Collection framework
collection interface

```
// Using the Collection interface with an ArrayList
import java.util.*;

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

        // Create a Collection using ArrayList
        Collection<String> c = new ArrayList<String>();

        // Adding elements to the collection
        c.add("Apple");
        c.add("Banana");
        c.add("Orange");

        System.out.println("Collection: " + c);
    }
}
```

Output
Collection: [Apple, Banana, Orange]

```
//Set Interface

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

```
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;
public class JavaIteratorExample1 {
	public static void main(String[] args)
	{
		// create a list
		List<String> list = new LinkedList<>();
		list.add("Welcome");
		list.add("to");
		list.add("GFG");

		System.out.println("The list is given as : "+ list);

		// get the iterator on the list
		Iterator<String> itr = list.iterator();

		// Returns true if there are more number of elements.
		while (itr.hasNext()) {
			// Returns the next element.
			System.out.println(itr.next());
```

```
                }

                // Removes the last element.
                itr.remove();
                System.out.println("After the remove() method is called : "+ list);
        }
}


}
}
```

TreeSet is one of the most important implementations of the SortedSet interface in Java that uses a Tree(red – black 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.

TreeSet does not allow duplicate elements. Any attempt to add a duplicate element will be ignored.
It doesn’t allow null values and throws NullPointerException null element is inserted in it.
TreeSet implements the NavigableSet interface and provides additional methods to navigate the set (e.g., higher(), lower(), ceiling(), and floor()).
It is not thread safe. For concurrent access, it should be synchronized externally using Collections.synchronizedSet().

Iterator Interface
// Java program to show the usage of Iterator()
Output
The list is given as : [Welcome, to, GFG]
Welcome
to
GFG
After the remove() method is called : [Welcome, to]

Develop Custom Class Iterator
To provide similar functionality for user-defined /custom class, we should follow the below steps:
Define a custom class.
Define the collection class to this custom class.
The collection class should import java.util package and implement iterable interface.
This collection class should now provide implementation to Iterable interface's method iterator().

```
import java.util.*;
import java.io.*;
class Employees implements Iterable {
	List<String> str = null;
	public Employees()
	{
		str = new ArrayList<String>();
		str.add("practice");
		str.add("geeks");
		str.add("for");
		str.add("geeks");
		str.add("to");
		str.add("learn");
		str.add("coding");
	}

	@Override public Iterator<String> iterator()
	{
		return str.iterator();
	}
}
public class EmployeesTester {
	public static void main(String[] args)
	{
		Employees emps = new Employees();
		for (String st : emps.str) {
			System.out.println(st);
		}
	}
}
```

Output

```
practice
geeks
for
geeks
to
```

learn
coding

List interface

```
package JAVAExamples;

import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;
import java.util.ListIterator;

public class InterfaceList {

 public static void main(String[] args) {
  // Add elements in ArrayList.
  List al = new ArrayList();
  al.add("New York");
  al.add("Delhi");
  al.add("Tokyo");

  // Print arraylist elements.
  System.out.println("ArrayList Elements");
  System.out.print(al);
  System.out.println();

  // Add elements in linkedlist.
  List ll = new LinkedList();
  ll.add("New York");
  ll.add("Delhi");
  ll.add("Tokyo");

  // Print linkedlist elements.
  System.out.println();
  System.out.println("LinkedList Elements");
  System.out.print(ll);
  System.out.println();

  // Get and print arraylist's 3rd element.
  System.out.println();
  System.out.println("Element at 3rd position in arralist is : " + al.get(2));

  // Using ListIterator to traverse through forward and backward directions in arralist.
  ListIterator<String> itrtr = al.listIterator();

  // Traverse in forward direction.
  System.out.println();
  System.out.println("Traversing through arralist elements in forward direction...");
  while (itrtr.hasNext()) {
   System.out.println(itrtr.next());
```

