PROGRAM
ON
OBJECT ORIENTED PROGRAMMING WITH JAVA

UNIT-IV
JAVA COLLECTION FRAMEWORK

BY

DR. BIRENDRA KR. SARASWAT

GLBITM

GREATER NOIDA

Email : birendra.saraswat@glbitm.ac.in Phone: +91-9999600474

Unit 4

IV	**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.

Collection in Java

- In Java, a Collection represents a group of objects, offering a structured way to store and manipulate them.
- Collection: A group of individual objects that represent a single entity is known as a collection. It is the common word that you used in your daily life. But if we are discussing Java programming language then it will become Java Collection Framework.
- To represent a group of objects as a single entity in the Java programming language we need classes and interfaces defined by the Collection Framework.

Cont.

- Collection Interface: Interfaces specify what a class must do and not how. It is the blueprint of the class. It is the root interface of the Collection Framework that defines the most common methods that can be used for any collection objects. Or you can say it represents the individual object as a single entity.
- Collections Class: It is present in java.util package and is a member of Collection Framework. This class provides many utility methods for the collection object.

Java Collections Framework

- The Java collections framework provides a set of interfaces and classes to implement various data structures and algorithms.

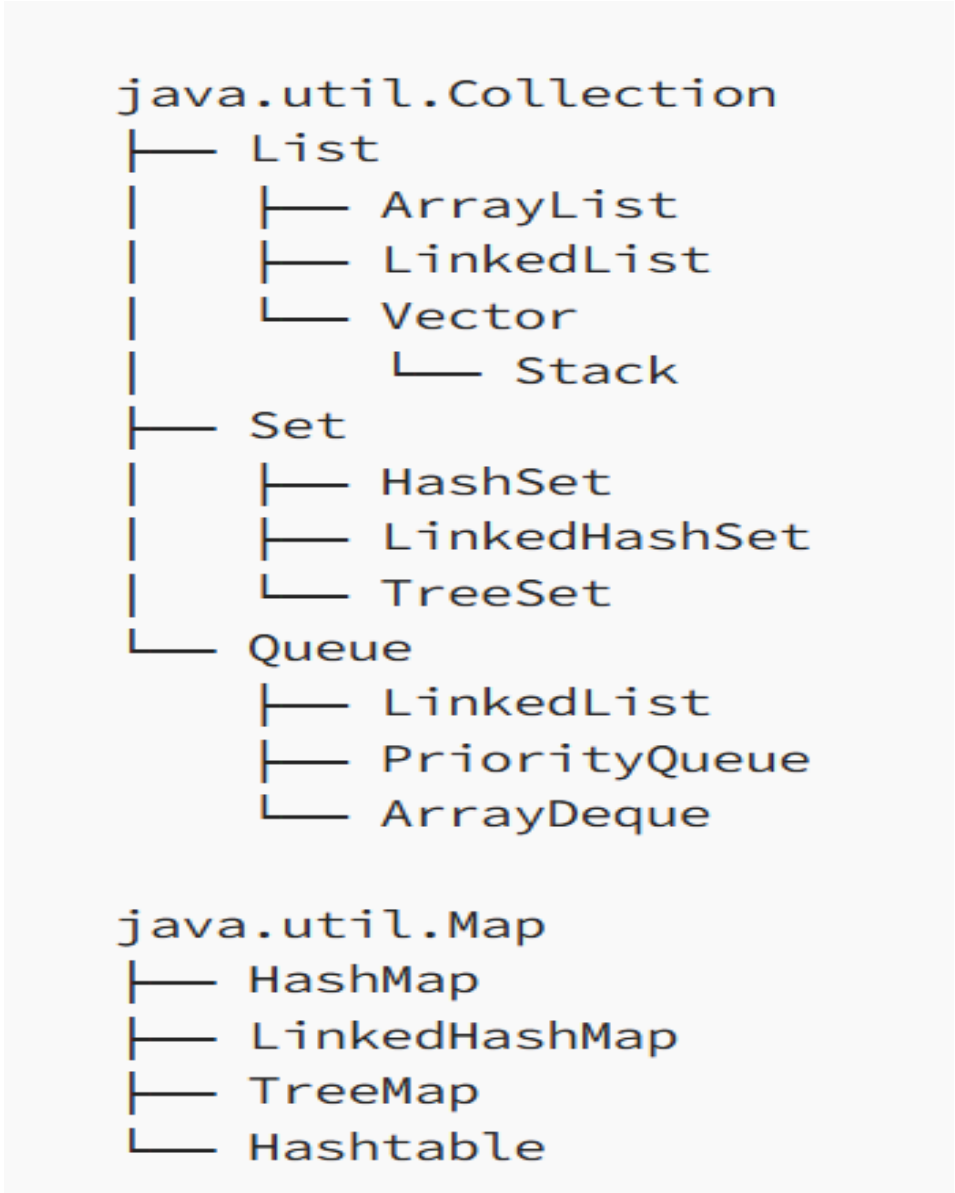
- For example, the LinkedList class of the collections framework provides the implementation of the doubly-linked list data structure.

