will allow duplicate values and null values.
- List will follow interstion order.
- List is introduced in Java version 1.2
- List has implementation classes like ArrayList, LinkedList, Vector and Stack.=

Heirarchy:



Predefined Methods of List Interface

All the methods of Collection Interface are available by default in List interface. Along with those methods, List’s own implementation methods are:

- public void add(int, Object)
- public boolean addAll(int, Collection)
- public boolean remove(int)
- public Object set(int, Object)
- public Object get(int)
- public int indexof(Object)
- public int lastIndexOf (Object)
- public ListIterator listIterator()
- public Object clone()
- public List subList(int from-index, int to-index)

LISTITERATOR (1.2V):

- It is a Predefined Interface available in the “java.util.” package. It works as a Cursor specially made for List.
- By using this, we can retrieve the values in both forward and backward direction.
- By using this, we can also remove elemets from a collection.

Methods

- public boolean hasNext()
- public Object next()
- public boolean hasPrevious()
- public Object previous()
- public void remove()

Example of Iterations in forward direction through ListIterator Interface:

```
ListIterator litr = collection.listIterator();
while(litr.hasNext()){
        System.out.println(itr.next());
}
```

Example of Iterations in backward direction using ListIterator Interface:

```
while(litr.hasPrevious()){
        System.out.println(litr.previous());
}
```

ARRAYLIST(1.2V):

- It is a predefined class present in the util package and the fully qualified name of ArrayList is “java.util.ArrayList”.

- ArrayList will allow duplicate elements as well as Null values.
- Insertion order is preserved in an ArrayList.
- Size of an Array is fixed but ArrayList is growable in nature.
- ArrayList internally follows a dynamic Array to perform operations like adding, removing, replacing elements etc.,
- The default capacity of an ArrayList is 10
- It will take more amount of time to perform manipulation operations.
- ArrayList is a non-Synchronized class.

Profile of ArrayList:

Constructors:

- ArrayList()
- ArrayList(int)
- ArrayList(Collection)

Methods:

- public int size()
- public boolean isEmpty()
- public boolean add(Object)
- public void add(int, Object)
- public boolean add(Collection)
- public boolean addAll(int, Collection)
- public boolean remove(Object)
- public boolean remove(int)
- public boolean removeAll(Collection)
- public Object set(int, Object)
- public Object get(int)
- public boolean contains(Object)
- public boolean containsAll(Object)
- public boolean equals(Object)
- public void clear()
- public Object clone()
- public int lastIndexOf()
- public int indexOf()
- public Iterator iterator()
- public ListIterator listIterator()

- public List subList(int from-index, int to-index)
- public Object[] toArray()
- public int hashcode()
- public boolean retainAll(Collection)

ArrayList Questions

1. Create a Java Application where we need to create an ArrayList Object and perform the following operations
    1. Add five elements to the ArrayList Object
    2. Print the ArrayList
    3. Remove the element from the second index
    4. Add a new element in the second index
    5. Replace the third index with a new element.
    6. Print the ArrayList by using Iterator.
2. Create a Java Application where we need to create an ArrayList Object by using Integer as its Generic type. Add a few elements to it and print the list in forward and backward directions by using ListIterator.
3. Create a Java Application where we need to create an ArrayList Object by using Integer as its Generic type, add 100 elements to it, and print all the elements alternatively by using Iterator.
4. Create a Java Application where we need to create two ArrayList Objects by using Integer as its Generic type, add multiples of 2 to the first ArrayList and multiples of 3 to the second ArrayList. Display the values of ArrayList2 which are present in ArrayList1.
5. Create a Java Application where we need to create an ArrayList Object by using Integer as its Generic type, add a few elements to it, and print all the elements using the ForEach loop.
6. Create a Java Application where we need to create an ArrayList Object by using Integer as its Generic type, Add a few elements to it, then print all the values in ascending order.
7. Create a Java Application where we have one class Student, it contains private fields like student ID, student Name, and student Marks without initialization. Initialize these private fields by using both setter Injection and Constructor injection. Add the data of 5 students and then create an ArrayList object by using the Student class as its Generic type. Add the data of all students to this ArrayList and then display all of them.

Note: Create the ArrayList Object in a separate user defined class.

COMPARABLE

- Comparable is a predefined Interface present in lang package. Its fully qualified name is “java.lang.Comaparable” which consists of an abstract method like public abstract int compareTo(Object);
- It is not necessary to import Comparable Interface because that Interface is present in lang package and lang package is a default import to the compiler. And all wrapper classes inherit this comparable Interface and overrides compareTo(Object) method.
- Due to this, whenever we create an list object by keeping wrapper class as generic and when we use Collections.sort(List) method then automatically this method will sort the elements with the help of overridden compareTo(Object) method in that respective wrapper class.
- when we are dealing with user defined generics and if we want to sort the elements based on our requirements then we must inherit comparable interface by keeping generic as our respective user defined class and as it is an interface we should override the compareTo(Object) method and provide its functionality to get our elements sorted in the Collection Object.
- But by using Comparable Interface we can sort elements based on single column or single type of values like Age / Name. This is a drawback of using Comparable Interface, So as to overcome this drawback we need to use Comparator Interface for Sorting the elements based on multiple elements /columns.