```
  }

  // Traverse in backward direction.
  System.out.println();
  System.out.println("Traversing through arralist elements in backward direction...");
  while (itrtr.hasPrevious()) {
   System.out.println(itrtr.previous());
  }
 }
}
```

Arryalist

```
import java.util.*;
// Define a class to demonstrate the usage of collections.
public class CollectionsFrameworkDemo {
    // Define a static method to showcase various collection usages.
    static void showcaseCollectionsUsage() {
        // Initialize an array of integers.
        int[] numbersArray = new int[]{1, 2, 3, 4};
        // Create an ArrayList to hold Integer objects.
        List<Integer> numbersList = new ArrayList<>();
        // Create a HashMap to map Integer keys to String values.
        Map<Integer, String> numbersMap = new HashMap<>();
        // Add elements to the ArrayList.
        numbersList.add(1);
        numbersList.add(2);
        // Put key-value pairs into the HashMap.
        numbersMap.put(1, "alpha");
        numbersMap.put(2, "beta");
        // Print the first element of the array.
        System.out.println("First element of numbersArray: " + numbersArray[0]);
        // Print the first element of the ArrayList.
        System.out.println("First element of numbersList: " + numbersList.get(0));
        // Print the value associated with key 1 in the HashMap.
        System.out.println("Value for key 1 in numbersMap: " + numbersMap.get(1));
        // Header for iterating over the array.
        System.out.println("\nIterating over numbersArray:");
        // Iterate through the array and print each element.
        for (int num : numbersArray) {
            System.out.println("Element: " + num);
        }
        // Header for iterating over the ArrayList.
        System.out.println("\nIterating over numbersList:");
        // Iterate through the ArrayList and print each element.
        for (Integer num : numbersList) {
            System.out.println("Element: " + num);
        }
        // Header for iterating over the HashMap.
        System.out.println("\nIterating over numbersMap:");
        // Iterate through the HashMap and print each key-value pair.
        for (Map.Entry<Integer, String> entry : numbersMap.entrySet()) {
            System.out.println("Key: " + entry.getKey() + ", Value: " + entry.getValue());
        }
    }
    // The main method to run the showcaseCollectionsUsage method.
    public static void main(String[] args) {
        // Call the showcase method to demonstrate collection usages.
```

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

Arraylist using iterator

```
import java.io.*;
import java.util.*;

public class JavaIteratorExample {
   public static void main(String[] args)
   {
      ArrayList<String> cityNames = new ArrayList<String>();

      cityNames.add("Delhi");
      cityNames.add("Mumbai");
      cityNames.add("Kolkata");
      cityNames.add("Chandigarh");
      cityNames.add("Noida");

      // Iterator to iterate the cityNames
      Iterator iterator = cityNames.iterator();

      System.out.println("CityNames elements : ");

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

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

Output:

```
CityNames elements:
Delhi Mumbai Kolkata Chandigarh Noida
```

ArrayListExample1.java

```
import java.util.*;
 public class ArrayListExample1{
 public static void main(String args[]){
  ArrayList<String> list=new ArrayList<String>();//Creating arraylist
      list.add("Mango");//Adding object in arraylist
      list.add("Apple");
      list.add("Banana");
      list.add("Grapes");
      //Printing the arraylist object
      System.out.println(list);
 }
}
```

Linked list

```
// Java Program to Demonstrate Implementation of LinkedList class
import java.util.*;
public class GFG
{
    public static void main(String args[])
    {
        // Creating object of the class linked list
        LinkedList<String> ll = new LinkedList<String>();
        // Adding elements to the linked list
        ll.add("A");
        ll.add("B");
        ll.addLast("C");
        ll.addFirst("D");
        ll.add(2, "E");
        System.out.println(ll);
        ll.remove("B");
        ll.remove(3);
        ll.removeFirst();
        ll.removeLast();
        System.out.println(ll);
    }
}
```

```
// Java program to Demonstrate Working of a LinkedList Importing required classes
import java.util.*;
// Main class
class GFG {
	// main driver method
	public static void main(String args[])
	{
		// Creating an object of the class linked list
LinkedList<String> object = new LinkedList<String>();
// Adding the elements to the object created using add() and addLast() method Custom
input elements
		object.add("A");
		object.add("B");
		object.addLast("C");

		// Print the current LinkedList
		System.out.println(object);

// Removing elements from the List object using remove() and removeFirst() method
		object.remove("B");
		object.removeFirst();

		System.out.println("Linked list after "+ "deletion: " + object);
	}
}
```