- The Java Collections Framework consists of several core interfaces such as Collection, List, Set, Queue, Deque, Map, and their corresponding implementations.

Cont…

- All these collections can be imported using:
- import java.util.*;
- However, single classes can also be imported by replacing * with the class name as shown
- import java.util.ArrayList;
- import java.util.LinkedList;

Hierarchical Structure of Java Collections Framework



Core Interfaces and Their Functions

- **Collection:** The root interface of the collection hierarchy. It defines the basic operations that all collections must implement, such as add, remove, size, and iterator.
- **List:** An ordered collection (also known as a sequence). Lists can contain duplicate elements. Examples include ArrayList, LinkedList, and Vector.

List Interface

- It 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 deals with the index or position-specific functions like getting an element or setting an element.
- It deals with the arrays and lists types of operations like ArrayList, LinkedList, Vector, and Stack.

Cont..

- 1. ArrayList provides us with dynamic arrays in Java. The size of an ArrayList is increased automatically if the collection grows or shrinks if the objects are removed from the collection.

- 2. LinkedList is class is an implementation of a doubly-linked list data structure.

3. Vector

- It provides us with dynamic arrays in Java.
- This is a legacy class. It is a thread-safe class.
- This is not recommended being used in a single-threaded environment as it might cause extra overheads.
- However, to overcome this in Vectors place one can readily use ArrayList.

4. Stack

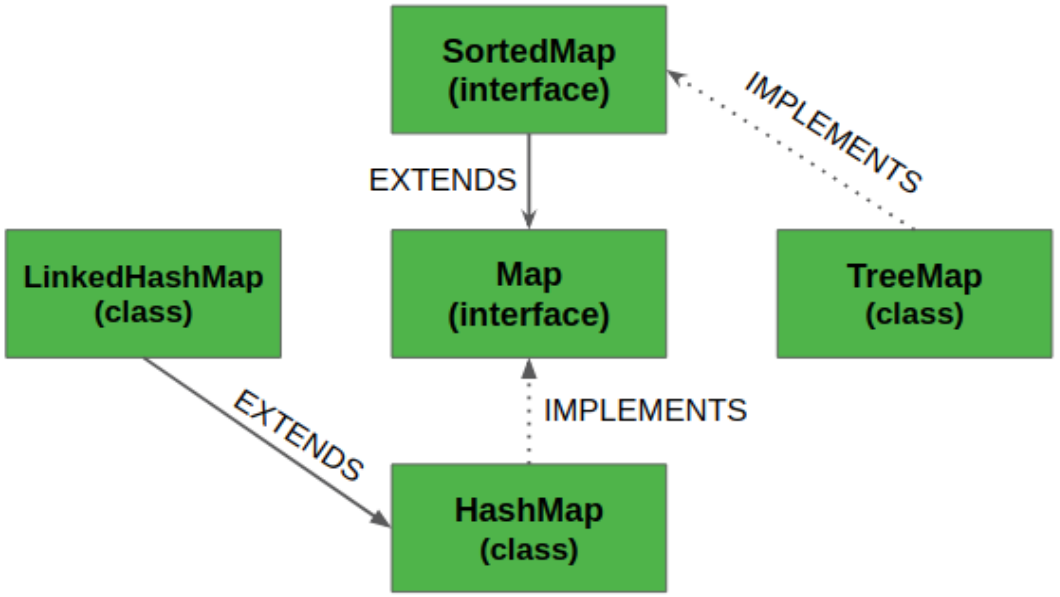
- 4. Stack is a class is based on the basic principle of last-in-first-out. This is a legacy class. This inherits from a Vector class. It is also a thread-safe class. This is not recommended being used in a single-threaded environment as it might cause extra overheads. However, to overcome this in Vectors place one can readily use ArrayDeque.

