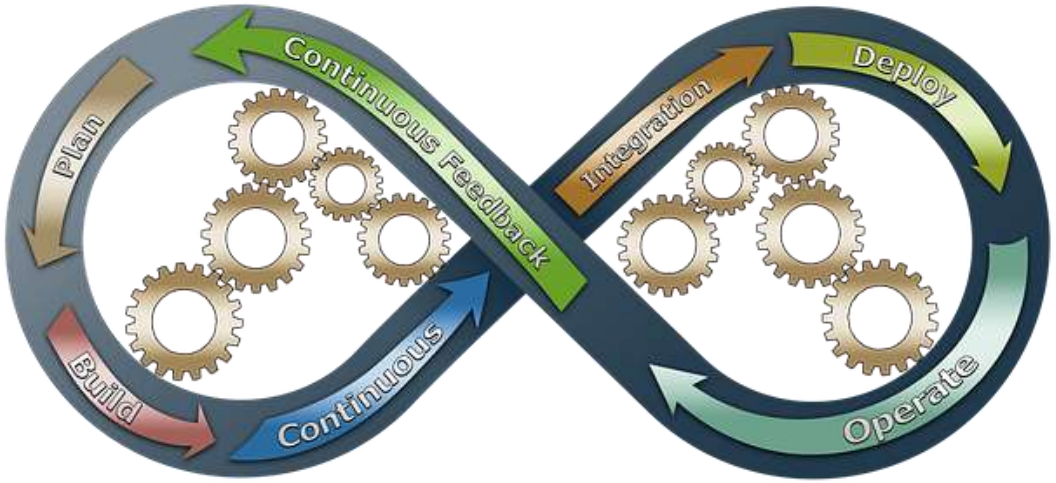


Session 3.9

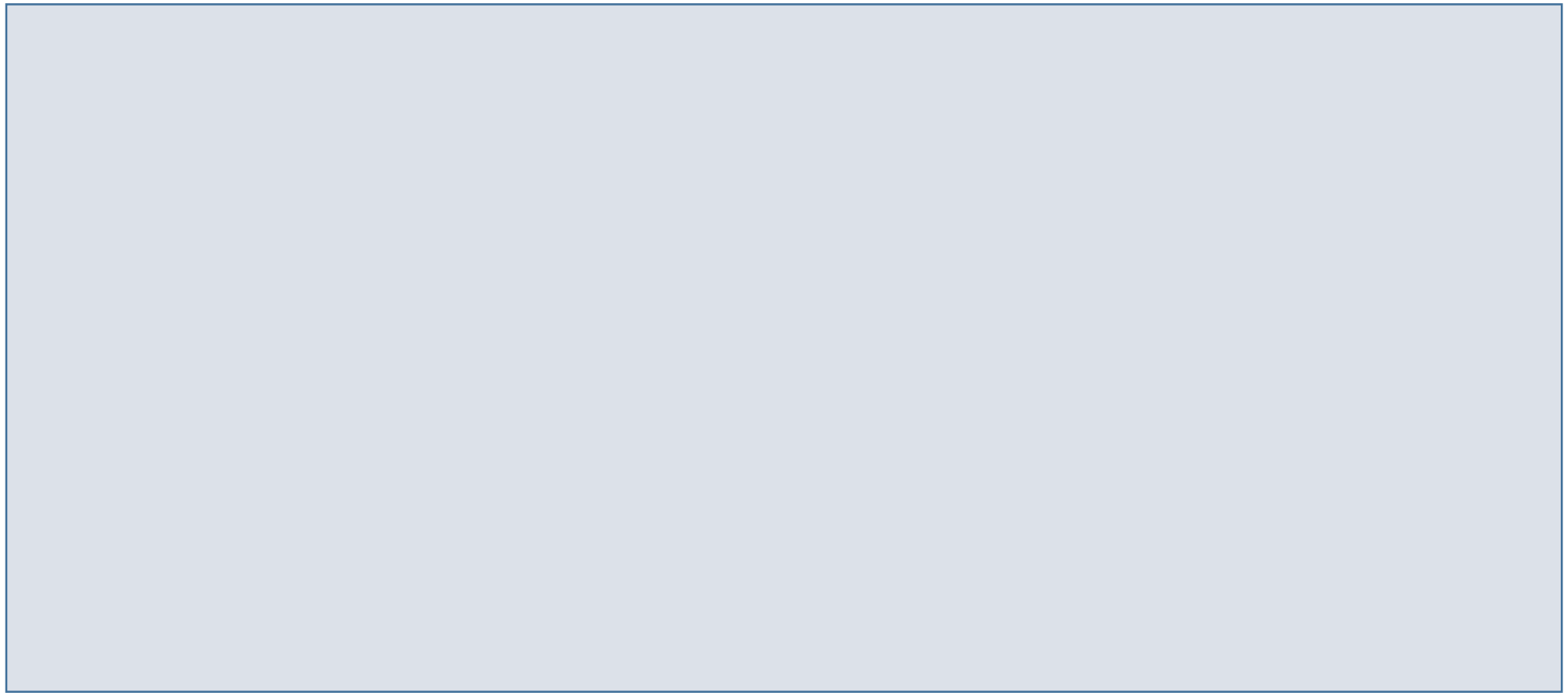
Collections

AN INITIATIVE BY

UNICAL ACADEMY

Introduction

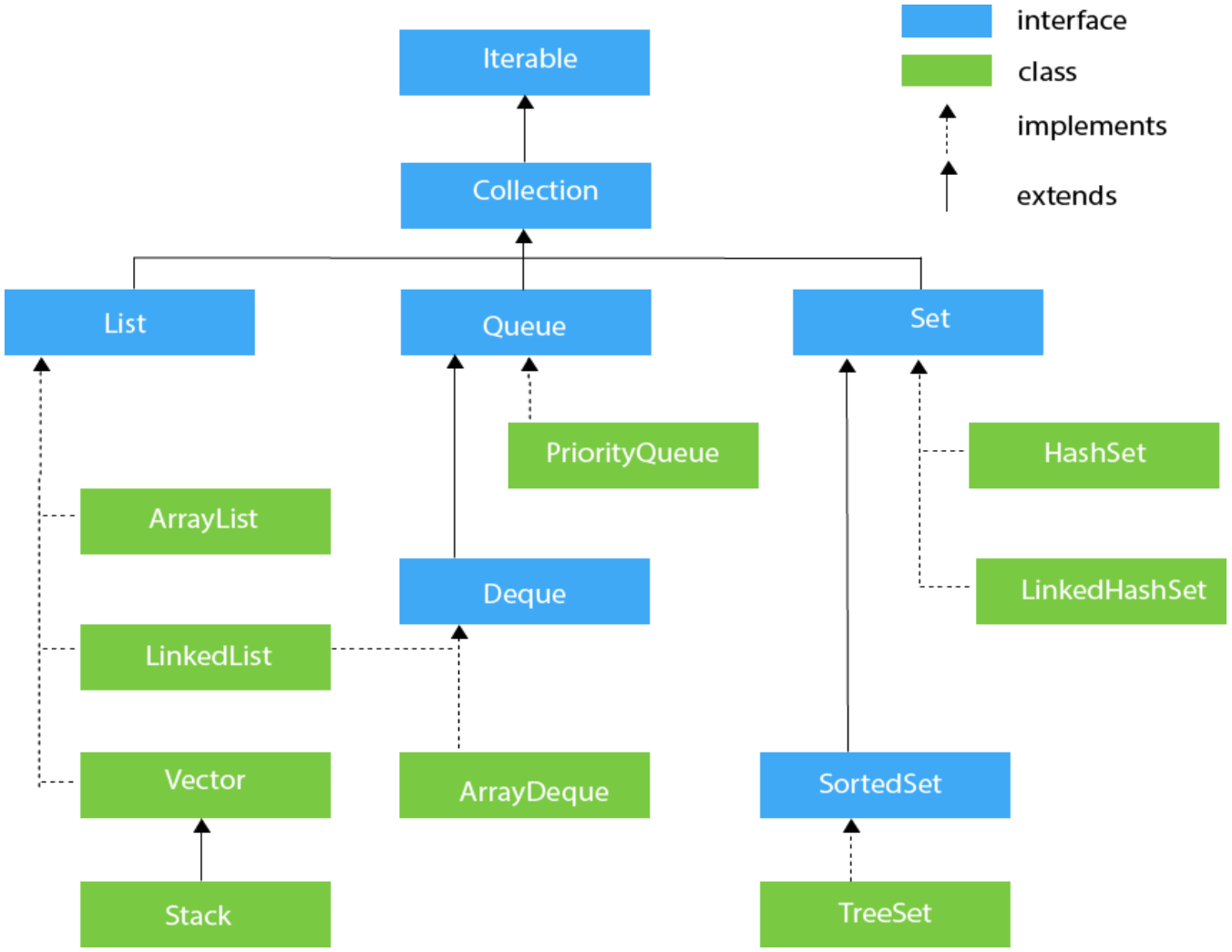
UNICAL ACADEMY



Let’s go!!!

Collections

- The Collection is a framework that provides an architecture to store and manipulate the group of objects.
- Collections can achieve all the operations that we perform on a data such as searching, sorting, insertion, manipulation, and deletion.
- Collection framework provides many interfaces (Set, List, Queue, Deque) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet).



Collection Interface

UNICAL ACADEMY

S. No.	Method	Description
1	public boolean add(E e)	It is used to insert an element in this collection.
2	public boolean addAll(Collection<? extends E> c)	It is used to insert the specified collection elements in the invoking collection.
3	public boolean remove(Object element)	It is used to delete an element from the collection.
4	public boolean removeAll(Collection<?> c)	It is used to delete all the elements of the specified collection from the invoking collection.
5	default boolean removeIf(Predicate<? super E> filter)	It is used to delete all the elements of the collection that satisfy the specified predicate.
6	public boolean retainAll(Collection<?> c)	It is used to delete all the elements of invoking collection except the specified collection.
7	public int size()	It returns the total number of elements in the collection.
8	public void clear()	It removes the total number of elements from the collection.
