/*
ArrayList and its methods:

--> Java ArrayList class uses a dynamic array for storing the elements.
--> It is like an array, but there is no size limit. We can add or remove elements anytime.
--> So, it is much more flexible than the traditional array. It is found in the java.util package.
--> It implements the List interface, so we can use all the methods of the List interface here.

Important Points:
--> Java ArrayList class can contain duplicate elements.
--> Java ArrayList class maintains insertion order.
--> Java ArrayList class is non synchronized.
--> Java ArrayList allows random access because the array works on an index basis.
--> In ArrayList, manipulation is a little bit slower than the LinkedList in Java because a lot of shifting needs to occur if any element is removed from the array list.
--> We can not create an array list of the primitive types, such as int, float, char, etc.
--> It is required to use the required wrapper class in such cases.
Ex:
ArrayList<int> al = ArrayList<int>(); // does not work
ArrayList<Integer> al = new ArrayList<Integer>(); // works fine
***/**

```
// import statement is mandatory
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Comparator;
public class A3
{
   public static void main(String[] args)
   {
      ArrayList a1=new ArrayList();
      // It is used to build an empty array list for Hetrogeneous data
      //with initial capacity "10" if Array list reaches its maximum
      //capacity then a new Arraylist object will be created with
      //New capacity=(current capacity *3/2)+1
      a1.add("Vikram");//string
      a1.add(10);//int
      a1.add('B');//char
      a1.add(10.5);//double
      a1.add(11.5f);//float
      a1.add(25l);//long
      a1.add(true);//boolean
      a1.add(null);//null
      System.out.println(a1);//[Vikram, 10, B, 10.5, 11.5, 25, true, null]

       // ArrayList object creation and Constructors
```

//ArrayList constructor (1)

ArrayList<Integer> al1 = new ArrayList<>(); // It is used to build an empty array list for specified Homogeneous data.

// 1. boolean add(E e) - It is used to append the specified element at the end of a list.

```
al1.add(10);
al1.add(20);
al1.add(30);
al1.add(10);
al1.add(40);
al1.add(50);
System.out.println(al1);//[10, 20, 30, 10, 40, 50]

System.out.println("\n----------*****----------");
```

// 2. void add(int index, E element) - It is used to insert the specified element at the specified position in a list.

```
al1.add(1,15); // it will add element 15 at position 1
System.out.println("Array list al1::"+al1);//Array list al1::[10, 15, 20, 30, 10, 40, 50]
```

//ArrayList constructor (2)

// It is used to build an array list that is initialized with the elements of the collection c i.e., ArrayList.

```
ArrayList<Integer> al2 = new ArrayList<>(al1);
System.out.println("Array list al2::"+al2);//Array list al2::[10, 15, 20, 30, 10, 40, 50]
```

// It is used to build an array list that has the specified initial capacity.
//note*:-we can specify the initial capacity but cannot see the capaity
//because capacity is growable

```
ArrayList<Integer> al3 = new ArrayList<>(5);
```

// 3. void clear() - It is used to remove all the elements from this list.

```
al2.clear();
System.out.println("Array list al2::"+al2);//Array list al2::[]
```

// 4. void ensureCapacity(int requiredCapacity) -
//It is used to enhance the capacity of an ArrayList instance.
//note*:-we can specify the initial capacity but cannot see the capaity
//because capacity is growable

```
al1.ensureCapacity(12);
```

// 5. E get(int index) - It is used to fetch the element from the particular position of the list.

System.out.println("ArrayList al1:"+al1**);//ArrayList al1:[10, 15, 20, 30, 10, 40, 50]**
System.out.println(al1.get(3));**//30**

// 6. E set(int index, E element) - It is used to replace the specified element in the list, present at the specified position.and this method returns replaced value.

System.out.println("ArrayList al1:"+al1);**//ArrayList al1:[10, 15, 20, 30, 10, 40, 50]**
System.out.println(al1.set(1,78));**//15 returning the replaced value**
System.out.println("ArrayList al1:"+al1);**//ArrayList al1:[10, 15, 20, 30, 10, 40, 50]**
//System.out.println(al1.set(10,78)); **// IndexOutOfBoundException**

// 7. boolean isEmpty() - It returns true if the list is empty, otherwise false.