COMPARATOR

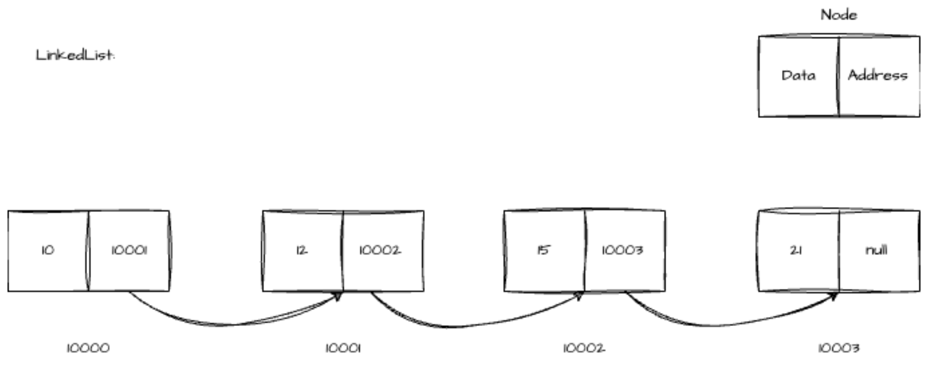
- It is a predefined Interface present in “ java.util “ package it consists of an abstract method like public abstract int compare(Object, Object)
- It is necessary to import Comparator Interface from “ java.util ” package.
- When we implement Comparator interface we must override compare method and should provide functionality to that method.
- When we deal with user defined classes then we need to mention generic of Comparator as User defined class name.
- After creating object of classes which are implementing Comparator interface, we need to give list object and Comparator object both as parameters to “Collections.sort (Object, Object)”. By using Comparator we can create multiple classes which are Implementing Comparator interface and can sort elements based on compare (Object, Object) method functionality.

Differences Between Comparable and Comparator

Comparable	Comparator
Comparable provides a single Sorting sequence. In other words, we can sort the collection based on a single element such as id, name, price.	The comparator provides Multiple Sorting Sequences. In other words, we can sort the collection based on multiple elements such as id, name, price etc.
Comparable affects the original class, i.e. the actual class is modified.	Comparator doesn't affect the original class i.e. the actual class is not modified.
Comparable provides "compareTo()" method to sort the elements.	Comparator provides "compare()" method to sort the elements.
Comparable is present in "java.lang" package.	Comparator is present in "java.util" package.
We can sort the list elements of Comparable type by Collections.sort(List) method.	We can sort the list elements of comparator type by Collections.sort(List, Comparator) method.

LINKEDLIST(1.2V):

- It is a Predefined class present in "util" package. And the fully qualified name of this implementation class of List interface is "java.util.LinkedList".
- It follows insertion order of elements
- Linkedlist internally uses a doubly linkedlist to store the data.
- LinkedList will store data interms of nodes. Each node holds the data and the address of the next node.
- LinkedList will allow both duplicate values and Null values.
- LinkedList is a non-synchronized class.



Profile of LinkedList:

Constructors:

- LinkedList()
- LinkedList(Collection)

Methods:

- public int size()
- public boolean isEmpty()
- public boolean add(Object)
- public void add(int, Object)
- public boolean add(Collection)
- public boolean addAll(int, Collection)
- public void push(Object)
- public boolean offer(Object)
- public boolean offerFirst(Object)
- public boolean offerLast(Object)
- public void addFirst(Object)
- public void addLast(Object)
- public boolean remove(Object)
- public boolean remove(int)
- public boolean removeAll(Collection)
- public Object pop()
- public Object pollFirst()
- public Object pollLast()
- public Object poll()
- public Object removeFirst()
- public Object removeLast()
- public Object removeFirstOccurance(Object)
- public Object removeLastOccurance(Object)
- public Object set(int, Object)
- public Object get(int)
- public Iterator iterator()
- public ListIterator listIterator()
- forEach(Lambda Expression)
- public Object peek()

- public Object peekFirst()
- public Object peekLast()
- public Object getFirst()
- public Object getLast()
- public Object clone()
- public int lastIndexOf()
- public int indexOf()
- public boolean contains(Object)
- public boolean containsAll(Object)
- public boolean equals(Object)
- public void clear()
- public List subList(int from-index, int to-index)
- public Object[] toArray()
- public int hashcode()
- public boolean retainAll(Collection)