9	public boolean contains(Object element)	It is used to search an element.
10	public boolean containsAll(Collection<?> c)	It is used to search the specified collection in the collection.
11	public Iterator iterator()	It returns an iterator.
12	public Object[] toArray()	It converts collection into array.
13	public <T> T[] toArray(T[] a)	It converts collection into array. Here, the runtime type of the returned array is that of the specified array.
14	public boolean isEmpty()	It checks if collection is empty.
15	default Stream<E> parallelStream()	It returns a possibly parallel Stream with the collection as its source.
16	default Stream<E> stream()	It returns a sequential Stream with the collection as its source.
17	default Spliterator<E> spliterator()	It generates a Spliterator over the specified elements in the collection.
18	public boolean equals(Object element)	It matches two collections.
19	public int hashCode()	It returns the hash code number of the collection.

List, Set, Queue

List:

- List interface is the child interface of Collection interface. It inhibits a list type data structure in which we can store the ordered collection of objects. It can have duplicate values.

To instantiate the List interface, we must use:

```
List <data-type> list1= new ArrayList();
List <data-type> list2 = new LinkedList();
List <data-type> list3 = new Vector();
List <data-type> list4 = new Stack();
```

ArrayList:

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

LinkedList:

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

Vector:

- Vector uses a dynamic array to store the data elements. It is similar to ArrayList. However, It is synchronized and contains many methods that are not the part of Collection framework.

Stack:

- The stack is the subclass of Vector. It implements the last-in-first-out data structure, i.e., Stack. The stack contains all of the methods of Vector class and also provides its methods like boolean push(), boolean peek(), boolean push(object o), which defines its properties.

UNICAL ACADEMY

Set:

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

Set can be instantiated as:

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

HashSet:

- HashSet class implements Set Interface. It represents the collection that uses a hash table for storage. Hashing is used to store the elements in the HashSet. It contains unique items.

LinkedHashSet:

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

TreeSet

.

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

UNICAL ACADEMY

Queue:

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

Queue interface can be instantiated as:

```
Queue<String> q1 = new PriorityQueue();
Queue<String> q2 = new ArrayDeque();
```

PriorityQueue:

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

Deque:

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

Deque can be instantiated as:

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

ArrayDeque:

ArrayDeque class implements the Deque interface. It facilitates us to use the Deque. Unlike queue, we can add or delete the elements from both the ends. ArrayDeque is faster than ArrayList and Stack and has no capacity restrictions.