System.out.println("ArrayList al1:"+al1);**//ArrayList al1:[10, 78, 20, 30, 10, 40, 50]**
System.out.println(al1.isEmpty()); **// false**
System.out.println("ArrayList al2:"+al2);**//ArrayList al2:[]**
System.out.println(al2.isEmpty()); **// true**

//8. int indexOf(Object o) -
//It is used to return the index in this list of the first occurrence of the specified element, or -1 if the List does not contain this element.

System.out.println("ArrayList al1:"+al1); **//ArrayList al1:[10, 78, 20, 30, 10, 40, 50]**
System.out.println(al1.indexOf(10)); **//0 returns first occurance**
System.out.println(al1.indexOf(50)); **//6 returns first occurance**
System.out.println(al1.indexOf(60));**//-1 List does not contain this element.**

//9. int lastIndexOf(Object o) -
//It is used to return the index in this list of the last occurrence of the specified element,
//or -1 if the list does not contain this element.

System.out.println("ArrayList al1:"+al1); **//ArrayList al1:[10, 78, 20, 30, 10, 40, 50]**
System.out.println(al1.lastIndexOf(10)); **//4 returns last occurance**
System.out.println(al1.lastIndexOf(70)); **//-1 List does not contain this element.**

//10. public int remove(int indexposition):
//It is used to delete an element from the collection.
//In argument pass index position that you want to remove.
//It returns removed value.

System.out.println("ArrayList al1:"+al1); **//ArrayList al1:[10, 78, 20, 30, 10, 40, 50]**
System.out.println(al1.remove(2)); **//20 returns removed value**
System.out.println("ArrayList al1:"+al1); **//ArrayList al1:[10, 78, 30, 10, 40, 50]**
System.out.println(al1.remove(4)); **//40 returns removed value**

```
        System.out.println("ArrayList al1:"+al1);//ArrayList al1:[10, 78, 30, 10, 50]

        //System.out.println(al1.remove(10));
       // R.E: IndexOutOfBoundsException, as entered position is not available in ArrayList

        //11. void sort(Comparator<? super E> c)
        //It is used to sort the elements of the list on the basis of the specified comparator.

        System.out.println("ArrayList al1:"+al1); //ArrayList al1:[10, 78, 30, 10, 50]

        // sort the ArrayList in ascending order
        al1.sort(Comparator.naturalOrder());
    System.out.println("Sorted ArrayList al1:"+al1);//Sorted ArrayList al1:[10, 10, 30, 50, 78]

        // sort the ArrayList in reverse order
        al1.sort(Comparator.reverseOrder());
  System.out.println("reverse ArrayList al1:"+al1); //reverse ArrayList al1:[78, 50, 30, 10, 10]

        // 12. int size() - It is used to return the number of elements present in the list.

        System.out.println(al1.size());//5

        // 13. Add Multiple element with Arrays.asList() in Constructor

      ArrayList<String> al4 = new ArrayList<>(Arrays.asList("AB","CD","EG","YZ"));

        // Use of for each loop to print elements of al4

        for (String s : al4)
        {
           System.out.println(s);
        }
    /*  AB
        CD
        EG
        YZ  */
    }
}
```