Differences Between ArrayList and LinkedList

ArrayList	LinkedList
ArrayList Internally Follows a Dynamic Array	LinkedList Internally follows a datastructure called as Doubly Linked List
Insertion and deletion operations at the end of the list are effective, but insertion or removal of elements in the middle of the list requires subsequent shifting of elements.	It provides efficient insertion and removal of elements at both end of the list and middle of the list.
Time complexity of insertion or retrieving elements in the middle of the list takes O(n) time complexity due to shifting.	Time complexity of insertion or retriving elements at beginnning or end of the list are faster taking O(1).
It requires less memory space	It requires more memory space because a node stores references of both previous and next node addresses.
To retrieve elements, it takes O(1) time complexity	To retrieve elements it takes O(n) time complexity.

LinkedList Questions

1. Create a Java Application where we need to create a LinkedList Object by using Integer as its Generic type, add 5 elements to the LinkedList, remove the element at the second index of the list, add a new element at the second index of the list, replace the third index element with a new element and print the LinkedList by using ListIterator in both forward and backward directions.
2. Create a Java Application where we need to create a LinkedList Object by using Integer as its Generic type, add a few elements to it, remove the middle element, and print the LinkedList.
3. Create a Java Application where we need to create a LinkedList Object by using Integer as its Generic type, add a few elements to it, remove the first and last element of the LinkedList, and print the LinkedList.
4. Create a Java Application where we need to create a LinkedList Object by using Integer as its Generic type, add a few elements to it, and sort the elements without using the sort method.

<u>Vector(1.0V)</u>

- It is a Predefined class present in util package and the fully qualified name of this class is "java.util.Vector".
- This class was introduced in java version 1.0 and is also known as Legacy class.
- Vector is a synchronized class, meaning - it provides Thread safety operations and hence, performance will be compromised.
- Vector allows Null values and Duplicate elements to be stored in it.
- Vector follows Insertion order and random accessing of elements is possible.
- The default capacity of a Vector is 10 and whenever the number of elements in a vector reach the capacity, the vector simply doubles its capacity.

Constructors:

Vector()
Vector(int)
Vector(Collection)

Methods:

- public void addElement(Object)
- public void insertElementAt(Object, int)
- public void setElementAt(Object, int)
- public void removeElement(Object)
- public void removeElementAt(int)

- public void removeAllElements()
- public Object elementAt(int)
- public Enumeration elements()
- public Object firstElement()
- public Object lastElement()
- public void setSize(int)
- public int capacity()

And all methods from List Interface are available here in Vector to use.

Vector Questions

1. Create a Java Application where we need to create a Vector class Object by using Integer as its Generic type. Add few elements to it, replace the second element with a new element, add a new element at third index, then remove first and last element and then iterate through all the elements by using Legacy methods only.

Enumeration(1.0V)

- It is a Predefined Interface present in Util package, and the fully qualified name of this interface is "java.util.Enumeration".
- This interface was introduced in Java version 1.0 and hence we call this interface as Legacy Cursor which is used to iterate only legacy classes.
- like Iterator, Enumeration also iterates the Objects only in Forward direction. But unlike Iterator, we cannot remove elements.

Methods:

- public boolean hasMoreElements()
- public Object nextElement()

Stack(1.0V)

- Stack is a predefined class present in util package and the fully qualified name of this class is "java.util.Stack"
- It was introduced in java version 1.0 and hence it is also known as Legacy class.
- It is an Implementation class of List interface and a subclass of Vector
- Internally it follows dynamic array and follows LIFO principle.(Last In First Out)

- All the methods that are available in List interface and Vector class is available in this class but are not recommended to use.
- Stack allows duplicate values and null values.
- Stack follows Insertion order. it allows random access. But is not recommended to use.
- Stack's default capacity is 10 and upon reaching its maximum limit, stack doubles its capacity.
- As Vector is a Synchronized class, Stack, a Subclass of Vector is also a synchronized class. and due to this, performance is compromised.

Constructors:

Stack()

Methods:

- public void push(Object)
- public Object peek()
- public Object pop()
- public int search(Object)
- public boolean empty()

Stack Questions

1. Create a Java Application where we need to create a Stack Object by using String as its generic type, add few elements to it, and print all the elements using Stack specific Operations.
2. Input Output:

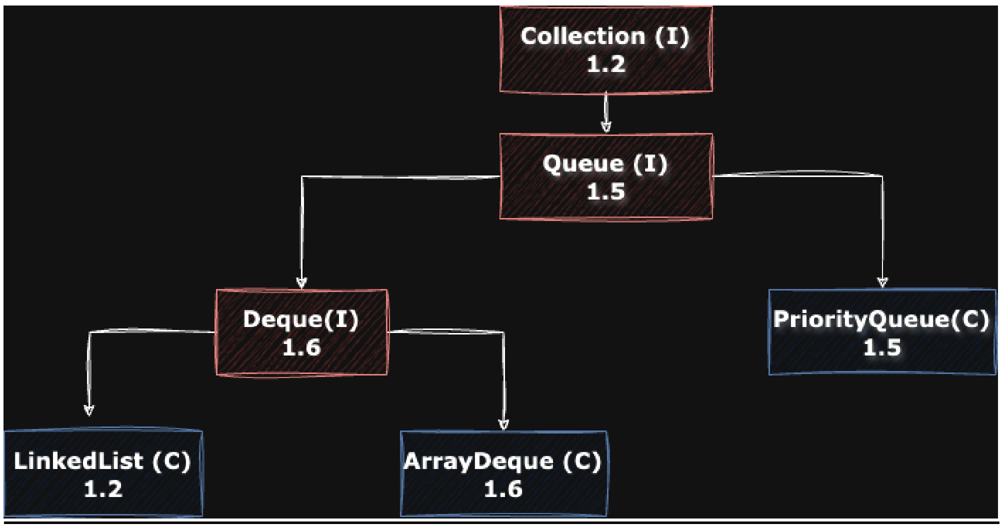
{[()]}	Valid
{}[]()	Valid
{[]()}	Valid
}[]{	invalid
{[(])}	invalid

<u>Queue(1.5V):</u>

- It is a predefined interface present in util package, and the fully qualified name of this interface is "java.util.Queue"
- It is a sub-interface of Collection which was introduces in java version 1.5
- Internally Queue follows dynamic array and the elements in queue are arranged according to FIFO principle(First In First Out) i.e., the elements will be added from the tail and retrieved from the head.

- Queue is by default a non synchronised i.e., it doesn't provide Thread safety.
- Queue allows duplicate values but it doesn't allow null values. Queue doesn't support random accessing of elements.
- But null values and random accessing of elements is possible when the implementation is provided by LinkedList.
- Its implementation classes are PriorityQueue, ArrayDeque and LinkedList.

<u>Heirarchy:</u>



Methods:

- Public boolean add(Object)
- public boolean offer(Object)
- public Object peek()
- public Object element()
- public Object remove()
- public Object poll()
- and all methods from Collection Interface are available to use here.

<u>PriorityQueue(1.5V):</u>

- It is a predefined class present in util package and the fully qualified name of this class is "java.util.PriorityQueue".
- It is the implementation class of Queue interface which was introduced in java version 1.5
- Internally this class uses a dynamic array to store and retrieve the elements and also follows Binary Tree algorithm to internally arrange elements.

- In PriorityQueue, the head of the Object is always prioritized to store the smallest value avaiable in the entire object.
- The remaining elements are arranged in random order.
- It is a Non-Synchronised class which doesn't provide thread safety.
- It allows duplicate values but doesn't allow null values to be stored in it.
- This class doesn't follow insertion order.
- The default capacity of PriorityQueue is 11 and upon reaching the capacity, it automatically increases its size as (initial capacity *2)+2

Constructors:

- PriorityQueue()
- PriorityQueue(int)
- PriorityQueue(Collection)
- PriorityQueue(Comparator)
- PriorityQueue(int, Comparator)

Methods:

- All the methods from Collection and Queue are available in this class to use.

PriorityQueue Questions:

1. Create a Java Application where we need to create a PriorityQueue Object with integer as its generic, add few elements to it and print the PriorityQueue.
2. Create a Java Application where we need to create a PriorityQueue Object with integer as its generic, add few elements to it and retrieve the values in ascending order.
3. Create a Java Application where we need to create a PriorityQueue Object with integer as its generic, add few elements to it and retrieve the values in descending order.
4. Create a Java Application where we need to create a PriorityQueue Object with userdefined class as its generic, add n objects to the Object and print the values in ascending order of integers.

Deque(1.6V):

- It is a Predefined Interface present in util package and the fully qualified name of this interface is "java.util.Deque" which was introduced in 1.6 version of Java

- it is a sub-interface of Queue. By using this interface, we can perform adding and retrieval of elements from both head and tail of the Object.
- By using this interface, we can satisfy both LIFO and FIFO principles.
- Its implementation classes are ArrayDeque and LinkedList.

Methods:

- public boolean add(Object)
- public boolean offer(Object)
- public void addFirst(Object)
- public void addLast(Object)
- public boolean offerFirst(Object)
- public boolean offerLast(Object)
- public Object element()
- public Object peek()
- public Object peekFirst()
- public Object peekLast()
- public Object getFirst()
- public Object getLast()
- public Object remove()
- public Object poll()
- public Object pollFirst()
- public Object pollLast()
- public Object removeFirst()
- public Object removeLast()
- public boolean isEmpty()
- All other methods that are present in Collection and Queue Interface are present in this interface.