Core Interfaces and Their Functions

- **Set:** A collection that cannot contain duplicate elements. It models the mathematical set abstraction. Examples include HashSet, LinkedHashSet, and TreeSet.
- **Queue:** A collection used to hold multiple elements prior to processing. Typically, they order elements in a FIFO (First-In-First-Out) manner. Examples include LinkedList, PriorityQueue, and ArrayDeque.

Core Interfaces and Their Functions

- Deque: A linear collection that supports element insertion and removal at both ends. Example includes ArrayDeque.
- Map: An object that maps keys to values. A map cannot contain duplicate keys; each key can map to at most one value. Examples include HashMap, LinkedHashMap, TreeMap, and Hashtable.



MAP Hierarchy in Java

Code Examples of java.util.Collection package

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

public class ArrayListExample {
    public static void main(String[] args) {
        List<String> arrayList = new ArrayList<>();
        arrayList.add("Apple");
        arrayList.add("Banana");
        arrayList.add("Cherry");

        // Enhanced for loop
        for (String fruit : arrayList) {
            System.out.println(fruit);
        }
```

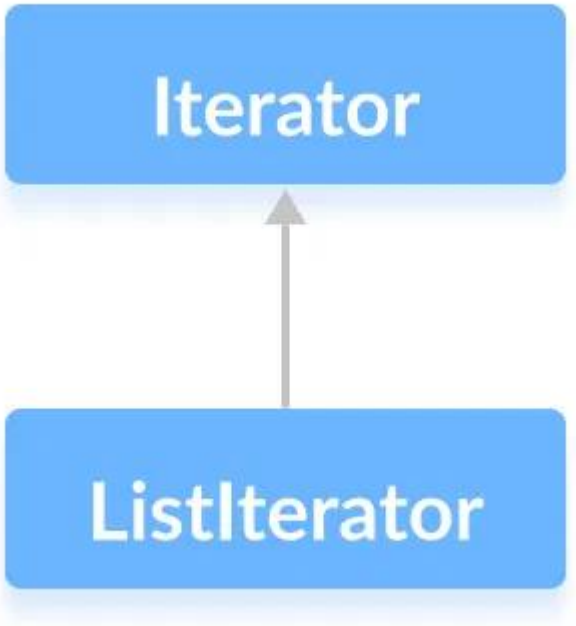
```
        // Iterator

        Iterator<String> iterator =
arrayList.iterator();
        while (iterator.hasNext()) {
            System.out.println(iterator.next());
        }
    }
}
```

•

Iterator Interface

- The Iterator interface of the Java collections framework allows us to access elements of a collection.
- It has a subinterface ListIterator.



Methods of Iterator

- The Iterator interface provides 4 methods that can be used to perform various operations on elements of collections.

- hasNext() - returns true if there exists an element in the collection
- next() - returns the next element of the collection
- remove() - removes the last element returned by the next()
- forEachRemaining() - performs the specified action for each remaining element of the collection

Ex

```
import java.util.ArrayList;
import java.util.Iterator;
class Main
{
        public static void main(String[] args)
        {
    // Creating an ArrayList
    ArrayList<Integer> numbers = new ArrayList<>();
    numbers.add(1);
    numbers.add(3);
   numbers.add(2);
    System.out.println("ArrayList: " + numbers);
    // Creating an instance of Iterator
    Iterator<Integer> iterate = numbers.iterator();
   // Using the next() method
   int number = iterate.next();
    System.out.println("Accessed Element: " + number);
    // Using the remove() method
        iterate.remove();
        System.out.println("Removed Element: " + number);