//Methods of Collection interface:
//add(), addAll(), clear(), contains(), isEmpty(), iterator(), remove(), removeAll(), toArray()

```
// import statements are mandatory
import java.util.ArrayList;
import java.util.Iterator;

public class A2
{
   public static void main(String[] args)
   {
      // Object creation using Any of the class of collection framework

      ArrayList<Integer> al1 = new ArrayList<Integer>();
      ArrayList<Integer> al2 = new ArrayList<>();
      ArrayList<Integer> al3 = new ArrayList();

   // 1. public boolean add(E e) method: It is used to insert an element
      al1.add(10);
      al1.add(20);
      al1.add(30);
      al1.add(40);
      al1.add(50);
      System.out.println(al1);//[10, 20, 30, 40, 50]
      al2.add(1);
      al2.add(2);
      al2.add(3);
      al2.add(4);
      al2.add(5);
      System.out.println(al2);//[1, 2, 3, 4, 5]
      al3.add(10);
      al3.add(30);
      al3.add(40);
      al3.add(50);
      al3.add(60);
      System.out.println(al3);//[10, 30, 40, 50, 60]

 // 2. public boolean addAll(Collection<? extends E> c) method:
//  It is used to insert the specified collection elements in the invoking collection.

      al1.addAll(al3);
      System.out.println(al1);//[10, 20, 30, 40, 50, 10, 30, 40, 50, 60]

      al1.addAll(2,al2);
      System.out.println(al1);//[10, 20, 1, 2, 3, 4, 5, 30, 40, 50, 10, 30, 40, 50, 60]
```

// 3. public void clear(): It removes the total number of elements from the collection.

```
System.out.println("Use of clear() method: ");
al2.clear();
System.out.println(al2);//[]
```

// 4. public boolean contains(Object element)
//It is used to search an element inside collection.

```
System.out.println("Use of contains() method: ");
System.out.println(al1.contains(10)); //true
System.out.println(al1.contains(100));//false
```

// 5. public boolean isEmpty(): It checks if collection is empty.

```
System.out.println("Use of isEmpty() method: ");
System.out.println(al1.isEmpty());//false
System.out.println(al2.isEmpty());//true
```

// 6. public int remove(int indexposition): It is used to delete an element from the collection.
// In argument pass index position that you want to remove.

```
System.out.println("Use of remove() method: ");
System.out.println(al1);//[10, 20, 1, 2, 3, 4, 5, 30, 40, 50, 10, 30, 40, 50, 60]
System.out.println(al1.remove(2));//1
System.out.println(al1);//[10, 20, 2, 3, 4, 5, 30, 40, 50, 10, 30, 40, 50, 60]
System.out.println(al1.remove(6));//30
System.out.println(al1);//[10, 20, 2, 3, 4, 5, 40, 50, 10, 30, 40, 50, 60]
System.out.println(al3);//[10, 30, 40, 50, 60]

//System.out.println(al1.remove(13));
//R.E: IndexOutOfBoundsException, as entered position is not available in ArrayList
```

//7. public boolean removeAll(Collection<?> c):
//It is used to delete all the elements of the specified collection from the invoking collection.

```
System.out.println(al1.removeAll(al3));//true
System.out.println(al1);//[20, 2, 3, 4, 5]
```

// 8. public Object[] toArray(): It converts collection into array.

```
System.out.println("Use of toArray() method: ");

Object[] obj = al1.toArray();
for (Object x : obj)
{
   System.out.println(x);
}
```

```
        /*Use of toArray() method:
        20
        2
        3
        4
        5*/

    // 9. public Iterator iterator() - It returns an iterator.

        System.out.println("Use of iterator() method: ");
        Iterator iterator = al1.iterator();
        while(iterator.hasNext()){
           System.out.println(iterator.next());
        }
        /*Use of iterator() method:
        20
        2
        3
        4
        5*/
    }
}
```

LinkedList and its methods:

- Java LinkedList class uses a doubly linked list to store the elements.
- It provides a linked-list data structure.
- It implements List and Deque interfaces.

The important points about Java LinkedList are:

- Java LinkedList class can contain duplicate elements.
- Java LinkedList class maintains insertion order.
- Java LinkedList class is non synchronized.
- In Java LinkedList class, manipulation is fast because no shifting needs to occur.
- Java LinkedList class can be used as a list, stack or queue.
- In the case of a doubly linked list, we can add or remove elements from both sides.

```
import java.util.Iterator;
import java.util.LinkedList;
public class A4
{
   public static void main(String[] args)
   {
      //LinkedList constructor (1)
      // It is used to construct an empty list.
      LinkedList<Double> L1 = new LinkedList<>();

      // 1. boolean add(E e)
      //It is used to append the specified element to the end of a list.

      // L1.add(10); // C.E.: Element must be Double
      L1.add(10.0);
      L1.add(20.0);
      L1.add(30.0);
      L1.add(40.0);
      L1.add(50.0);
      System.out.println("LinkedList-L1::"+L1);
      //LinkedList-L1::[10.0, 20.0, 30.0, 40.0, 50.0]

      //LinkedList constructor (2)
```

```
// It is used to construct a list containing the elements of the
// specified collection, in the order, they are returned by the
// collection's iterator.

LinkedList<Double> L2 = new LinkedList<>(L1);
System.out.println("LinkedList L2 is (Copy of L1): "+L2);
//LinkedList L2 is (Copy of L1): [10.0, 20.0, 30.0, 40, 50.0]

//LinkedList<Double> L3 = new LinkedList<>(10);
//C.E:Cannot infer type arguments for LinkedList<>
// C.E.: No concept of initial capacity

// 2. void add(int index, E element)
//It is used to insert the specified element at the specified
//position index in a list.

LinkedList<Double> L3 = new LinkedList<>();
//L3.add(2,16.0);
//RE:IndexOutOfBoundsException: Index: 2, Size: 1
//Always start from index 0

System.out.println("LinkedList-L1::"+L1);
//LinkedList-L1::[10.0, 20.0, 30.0, 40.0, 50.0]
L1.add(2,15.0);
// L1.add(7,15.0); // R.E.: IndexOutOfBoundsException
L1.add(6,12.0);
// You can add up to +1 position of given Linked List
System.out.println("LinkedList-L1::"+L1);
//LinkedList-L1::[10.0, 20.0, 15.0, 30.0, 40.0, 50.0, 12.0]

//3. boolean addAll(Collection<? extends E> c)
//It is used to append all of the elements in the specified
//collection to the end of this list,in the order that they
//are returned by the specified collection's iterator.

System.out.println("LinkedList-L3::"+L3);//LinkedList-L3::[]
System.out.println(L3.addAll(L1));//true
System.out.println("LinkedList-L3::"+L3);
//LinkedList-L3::[10.0, 20.0, 15.0, 30.0, 40.0, 50.0, 12.0]

//4. boolean addAll(int index, Collection<? extends E> c)
//It is used to append all the elements in the specified
//collection, starting at the specified position of the list.
```

```
System.out.println("LinkedList L2 is: "+L2);
//LinkedList L2 is: [10.0, 20.0, 30.0, 40.0, 50.0]
System.out.println("LinkedList L1 is: "+L1);
//LinkedList L1 is: [10.0, 20.0, 15.0, 30.0, 40.0, 50.0, 12.0]
System.out.println(L2.addAll(2,L1));//true
System.out.println("LinkedList L2 is: "+L2);
//LinkedList L2 is: [10.0, 20.0, 10.0, 20.0, 15.0, 30.0, 40.0, 50.0, 12.0, 30.0,
40.0, 50.0]

// 5. void addFirst(E e)
  //It is used to insert the given element at the beginning of a list.

L1.addFirst(121.0);

// 6. void addLast(E e)
  //It is used to append the given element to the end of a list.

L1.addLast(225.0);
System.out.println("LinkedList L1 is: "+L1);
//LinkedList L1 is: [121.0, 10.0, 20.0, 15.0, 30.0, 40.0, 50.0, 12.0, 225.0]

// 7. void clear()
//It is used to remove all the elements from a list.

L2.clear();
System.out.println("LinkedList L2 is: "+L2);//LinkedList L2 is: []

// 8. boolean contains(Object o)
//It is used to return true if a list contains a specified element.

System.out.println("L1 contains 12.0 ? "+L1.contains(12.0));
//L1 contains 12.0 ? true
System.out.println("L1 contains 331.0 ? "+L1.contains(331.0));
//L1 contains 331.0 ? false

// 9. E get(int index)
//It is used to return the element at the specified position in a list.

System.out.println("LinkedList L1 is: "+L1);
//LinkedList L1 is: [121.0, 10.0, 20.0, 15.0, 30.0, 40.0, 50.0, 12.0, 225.0]
System.out.println("Element at Index 6 is: "+L1.get(6));
//Element at Index 6 is: 50.0
```

```
        //System.out.println("Element at Index 11 is: "+L1.get(11));
        //R.E:IndexOutOfBoundsException

        // 10. E getFirst()
        //It is used to return the first element in a list.

        System.out.println("First element of L1 is: "+L1.getFirst());
        //First element of L1 is: 121.0

        // 11. E getLast()
        //It is used to return the last element in a list.

        System.out.println("Last element of L1 is: "+L1.getLast());
        //Last element of L1 is: 225.0

        // 12. E removeFirst()
        //It is used to retrieve and remove the first element of a list.
        System.out.println("LinkedList L1 is: "+L1);
        //LinkedList L1 is: [121.0, 10.0, 20.0, 15.0, 30.0, 40.0, 50.0, 12.0,
        225.0]
        System.out.println(L1.removeFirst());//121.0

        //13. E removeLast()
        //It is used to retrieve and remove the Last element of a list.

        System.out.println(L1.removeLast());//225.0
        System.out.println("LinkedList L1 is: "+L1);
        //LinkedList L1 is: [10.0, 20.0, 15.0, 30.0, 40.0, 50.0, 12.0]

        //Also explain remove() and remove(int Index)

        // 14. Use of iterator to print elements of List

        Iterator itr = L1.iterator();
        while(itr.hasNext())
        {
           System.out.println(itr.next());
        }
    }
}
```