ArrayDeque(1.6V):

- It is a predefined class present in util package and the fully qualified name of this class is "java.util.ArrayDeque".
- It is a Non-Synchronized class, meaning – it does not provide Thread safety.
- Internally ArrayDeque follows a dynamic Array whose initial capacity is 17 and upon reaching its maximum capacity, it increases its capacity as (currentCapacity*2)+2
- It allows duplicate values, but no null values are accepted.

- It follows Insertion Order but random accessing is not possible.

Constructors:

- ArrayDeque()
- ArrayDeque(int)
- ArrayDeque(Collection)

Methods:

All methods from Deque, Queue and Collection Interface are available in this class to use.

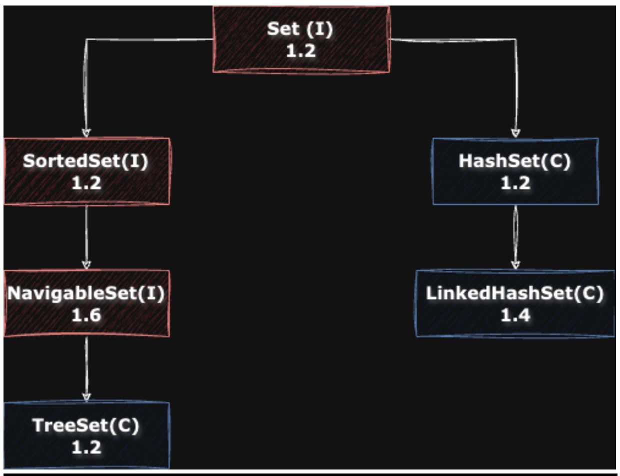
ArrayDeque Questions:

1. Create a Java Application where we need to create an ArrayDeque Object with Integer as its generic type, add few elements from head and tail to it, retrieve the elements from head and tail, Print the ArrayDeque and then remove the elements from head and tail and then iterate the Object to print the elements in it after all Operations.

Set(1.2V):

- It is a Predefined Interface present in util package and the fully qualified name of this interface is "java.util.Set".
- It is a Sub-Interface of Collection interface which was introduced in java version 1.2.
- This interface is used to store only unique type of Objecs i.e., No duplicates are allowed.
- It allows null values but only One.
- Set doesn't follow insertion order.
- Its implementation classes are HashSet, LinkedHashSet, TreeSet.
- Set doesn't provide any new methods i.e., we have to use whatever methods which are present in Collection interface.

Heirarchy:



HashSet(1.2V):

- It is a Predefined class present in util package and the fully qualified name of this class is "java.util.HashSet".
- It is the direct implementation class of Set interface which was introduced in java version 1.2
- It doesn't allow duplicate values.
- It allows null values but only one.
- It allows heterogeneous type of data.
- It doesnot follow insertion order because it follows hashing mechanism.
- Internally it follows Hashtable datastructure and based on hashing mechanism the objects will be stored in random order.
- It is a non synchronised class, meaning – it doesn't provide thread safety
- The default capacity of HashSet is 16
- Whenever the number of elements reach its 75% of the capacity, this Object automatically doubles its capacity.

Constructors:

- HashSet()
- HashSet(int)
- HashSet(int, float)
- HashSet(Collection)

Methods:

- All methods which are present in Collection interface are available in HashSet.

HashSet Questions

1. Create a Java Application where we need to create a Hashset Object (Rawtype), add a few elements to it, and print the HashSet.
2. Create a Java Application where need to create a HashSet Object with Integer as its Generic Type, add 1 - 10 Numbers to it and print the HashSet.
3. Create a java application where we need to create object of hashset class and add duplicate values and null values then display the elements using iterator.
4. Create a Java Application where we need to create two HashSet Objects, add a few elements to both the Objects and print the common elements from it.

LinkedHashSet(1.4V):

- It is a predefined class present in util package and its fully qualified name is "java.util.LinkedHashSet".

- It is a Subclass of HashSet which was introduced in 1.4 version of Java.
- It follows Insertion Order because along with Hashtable, LinkedHashSet also follows doubly Linked List datastructure.
- It is a Non-Synchronised class, meaning – it does not provide thread safety operations.
- It doesn't allow duplicate values but allows only one null value.
- Its default capacity is 16 and the load factor is 75%.

Constructors:

- LinkedHashSet()
- LinkedHashSet(int)
- LinkedHashSet(int, float)
- LinkedHashSet(Collection)

Methods:

All methods that are present in Collection interface are available in this class.

LinkedHashSet Questions

1. Create a java application where we need to create object for LinkedHashSet class and add only Integer type of values, Create second object for LinkedHashset add only Integer values including the values of previous object of LinkedHashSet then display all these values including the sum of all these values by using iterator.
2. Create a java application where we have an array of integer remove duplicate elements and preserve the order of the remaining elements using a LinkedHashSet.