Hashset

```
// Java program to show the use of HashSet
import java.util.*;

class GFG
{
    public static void main(String[] args)
    {
        // Instantiate an object of HashSet
        HashSet<Integer> hs = new HashSet<>();

          // Adding elements
        hs.add(1);
        hs.add(2);
        hs.add(1);

        // Printing the Size and Element of HashSet
        System.out.println("HashSet Size: " + hs.size());
        System.out.println("Elements in HashSet: " + hs);
    }
}
```

Output

```
HashSet Size: 2
Elements in HashSet: [1, 2]
```

HashSetExample.java

```
import java.util.*;
class Book {
int id;
String name,author,publisher;
int quantity;
public Book(int id, String name, String author, String publisher, int quantity) {
    this.id = id;
    this.name = name;
    this.author = author;
    this.publisher = publisher;
    this.quantity = quantity;
}
}
public class HashSetExample {
public static void main(String[] args) {
    HashSet<Book> set=new HashSet<Book>();
    //Creating Books
    Book b1=new Book(101,"Let us C","Yashwant Kanetkar","BPB",8);
    Book b2=new Book(102,"Data Communications & Networking","Forouzan","Mc Graw
Hill",4);
    Book b3=new Book(103,"Operating System","Galvin","Wiley",6);
    //Adding Books to HashSet
    set.add(b1);
    set.add(b2);
    set.add(b3);
    //Traversing HashSet
    for(Book b:set){
    System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity);
    }
}
}
```

Output:

```
101 Let us C Yashwant Kanetkar BPB 8
102 Data Communications & Networking Forouzan Mc Graw Hill 4
103 Operating System Galvin Wiley 6
```

Java LinkedHashSet Example: Book
FileName: Book.java

```
import java.util.*;
class Book {
int id;
String name,author,publisher;
int quantity;
public Book(int id, String name, String author, String publisher, int quantity) {
    this.id = id;
    this.name = name;
    this.author = author;
    this.publisher = publisher;
    this.quantity = quantity;
}
}
public class LinkedHashSetExample {
public static void main(String[] args) {
    LinkedHashSet<Book> hs=new LinkedHashSet<Book>();
    //Creating Books
    Book b1=new Book(101,"Let us C","Yashwant Kanetkar","BPB",8);
    Book b2=new Book(102,"Data Communications & Networking","Forouzan","Mc Graw
Hill",4);
    Book b3=new Book(103,"Operating System","Galvin","Wiley",6);
    //Adding Books to hash table
    hs.add(b1);
    hs.add(b2);
    hs.add(b3);
    //Traversing hash table
    for(Book b:hs){
    System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity);
    }
}
}
```

Output:

```
101 Let us C Yashwant Kanetkar BPB 8
102 Data Communications & Networking Forouzan Mc Graw Hill 4
103 Operating System Galvin Wiley 6
```