Stack:

- The stack is a linear data structure that is used to store the collection of objects.
- It is based on Last-In-First-Out (LIFO).
- The stack data structure has the two most important operations that are push and pop.
- The push operation inserts an element into the stack and pop operation removes an element from the top of the stack.
- Stack only defines the default constructor, which creates an empty stack.
- Stack includes all the methods defined by Vector, and adds several of its own.

```
import java.util.*;

class Collection6
{
   public static void main(String[] args)
   {

      // Only no argument constructor for Stack
      Stack<Integer> s1 = new Stack<>();

      System.out.println("Default capacity of Stack: "+s1.capacity());
      //Default capacity of Stack: 10

 // 1. Object push(Object element) Pushes the element onto the stack. Element is also returned.

      System.out.println("Element 1 added is: "+s1.push(12));
      System.out.println("Element 2 added is: "+s1.push(10));
      System.out.println("Element 3 added is: "+s1.push(14));
      System.out.println("Element 3 added is: "+s1.push(17));
      System.out.println("Element 3 added is: "+s1.push(22));

      System.out.println("s1 is: "+s1);//s1 is: [12, 10, 14, 17, 22]


// 2. Object pop( )- Returns the element on the top of the stack, removing it in the process.
```

```
        System.out.println("Top most element removed is: "+s1.pop());
        //Top most element removed is: 22

        System.out.println("Updated s1 after pop() is: "+s1);
        //Updated s1 after pop() is: [12, 10, 14, 17]

// 3. Object peek( ) - Returns the element on the top of the stack, but does not remove it.

        System.out.println("Top most element is: "+s1.peek());
        //Top most element is: 17

        System.out.println("Updated s1 after peek() is: "+s1);
        //Updated s1 after peek() is: [12, 10, 14, 17]

// 4. int search(Element e) - Searches for element in the stack. If found, its offset from the top of stack is returned, else -1is returned.

        System.out.println("Position of element 17 from top is: "+s1.search(17));
        //Position of element 17 from top is: 1

        System.out.println("Position of element 10 from top is: "+s1.search(+10));
       // Position of element 10 from top is: 3

        System.out.println("Position of element 21 from top is: "+s1.search(21));
        //Position of element 21 from top is: -1

        // 5. Iterator iterator() - Fetching value of element

        Iterator itr = s1.iterator();
        while(itr.hasNext())
        {
           System.out.println(itr.next());
        }
    }
}
```