SortedSet(I):

- It is a predefined Interface present in "java.util" Package.
- It is a Sub interface of Set interface which was introduced in the version 1.2.
- It follows "Sorted Order" i.e., the elements will be stored in ascending order by default.
- It doesn't allow duplicate values. it allows null values but only once when the object is empty.
- It doesn't allow heterogeneous type of data, if we try to store heterogeneous type of data, it throws ClassCastException
- Its implementation class is "TreeSet".

Methods:

- public Object first()
- public Object last()
- public SortedSet headSet(Object)
- public SortedSet tailSet(Object)
- public SortedSet subSet(Object, Object)

<u>NavigableSet(I):</u>

- It is a Predefined Interface present in Util Package.
- It is a Sub interface of SortedSet.
- It was introduced in the version 1.6 of Java
- It follows "Sorted Order" i.e., the elements will be stored in ascending order by default.
- It doesn't allow duplicate values, it allows null values but only once when the object is empty.
- It doesn't allow heterogeneous type of data, if we try to store heterogeneous type of data, it throws ClassCastException
- Its implementation class is "TreeSet".
- The main purpose of NavigableSet is to navigate the elements from head to tail of the Object. That is we can retrieve the elements in both forward and backward directions, we can iterate the elements in descending order or we can retrieve the descending set.

Methods:

- All Methods from SortedSet, Set and are available here in this Interface.
- Some extra methods are:
    - public Object pollFirst()
    - public Object pollLast()
    - public Iterator descendingIterator()
    - public Set descendingSet()
    - public Object higher(Object)
    - public Object lower(Object)

<u>TreeSet(C):</u>

- It is a Predefined class present in Util Package
- It was introduced in the version 1.2
- It is an implementation class of NavigableSet, SortedSet, Set, Collection

- It doesn’t allow duplicate elements and by default, elements will be arranged in ascending order.
- Internally, it follows Binary Search Tree data structure. We also call it as self balancing tree.
- It is a non-synchronized class, meaning it doesn’t provide thread safety operations.

Constructors:

- TreeSet()
- TreeSet(Comparator)
- TreeSet(Collection)
- TreeSet(SortedSet)

Methods:

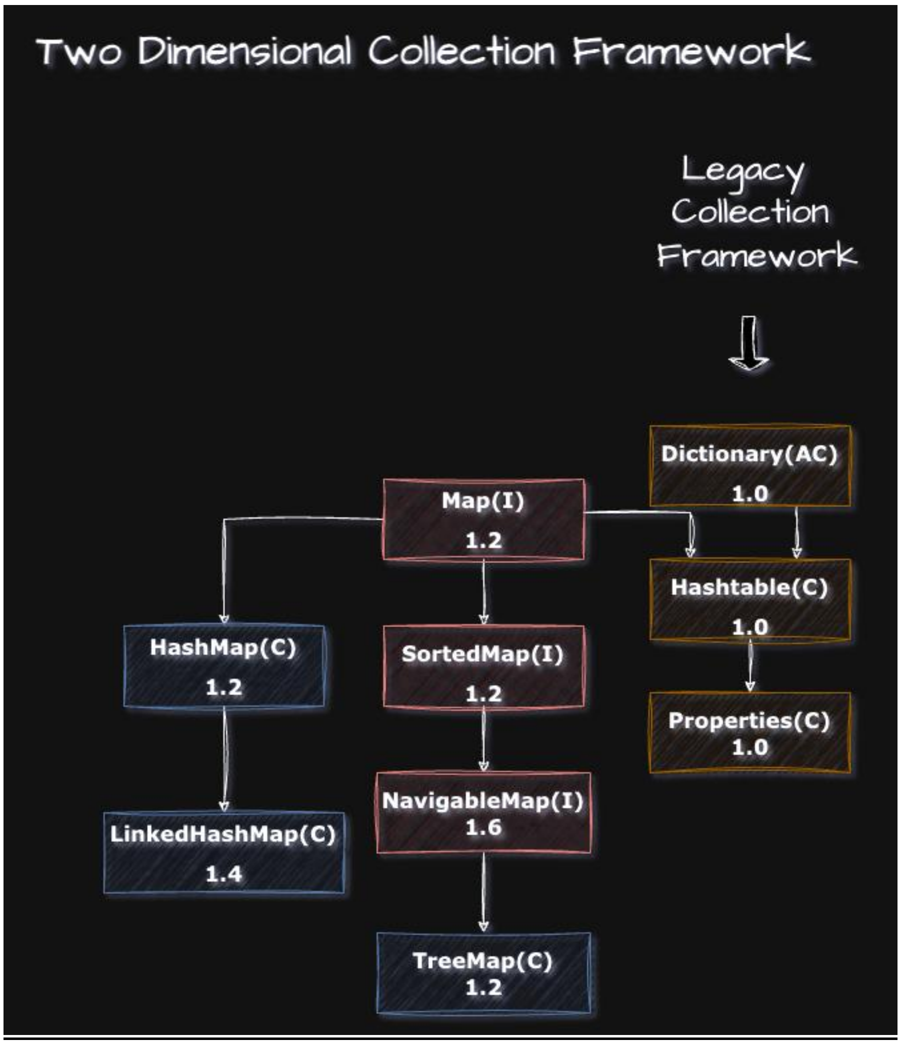
All the methods of NavigableSet, SortedSet, Set, Collection can be used here in TreeSet