        System.out.print("Updated ArrayList: ");

        // Using the hasNext() method
        while(iterate.hasNext()) {
           // Using the forEachRemaining() method
           iterate.forEachRemaining((value) -> System.out.print(value + ", "));
        }
    }
}
```

Output

```
ArrayList: [1, 3, 2]
Acessed Element: 1
Removed Element: 1
Updated ArrayList: 3, 2,
```

Comparison	List	Array
Definition	List is an Interface in Java that extends collection. The List has multiple implementations ArrayList is one of the common implementations of List	Arrays organize items in a sequential manner in memory. Arrays are immutable which means they are unchangeable, and it allows duplicate elements
Static/ Dynamic	List holds objects with initial capacity and dynamically grows once it gets filled. The list is dynamic in nature	An array is a container object that can hold the static size of objects of the same data type. An array is static in nature
Storage	List can store only objects.	An array can store both primitives values and also objects.
Length	To find the length of the List we use size() method	To find the length of an array we use length() method
Initialization	An ArrayList instance can be created without specifying its size. Java creates its default size.	while initializing the array, it is mandatory to mention its size.
Performance	List usually takes more time than array as it resizes itself and this operation slows its performance.	Array performance is fast because of its fixed size.

Features	Array	ArrayList
Data structure	Fixed-length	Variable-length
Length	Cannot be changed	Can be resized
Data types	Can hold both primitives and objects	Can only hold objects
Accessing elements	Using [ ] operator	Using a set of methods
Memory allocation	Contiguous memory locations	Non-contiguous memory locations

	ArrayList	LinkedList
Structure	ArrayList is an index based data structure where each element is associated with an index.	Elements in the LinkedList are called as nodes, where each node consists of three things – Reference to previous element, Actual value of the element and Reference to next element.
Insertion And Removal	Insertions and Removals in the middle of the ArrayList are very slow. Because after each insertion and removal, elements need to be shifted.	Insertions and Removals from any position in the LinkedList are faster than the ArrayList. Because there is no need to shift the elements after every insertion and removal. Only references of previous and next elements are to be changed.
Retrieval (Searching or getting an element)	Insertion and removal operations in ArrayList are of order O(n). Retrieval of elements in the ArrayList is faster than the LinkedList . Because all elements in ArrayList are index based.	Insertion and removal in LinkedList are of order O(1). Retrieval of elements in LinkedList is very slow compared to ArrayList. Because to retrieve an element, you have to traverse from beginning or end (Whichever is closer to that element) to reach that element.
Random Access	Retrieval operation in ArrayList is of order of O(1). ArrayList is of type Random Access. i.e elements can be accessed randomly.	Retrieval operation in LinkedList is of order of O(n). LinkedList is not of type Random Access. i.e elements can not be accessed randomly. you have to traverse from beginning or end to reach a particular element.
Usage	ArrayList can not be used as a Stack or Queue.	LinkedList, once defined, can be used as ArrayList, Stack, Queue, Singly Linked List and Doubly Linked List.
Memory Occupation	ArrayList requires less memory compared to LinkedList. Because ArrayList holds only actual data and it's index.	LinkedList requires more memory compared to ArrayList. Because, each node in LinkedList holds data and reference to next and previous elements.
When To Use	If your application does more retrieval than the insertions and deletions, then use ArravList.	If your application does more insertions and deletions than the retrieval, then use LinkedList.

ArrayList	Vector
ArrayList is not synchronized.	Vector is synchronized.
Since ArrayList is not synchronized. Hence, its operation is faster as compared to vector.	Vector is slower than ArrayList.