Map

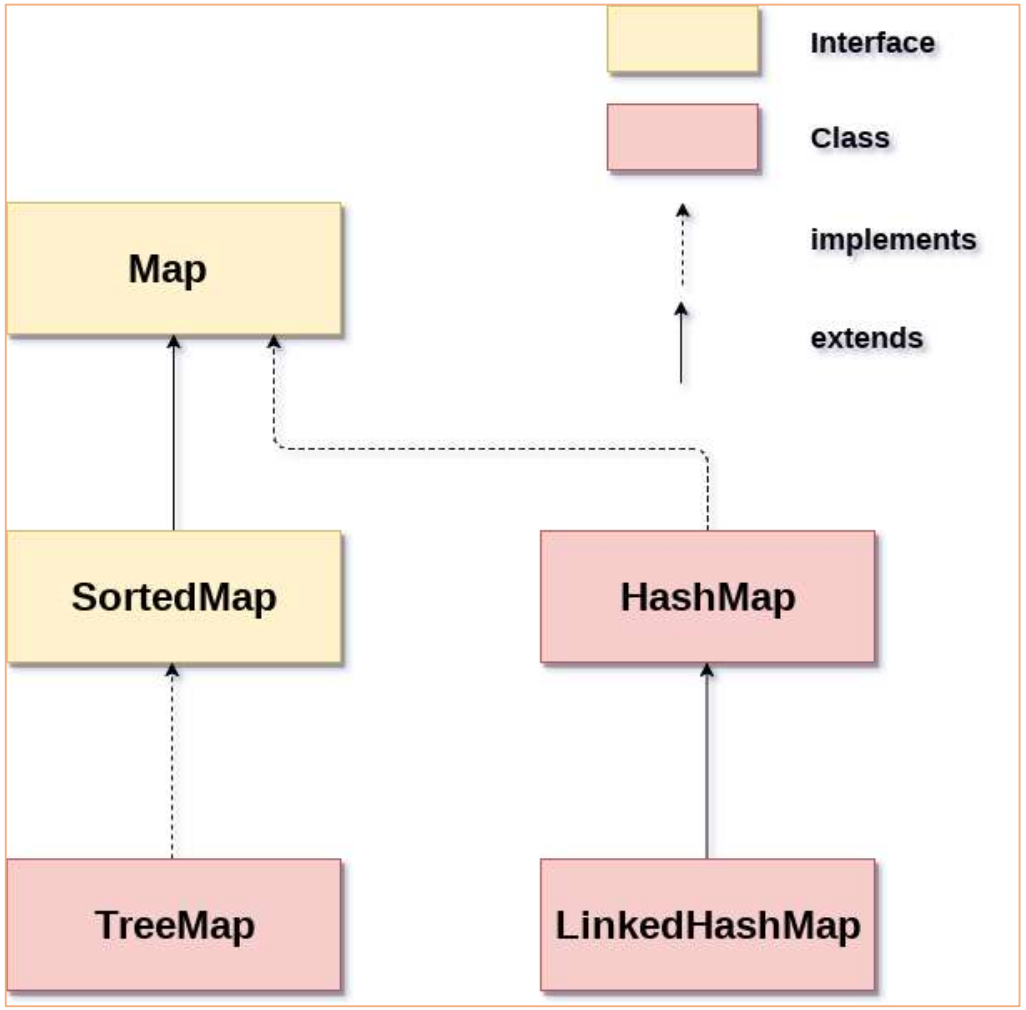
UNICAL ACADEMY

Map:

A map contains values on the basis of key, i.e. key and value pair. Each key and value pair is known as an entry. A Map contains unique keys.

Class	Description
HashMap	HashMap is the implementation of Map, but it doesn't maintain any order.
LinkedHashMap	LinkedHashMap is the implementation of Map. It inherits HashMap class. It maintains insertion order.
TreeMap	TreeMap is the implementation of Map and SortedMap. It maintains ascending order.