TreeSet Questions

1. Create a Java Application where we need to create an object for TreeSet, add only integer type of values then display all these values in both ascending and descending order.
2. Create a Java Application Where we need to create a TreeSet Object and perform the following operations:
    1. Add a few elements to it.
    2. Retrieve the values which are less than a given value and greater than a given value.
    3. Retrieve the values that are in between the range of two specific values
    4. Retrieve the first and last elements
    5. Remove the first and last elements
    6. Convert the set to descending order and add two elements
    7. Print the set in descending order by using a descending iterator
    8. Print the value that is next greater than the given value
    9. Print the value that is next lesser than the given value.
3. Create a Java Application where we need to create an object of TreeSet and add multiple employee salaries to the object then display the values of Highest and Lowest salaries, then display the range of salaries, Then salaries which are less than the given value and the salaries which are greater than the given value.

2-D Collection Framework:

Heirarchy:



Map:

- It is a Predefined interface introduced in java version 1.2 which is present in util package. The fully qualified name of this interface is "java.util.Map".
- Whenever we want to store a group of individual objects in the form of key-value pairs, we will use Map.
- Both Keys and Values are objects and the combination of each and every key and value pair is called as an entry object.
- Hence, the group of entry objects can be considered as Map
- Without Map, we cannot create any entry object because Entry is an inner interface of Map.
- here, duplicate keys are not allowed but values can be duplicate and It doesn't allow random access.

- Its implementation classes are:
  - HashMap
  - LinkedHashMap
  - TreeMap
  - Hashtable
  - Properties

Methods:

- public Object put(Object key, Object value)
- public void putAll(Map)
- public void putIfAbsent(Object key, Object value)
- public Object get(Object key)
- public Object remove(Object key)
- public boolean remove(Object key, Object value)
- public Object replace(Object key, Object value)
- public int size()
- public void clear()
- public boolean isEmpty()
- public boolean equals(Object)
- public int hashcode()
- public Set entrySet()
- public Set keySet()
- public Collection values()
- public boolean containsKey(Object key)
- public boolean containsValue(Object value)

HashMap:

- It is a predefined implementation class of Map which was introduced in 1.2 version of java and is present in util package. Its fully qualified name is "java.util.HashMap"
- Internally, it follows hashtable datastructure to store data in the form of key-value pairs.
- The objects will be arranged in random order due to hashing mechanism
- It will allow heterogeneous type of data. But the keys should be unique and values can either be unique or duplicate.
- It is a non-synchronized class which doesn't provide thread safety.
- The default capacity is 16 and the load factor is 0.75

Constructors:

- HashMap()
- HashMap(int)
- HashMap(int, float)
- HashMap(Map Object)

Methods:

The methods which are present in Map interface are available in this class.

LinkedHashMap:

- it is a predefined implementation class of Map which was introduced in java version 1.4 and is present in util package, its fully qualified name is "java.util.LinkedHashMap".
- internally, it follows hashtable and doubly linkedlist datastructure to store the data in the form of key-value pairs.
- The objects will be arranged in insertion order due to hashing mechanism. It will allow heterogeneous data where duplicate keys are not allowed and duplicate values are allowed.
- It is a non synchronised class which doesn't provide any thread safety.
- its default capacity is 16 and the load factor is 0.75

Constructors:

- LinkedHashMap()
- LinkedHashMap(int)
- LinkedHashMap(int, float)
- LinkedHashMap(Map object)

Methods:

- All the methods from Map interface are available in this class.

<u>SortedMap</u>

- It is a sub interface of Map interface which is present in util package, its fully qualified name is "java.util.SortedMap".
- It was introduced in java version 1.2
- It will store the Objects in the form of key value pairs where duplicate keys are not allowed and duplicate values are allowed.
- The main purpose of SortedMap is to store key value pairs(Entry Objects) in sorted order ans by default ascending order.