TreeSetExample.java

```
import java.util.*;
class Book implements Comparable<book>{
int id;
String name,author,publisher;
int quantity;
public Book(int id, String name, String author, String publisher, int quantity) {
    this.id = id;
    this.name = name;
    this.author = author;
    this.publisher = publisher;
    this.quantity = quantity;
}
// implementing the abstract method
public int compareTo(Book b) {
    if(id>b.id){
        return 1;
    }else if(id<b.id){
        return -1;
    }else{
    return 0;
    }
}
}
public class TreeSetExample {
public static void main(String[] args) {
    Set<book> set=new TreeSet<book>();
    //Creating Books
    Book b1=new Book(121,"Let us C","Yashwant Kanetkar","BPB",8);
    Book b2=new Book(233,"Operating System","Galvin","Wiley",6);
    Book b3=new Book(101,"Data Communications & Networking","Forouzan","Mc Graw
Hill",4);
    //Adding Books to TreeSet
    set.add(b1);
    set.add(b2);
    set.add(b3);
    //Traversing TreeSet
    for(Book b:set){
    System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.quantity);
    }
}
}
```

Output:

101 Data Communications & Networking Forouzan Mc Graw Hill 4
121 Let us C Yashwant Kanetkar BPB 8
233 Operating System Galvin Wiley 6

```
// Java Program to Demonstrate Working of Map interface
import java.util.*;

// Main class
class GFG {
    // Main driver method
    public static void main(String args[])
    {
        // Creating an empty HashMap
        Map<String, Integer> hm  = new HashMap<String, Integer>();
        // Inserting pairs in above Map using put() method
        hm.put("a", new Integer(100));
        hm.put("b", new Integer(200));
        hm.put("c", new Integer(300));
        hm.put("d", new Integer(400));
        // Traversing through Map using for-each loop
        for (Map.Entry<String, Integer> me : hm.entrySet()) {
            // Printing keys
            System.out.print(me.getKey() + ":");
            System.out.println(me.getValue());
        }
    }
}
```

Output

```
a:100
b:200
c:300
d:400
```

HashMap

```
// Java program to demonstrate the working of Map interface

import java.util.*;
class GFG {
    public static void main(String args[])
    {
        // Default Initialization of a Map
        Map<Integer, String> hm1 = new HashMap<>();

        // Initialization of a Map using Generics
        Map<Integer, String> hm2  = new HashMap<Integer, String>();

        // Inserting the Elements
        hm1.put(1, "Geeks");
        hm1.put(2, "For");
        hm1.put(3, "Geeks");
        hm2.put(new Integer(1), "Geeks");
        hm2.put(new Integer(2), "For");
        hm2.put(new Integer(3), "Geeks");
        System.out.println(hm1);
        System.out.println(hm2);
        // demonstrate the working of Map interface
        Map<Integer, String> hm1 = new HashMap<Integer, String>();

        // Inserting the Elements
        hm1.put(new Integer(1), "Geeks");
        hm1.put(new Integer(2), "Geeks");
        hm1.put(new Integer(3), "Geeks");

        System.out.println("Initial Map " + hm1);

        hm1.put(new Integer(2), "For");

        System.out.println("Updated Map " + hm1);


//to demonstrate the working of Map interface
        hm1.remove(new Integer(4));
        // Final Map
        System.out.println(hm1);
// to demonstrate the working of Map interface

        for (Map.Entry mapElement : hm1.entrySet()) {
            int key = (int)mapElement.getKey();
```

```
            // Finding the value
            String value = (String)mapElement.getValue();
            System.out.println(key + " : " + value);
        }
// to Count the Occurrence of numbers using Hashmap
        int a[] = { 1, 13, 4, 1, 41, 31, 31, 4, 13, 2 };
        // put all elements in arraylist
        ArrayList<Integer> aa = new ArrayList();
        for (int i = 0; i < a.length; i++) {
            aa.add(a[i]);
        }
        HashMap<Integer, Integer> h = new HashMap();
        // counting occurrence of numbers
       for (int i = 0; i < aa.size(); i++) {
         h.putIfAbsent(aa.get(i), Collections.frequency(aa, aa.get(i)));
        }
        System.out.println(h);
    }
}
```