//Vector class Methods
/*

-In vector class Every method is synchronized
-At a time only one Thread is allow to
operate on Vector object and hence Vector
object is Thread safe.
-Relatively performance is low because
Threads are required to wait.
-Vector class is legacy and introduced in 1.0v

(1)The underlying data structur is resizable array (or) growable array
(2)Duplicate objects are allowed
(3)Insertion order is preserved
(4)Hetrogeneous objects are allowed.
(5)Null insertion is possible

*/

```
import java.util.*;
 class a2
{
   public static void main(String[] args)
   {
   //Constructor-1
      Vector<Integer> al1 = new Vector<>();
   //Creates an empty collection with initial default capacity 10
      System.out.println(al1);
      System.out.println(al1.capacity());
      al1.add(1);
      al1.add(2);
      al1.add(3);
      al1.add(4);
      al1.add(5);
      al1.add(6);
      al1.add(7);
      al1.add(8);
      al1.add(9);
      al1.add(10);
      System.out.println(al1);
      System.out.println(al1.capacity());
```

```
        al1.add(11);
        System.out.println(al1);
        System.out.println(al1.capacity());

    //Constructor-2
    //Vector<Integer> al2 = new Vector<>(int initialCapacity);
        Vector<Integer> al2 = new Vector<>(2);
    //Creates an empty collection with default capacity 2
        System.out.println(al2);
        System.out.println(al2.capacity());
        al2.add(10);
        al2.add(20);
        al2.add(30);
        System.out.println(al2);
        System.out.println(al2.capacity());

    //Constructor-3
    //Vector<Integer> al3 = new Vector<>(int initialCapacity,int
capacityIncrement));
         Vector<Integer> al3 = new Vector<>(1,1);
    //Creates an empty collection with default capacity 1 and will
    //increment by 1 when max limit reached
        System.out.println(al3);
        System.out.println(al3.capacity());
        al3.add(100);
        al3.add(200);
        System.out.println(al3);
        System.out.println(al3.capacity());

    //Constructor-4
    //Vector<Integer> al4 = new Vector<>(collection c);
        Vector<Integer> al4 = new Vector<>(al3);
    //creates a collection that is initialized with the elements of collection
        System.out.println(al4);

    //elementAt(int Index)

        System.out.println(al1);
        System.out.println(al1.elementAt(2));

    //equals(Object o)
        System.out.println(al1.equals(al2));
        System.out.println(al3.equals(al4));
```

```
    //public synchronized Object clone()
        System.out.println(al1.clone());
        Vector<Integer> al5 = (Vector)al1.clone();
        System.out.println(al5);
    }
}
```

Collection class -

- Java collection class is used exclusively with static methods that operate on or return collections.
- It inherits Object class.
- The important points about Java Collections class are:
- Java Collection class throws a NullPointerException if the collections or class objects provided to them are null.

```
import java.util.*;
public class A5
{
   public static void main(String[] args)
   {
      ArrayList<Integer> al1 = new ArrayList<>();

      al1.add(10);
      al1.add(20);
      al1.add(30);
      al1.add(10);
      al1.add(10);
      al1.add(40);
      al1.add(-50);
      al1.add(0);
      System.out.println("ArrayList al1: "+al1);
      //ArrayList al1: [10, 20, 30, 10, 10, 40, -50, 0]

      //1. Collections.max(Collection c) -
      //It is used to get the maximum value of the given collection,
      //according to the natural ordering of its elements.

      System.out.println("Max. element in al1 is: "+Collections.max(al1));
      //Max. element in al1 is: 40

      //2. Collections.min(Collection c)
      //It is used to get the minimum value of the given collection,
      //according to the natural ordering of its elements.

      System.out.println("Max. element in al1 is: "+Collections.min(al1));
      //Max. element in al1 is: -50

      // 3. static void reverse(Collection c)
      //It is used to reverse the order of the elements in the specified list.
```

```
        Collections.reverse(al1);
        System.out.println("Reversed al1 is: "+al1);
        //Reversed al1 is: [0, -50, 40, 10, 10, 30, 20, 10]

        // 4. Collections.sort(Collection c)
        //It is used to sort the elements presents in the specified list
        //of collection inascending order.

        Collections.sort(al1);
        System.out.println("al1 Sorted in ascending order: "+al1);
        //al1 Sorted in ascending order: [-50, 0, 10, 10, 10, 20, 30, 40]

        //5. Collections.frequency(Collections c, Element e)
        //It is used to get the number of elements in the specified
        //collection equal to the specified object.

System.out.println("Frequency of 10 in al1is:"+Collections.frequency(al1,10));
        //Frequency of 10 in al1 is: 3
    }
}
```