ArrayList was introduced in JDK 2.0.	Vector was introduced in JDK 1.0.
ArrayList is created with an initial capacity of 10. Its size is increased by 50%.	Vector is created with an initial capacity of 10 but its size is increased by 100%.
In the ArrayList, Enumeration is fail-fast. Any modification in ArrayList during the iteration using Enumeration will throw ConcurrentModificatioException.	Enumeration is fail-safe in the vector. Any modification during the iteration using Enumeration will not throw any exception.

POINTS	Array List	Linked List	Vector
Data Structure	Dynamic array	Doubly linked list	Dynamic array
Random Access	Efficient (O(1))	Inefficient (O(n)))	Efficient (O(1)))
Default Capacity	10	Not Applicable	10
Capacity Adjustment	Resizing operation when full	Not Applicable	Resizing operation when full
Thread Safety	Not Thread Safe	Not Thread Safe	**(Synchronized)** Thread Safe
Memory Wastage	If capacity exceeds element count	Not Applicable	If capacity exceeds element count
Insertion/Deletion at Arbitrary Positions	Costly (Shift elements)	Efficient (Update node references)	Costly (Shift elements)
Performance	Faster for random access	Faster for insertion/deletion	Slightly slower than ArrayList

	ArrayList	LinkedList	Vector	Stack
Duplicates	Allow	Allow	Allow	Allow
Order	Insertion order	Insertion order	Insertion order	Insertion order
Insert / Delete	Slow	Fast	Slow	Slow
Accessibility	Radom and fast	Sequential and slow	Random and fast	Slow
Traverse	Uses Iterator / ListIterator	Uses Iterator / ListIterator	Enumeration	Enumeration
Synchronization	No	No	Yes	Yes
Increment size	50%	No initial size	100%	100%

TreeSet, LinkedHashSet, and HashSet

- TreeSet, LinkedHashSet, and HashSet in Java are three Set implementations in the collection framework and like many others they are also used to store objects.
- The main feature of TreeSet is sorting,
- LinkedHashSet is insertion order and
- HashSet is just general purpose collection for storing objects.

- HashSet is implemented using HashMap in Java
- while TreeSet is implemented using TreeMap.
- TreeSet is a SortedSet implementation that allows it to keep elements in the sorted order defined by either Comparable or Comparator interface.

	TreeSet	LinkedHashSet	HashSet
Ordering	Sorted in ascending order	Maintains insertion order	No defined ordering
Underlying Data Structure	Balanced Binary Search Tree (Red-Black Tree)	Doubly Linked List + Hash Table	Hash Table
Duplicates	Does not allow duplicate elements	Does not allow duplicate elements	Does not allow duplicate elements
Performance	Slower performance for insertion, deletion, and retrieval compared to HashSet	Slightly slower performance compared to HashSet	Fastest performance for insertion, deletion, and retrieval
Iteration Order	Sorted based on natural ordering or custom comparator	Maintains the insertion order	No defined iteration order
Null Values	Does not allow null values	Allows a single null value	Allows multiple null values
Usage	When elements need to be sorted in a specific order	When elements need to be maintained in insertion order	When elements need to be stored without any ordering

HashMap Class

- It is used for storing and processing items in key, value pair.
- Syntax:
- HashMap<keyDataType,ValueDataType> objectname=new HashMap()

- Methods:
- V put(object key, object value): used to insert an entry in map
- V get(object key): returns the object that contains the value associated to key
- int size()
- Set entrySet()
- etc

Ex:

```
import java.util.HashMap;

public class Main
{
   public static void main(String[] args)
    {
         HashMap<Integer, String> mp = new HashMap<>();
         mp.put(1, "Apple");
          mp.put(2, "Banana");
          mp.put(3, "Cherry");

         System.out.println(mp)
 }
```

- .

Ex2

```
import java.util.HashMap;
import java.util.Map;
public class Main
{
   public static void main(String[] args)
      {
            HashMap<Integer, String> mp = new HashMap<>();
            mp.put(1, "Apple");
             mp.put(2, "Banana");
             mp.put(3, "Cherry");
            for(Map.Entry item:mp.entrySet())
           {
                 System.out.println(item.getKey()+”\t”+item.getValue());
                 }
                 }
                 }
```