Output
{1=Geeks, 2=For, 3=Geeks}
{1=Geeks, 2=For, 3=Geeks}

Initial Map {1=Geeks, 2=Geeks, 3=Geeks}
Updated Map {1=Geeks, 2=For, 3=Geeks}

{1=Geeks, 2=For, 3=Geeks, 4=For}
{1=Geeks, 2=For, 3=Geeks}

1 : Geeks
2 : For
3 : Geeks

{1=2, 2=1, 4=2, 41=1, 13=2, 31=2}
Treemap

```
import java.util.*;
public class TreeMap2 {
   public static void main(String args[]) {
    TreeMap<integer ,string> map=new TreeMap<integer ,string>();
      map.put(100,"Amit");
      map.put(102,"Ravi");
      map.put(101,"Vijay");
      map.put(103,"Rahul");
      System.out.println("Before invoking remove() method");
```

```
    for(Map.Entry m:map.entrySet())
    {
       System.out.println(m.getKey()+" "+m.getValue());
    }
    map.remove(102);
    System.out.println("After invoking remove() method");
    for(Map.Entry m:map.entrySet())
    {
       System.out.println(m.getKey()+" "+m.getValue());
    }
    }
}
```

Output:

```
Before invoking remove() method
100 Amit
101 Vijay
102 Ravi
103 Rahul
After invoking remove() method
100 Amit
101 Vijay
103 Rahul
```

Treemap

```
public void givenTreeMap_whenOrdersEntriesNaturally_thenCorrect() {
    TreeMap<Integer, String> map = new TreeMap<>();
    map.put(3, "val");
    map.put(2, "val");
    map.put(1, "val");
    map.put(5, "val");
    map.put(4, "val");

    assertEquals("[1, 2, 3, 4, 5]", map.keySet().toString());

TreeMap<Integer, String> map = new TreeMap<>(Comparator.reverseOrder());
    map.put(3, "val");
    map.put(2, "val");
    map.put(1, "val");
    map.put(5, "val");
    map.put(4, "val");

    assertEquals("[5, 4, 3, 2, 1]", map.keySet().toString());


//TreeMap stores all its entries in sorted order.
Integer highestKey = map.lastKey();
    Integer lowestKey = map.firstKey();
    Set<Integer> keysLessThan3 = map.headMap(3).keySet();
    Set<Integer> keysGreaterThanEqTo3 = map.tailMap(3).keySet();

    assertEquals(new Integer(5), highestKey);
    assertEquals(new Integer(1), lowestKey);
    assertEquals("[1, 2]", keysLessThan3.toString());
    assertEquals("[3, 4, 5]", keysGreaterThanEqTo3.toString());
}
```

Queue

```
import java.util.Queue;
import java.util.LinkedList;

class Main {
    public static void main(String[] args) {
        // Creating Queue using the LinkedList class
        Queue<Integer> numbers = new LinkedList<>();
        // offer elements to the Queue
        numbers.offer(1);
        numbers.offer(2);
        numbers.offer(3);
        System.out.println("Queue: " + numbers);
        // Access elements of the Queue
        int accessedNumber = numbers.peek();
        System.out.println("Accessed Element: " + accessedNumber);
        // Remove elements from the Queue
        int removedNumber = numbers.poll();
        System.out.println("Removed Element: " + removedNumber);
        System.out.println("Updated Queue: " + numbers);
    }
}
```

Queue

```
import java.util.LinkedList;
import java.util.Queue;

public class QueueExample {

    public static void main(String[] args)
    {
        Queue<Integer> q  = new LinkedList<>();
        // Adds elements {0, 1, 2, 3, 4} to the queue
        for (int i = 0; i < 5; i++)
            q.add(i);
        // Display contents of the queue.
        System.out.println("Elements of queue "   + q);

        // To remove the head of queue.
        int removedele = q.remove();
        System.out.println("removed element-"        + removedele);

        System.out.println(q);

        // To view the head of queue
        int head = q.peek();
        System.out.println("head of queue-" + head);
    // Rest all methods of collection interface like size and contains can be used with this
implementation.
        int size = q.size();
        System.out.println("Size of queue-"   + size);
    }
}
```