// Use of Comperator.comparing() Method

Sort class Objects by its properties for this we use Comparator class and comparing() method in this program create getter methods for each property which are used for sorting sort by Id ,Sort by Name

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

 class Book
 {
    int bookId;
    String bookName;

    public Book(int bookId, String bookName)
    {
        this.bookId = bookId;
        this.bookName = bookName;
    }
```

```
  //getter method for Task-5
  public int getBookId()
  {
     return bookId;
  }

  //getter method for Task-4
  public String getBookName()
  {
     return bookName;
  }

  public String toString()
  {
     String msg="{id="+bookId+"::Name="+bookName+"}";
     return msg;
  }
}

class Main
{
  public static void main(String[] args)
  {
     //Task 1 adding list of book and returning the id and name
     //using toString method

     Book b1=new Book(12,"Java");
     Book b2=new Book(13,"DBMS");
     Book b3=new Book(14,"Maths");
     Book b4=new Book(15,"DS");
     System.out.println(b1);
     System.out.println(b2);
     System.out.println(b3);
     System.out.println(b4);

     //Output task-1
     /*  {id=12::Name=Java}
        {id=13::Name=DBMS}
        {id=14::Name=Maths}
        {id=15::Name=DS}       */
```

//Task 2 Adding Books in Array list and printing List of books

```
ArrayList<Book> l1=new ArrayList<Book>();
l1.add(b1);
l1.add(b2);
l1.add(b3);
l1.add(b4);
System.out.println();
System.out.println("List of Books::"+l1);
System.out.println();
```

// Output Task-2
//List of Books::[{id=12::Name=Java}, {id=13::Name=DBMS}, {id=14::Name=Maths}, {id=15::Name=DS}]

//Task-3 Printing ArrayList usingIterator.iterator() Method

```
Iterator itr=l1.iterator();
while(itr.hasNext())
{
   System.out.println(itr.next());
   System.out.println();
}
```

// Task-3 Output

/* {id=12::Name=Java}

{id=13::Name=DBMS}

{id=14::Name=Maths}

{id=15::Name=DS} */

//Task-4 Sorting the list of books as per book name using comperator.comparing()

```
Collections.sort(l1,Comparator.comparing(Book::getBookName));

System.out.println(">>>>>>Sorting as per Book Name<<<<<<<");
System.out.println();

Iterator itr1=l1.iterator();
while(itr1.hasNext())
```

```
        {
            System.out.println(itr1.next());
            System.out.println();
        }
        //OutPut Task-4
        /*>>>>>>Sorting as per Book Name<<<<<<<

            {id=13::Name=DBMS}

            {id=15::Name=DS}

            {id=12::Name=Java}

            {id=14::Name=Maths}     */

//Task-5 Sorting the list of books as per bookId using comperator.comparing()

        Collections.sort(l1,Comparator.comparing(Book::getBookId));

        System.out.println(">>>>>>Sorting as per Book ID<<<<<<<");
        System.out.println();

        Iterator itr2=l1.iterator();
        while(itr2.hasNext())
        {
            System.out.println(itr2.next());
            System.out.println();
        }

        //Output Task-5
        /*
            >>>>>>Sorting as per Book ID<<<<<<<

            {id=12::Name=Java}

            {id=13::Name=DBMS}

            {id=14::Name=Maths}

            {id=15::Name=DS}    */
    }
}
```