Methods:

- public Object firstKey()
- public Object lastKey()
- public SortedMap headMap(Object key)
- public SortedMap tailMap(Object key)
- public SortedMap subMap(Object key, Object value)

NavigableMap

- It is a sub interface of Map interface which is present in util package, its fully qualified name is “java.util.NavigableMap”.
- It was introduced in java version 1.6
- It will store the Objects in the form of key value pairs where duplicate keys are not allowed and duplicate values are allowed.
- It doesn’t allowed null keys but null values are allowed.
- The main purpose of NavigableMap is it will provide flexibility like where we can retrieve the map object in descending order and we can remove the entry object from both ends and we can retrieve next higher value or next lower value.
- its implementation class is TreeMap

Methods:

- public Object pollFirstEntry()
- public Object pollLastEntry()
- public Object firstEntry()
- public Object lastEntry()
- public Object higherKey(Object key)
- public Object lowerKey(Object key)
- public Object higherEntry(Object key)
- public Object lowerEntry(Object key)
- public NavigableSet descendingKeySet()
- public NavigableMap descendingMap()

TreeMap:

- it is a Predefined implementation class of NavigableMap, SortedMap and Map which is present in java.util.TreeMap and is introduced in 1.2 version of java.

- It stored Objects in the form of key value pairs where duplicate keys and null keys are not allowed and duplicate values and null values are allowed.
- Internally it follows Binary Search Tree data structure which is also called as self balancing tree.
- It is a non synchronised class which doesn't provide thread safety.
- It will follow sorted order and doesn't allow heterogeneous type of data.

Constructors :

- TreeMap()
- TreeMap(Comparator)
- TreeMap(Map Object)
- TreeMap(SortedMap)

Methods:

All the methods from Map, SortedMap and NavigableMap interface are available in TreeMap.

Map Questions

1. Create a Java Application where we need to create an object of HashMap class add the values in the form of key-value pairs then display all keys and values.
2. Create a Java Application where we need to create a HashMap Object. Add a few elements and then check whether a given Key element is present in the map and then print the key-value pairs by using the entrySet method. And then print only keys.
3. Create a Java Application where we need to create an object of LinkedHashMap class add the values in the form of key-value pairs then create a second object of Linked HashMap add the values including the previous object of Linked HashMap then display only keys.
4. Create a Java Application where we need to create an object of the TreeMap class add the values then display all the values in both ascending and descending order.
5. Create a Java Application where we need to create an object for the TreeMap insert some values to it and then print the values using the Entry interface and then, get the values of the map which are less than a given key and get the higher key than a specific value.

Dictionary(AC):

- It is a Predefined abstract class which is present in util package and the fully qualified name of this class is "java.util.Dictionary".
- This class is introduced in 1.0 version of java.
- By using this class we can store the Objects in the form of key value pairs where keys cannot be duplicated or nullified and value can be duplicated or Nullified.
- By default all the methods which are present in this class are synchronized, hence dictionary is a synchronized class.
- Due to this, the Thread ssafety operations can be performed and hence, performance is compromised.
- Its implementation classes are "Hashtable" and "Properties".

Constructors:

- Dictionary()

Methods:

- public Enumeration elements()
- public Object get(Object key)
- public boolean isEmpty()
- public Enumeration keys()
- public void put(Object key, Object value)
- public void remove(Object key)
- public int size()

Hashtable:

- it is a predefined implementation class of Map and Dictionary which is present in util package and its fully qualified name is "java.util.Hashtable".
- it was introduced in java version 1.0 and hence also called as Legacy class.
- By using this class we can store the Objects in the form of key value pairs where keys cannot be duplicated or nullified and value can be duplicated or Nullified.
- It is a synchronized class and provides thread safety due to which performance is compromised.
- Internally it follows hashtable datastructure and all objects are arranged in a random order due to hashing mechanism.
- The default capacity of this class is 11 and the load factor is 0.75

Constructors:

- Hashtable()
- Hashtable(int)
- Hashtable(int, float)
- Hashtable(Map Object)

Methods:

All the methods from Map and Dictionary are available in this class for use.

<u>Properties:</u>

- It is a predefined class present in util package and the fully qualified name of this class is "java.util.Properties".
- It was introduced in java version 1.0 and is also called as legacy class.
- It is by default synchronized which provided thread safety operations due to which performance is compromised.
- By using this class we can store the Objects in the form of key value pairs where keys cannot be duplicated or nullified and value can be duplicated or Nullified.
- In addition to this the main advantage of this class is to store data in an external file. This external file is created by using the extensions like .rbf or .properties
- Whenever we want to perform any frequent changes or updations, we can directly make those changes in the properties file instead of perdoeming changes in source code.
- Due to this, we can overcome repearted recompilation of our source code.
- It doesn't allow heterogeneous data so whatever keys or values we want to add to the properties class should be a String type of value only.

Constructors:

- Properties()

Methods:

- public void setProperty(String key, String value)
- public void getProperty(String key)
- public void load(InputStream Object)
- public void store(OutputStream Object, String comments)
- public Enumeration propertyNames()
- and all other methods from Map and Dictionary are avaiable here to use.