Java PriorityQueue Example
FileName: TestCollection12.java

```
import java.util.*;
class TestCollection12 {
    public static void main(String args[]) {
        // Creating a PriorityQueue of Strings
        PriorityQueue<string> queue = new PriorityQueue<string>();
        // Adding elements to the PriorityQueue
        queue.add("Amit");
        queue.add("Vijay");
        queue.add("Karan");
        queue.add("Jai");
        queue.add("Rahul");
        // Displaying the head of the queue using element() method
        // This method throws an exception if the queue is empty
        System.out.println("head:" + queue.element());
        // Displaying the head of the queue using peek() method
        // This method returns null if the queue is empty
        System.out.println("head:" + queue.peek());
        // Iterating through the queue elements
        System.out.println("iterating the queue elements:");
        Iterator<string> itr = queue.iterator();
        while (itr.hasNext()) {
            System.out.println(itr.next());
        }
        // Removing the head of the queue using remove() method
        // This method throws an exception if the queue is empty
        queue.remove();
        // Removing the head of the queue using poll() method
        // This method returns null if the queue is empty
        queue.poll();
        // Displaying the queue elements after removing two elements
        System.out.println("after removing two elements:");
        Iterator<string> itr2 = queue.iterator();
        while (itr2.hasNext()) {
            System.out.println(itr2.next());
        }
    }
}
</string></string></string></string>
```

Output:

```
head:Amit
head:Amit
iterating the queue elements:
```

Amit
Jai
Karan
Vijay
Rahul
after removing two elements:
Karan
Rahul
Vijay

Comparable interface to sort integers.

```
import java.util.*;

class Number implements Comparable<Number> {
    int v; // Value of the number

    // Constructor
    public Number(int v) {
        this.v = v;
    }

    // toString() for displaying the number
    @Override
    public String toString() {
        return String.valueOf(v);
    }

    // compareTo() method to
    // define sorting logic
    @Override
    public int compareTo(Number o) {

        // Ascending order
        return this.v - o.v;
    }

    public static void main(String[] args) {

        // Create an array of Number objects
        Number[] n = { new Number(4), new Number(1),
                       new Number(7), new Number(2) };

        System.out.println("Before Sorting: "
                           + Arrays.toString(n));

        // Sort the array
        Arrays.sort(n);

        // Display numbers after sorting
        System.out.println("After Sorting: " + Arrays.toString(n));
    }
}
```

Output
Before Sorting: [4, 1, 7, 2]
After Sorting: [1, 2, 4, 7]

Comparator interface

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;

// Define a Car class
class Car {
  public String brand;
  public String model;
  public int year;

  public Car(String b, String m, int y) {
    brand = b;
    model = m;
    year = y;
  }
}

// Create a comparator
class SortByYear implements Comparator {
  public int compare(Object obj1, Object obj2) {
    // Make sure that the objects are Car objects
    Car a = (Car) obj1;
    Car b = (Car) obj2;

    // Compare the year of both objects
    if (a.year < b.year) return -1; // The first car has a smaller year
    if (a.year > b.year) return 1;  // The first car has a larger year
    return 0; // Both cars have the same year
  }
}

public class Main {
  public static void main(String[] args) {
    // Create a list of cars
    ArrayList<Car> myCars = new ArrayList<Car>();
    myCars.add(new Car("BMW", "X5", 1999));
    myCars.add(new Car("Honda", "Accord", 2006));
    myCars.add(new Car("Ford", "Mustang", 1970));

    // Use a comparator to sort the cars
    Comparator myComparator = new SortByYear();
    Collections.sort(myCars, myComparator);

    // Display the cars
```

```
    for (Car c : myCars) {
      System.out.println(c.brand + " " + c.model + " " + c.year);
    }
  }
}
```