Map Hierarchy:



UNICAL ACADEMY

Map. Entry Interface:

Entry is the sub interface of Map. So we will be accessed it by Map. Entry name. It returns a collection-view of the map, whose elements are of this class. It provides methods to get key and value.

Method	Description
K getKey()	It is used to obtain a key.
V getValue()	It is used to obtain value.
int hashCode()	It is used to obtain hashCode.
V setValue(V value)	It is used to replace the value corresponding to this entry with the specified value.
boolean equals(Object o)	It is used to compare the specified object with the other existing objects.
static <K extends Comparable<? super K>,V> Comparator<Map.Entry<K,V>> comparingByKey()	It returns a comparator that compare the objects in natural order on key.
static <K,V> Comparator<Map.Entry<K,V>> comparingByKey(Comparator<? super K> cmp)	It returns a comparator that compare the objects by key using the given Comparator.
static <K,V extends Comparable<? super V>> Comparator<Map.Entry<K,V>> comparingByValue()	It returns a comparator that compare the objects in natural order on value.
static <K,V> Comparator<Map.Entry<K,V>> comparingByValue(Comparator<? super V> cmp)	It returns a comparator that compare the objects by value using the given Comparator.

Collection implementation classes

UNICAL ACADEMY

S. No.	Modifier & Type	Methods	Descriptions
1)	static <T> boolean	addAll()	It is used to adds all of the specified elements to the specified collection.
2)	static <T> Queue<T>	asLifoQueue()	It returns a view of a Deque as a Last-in-first-out (LIFO) Queue.
3)	static <T> int	binarySearch()	It searches the list for the specified object and returns their position in a sorted list.
4)	static <E> Collection<E>	checkedCollection()	It is used to returns a dynamically typesafe view of the specified collection.
5)	static <E> List<E>	checkedList()	It is used to returns a dynamically typesafe view of the specified list.
6)	static <K,V> Map<K,V>	checkedMap()	It is used to returns a dynamically typesafe view of the specified map.
7)	static <K,V> NavigableMap<K,V>	checkedNavigableMap()	It is used to returns a dynamically typesafe view of the specified navigable map.
8)	static <E> NavigableSet<E>	checkedNavigableSet()	It is used to returns a dynamically typesafe view of the specified navigable set.
9)	static <E> Queue<E>	checkedQueue()	It is used to returns a dynamically typesafe view of the specified queue.
10)	static <E> Set<E>	checkedSet()	It is used to returns a dynamically typesafe view of the specified set.
11)	static <K,V> SortedMap<K,V>	checkedSortedMap()	It is used to returns a dynamically typesafe view of the specified sorted map.
12)	static <E> SortedSet<E>	checkedSortedSet()	It is used to returns a dynamically typesafe view of the specified sorted set.
13)	static <T> void	copy()	It is used to copy all the elements from one list into another list.
14)	static boolean	disjoint()	It returns true if the two specified collections have no elements in common.
15)	static <T> Enumeration<T>	emptyEnumeration()	It is used to get an enumeration that has no elements.
16)	static <T> Iterator<T>	emptyIterator()	It is used to get an Iterator that has no elements.
17)	static <T> List<T>	emptyList()	It is used to get a List that has no elements.
18)	static <T> ListIterator<T>	emptyListIterator()	It is used to get a List Iterator that has no elements.
19)	static <K,V> Map<K,V>	emptyMap()	It returns an empty map which is immutable.

UNICAL ACADEMY

S. No.	Modifier & Type	Methods	Descriptions
20)	static <K,V> NavigableMap<K,V>	emptyNavigableMap()	It returns an empty navigable map which is immutable.
21)	static <E> NavigableSet<E>	emptyNavigableSet()	It is used to get an empty navigable set which is immutable in nature.
22)	static <T> Set<T>	emptySet()	It is used to get the set that has no elements.
23)	static <K,V> SortedMap<K,V>	emptySortedMap()	It returns an empty sorted map which is immutable.
24)	static <E> SortedSet<E>	emptySortedSet()	It is used to get the sorted set that has no elements.
25)	static <T> Enumeration<T>	enumeration()	It is used to get the enumeration over the specified collection.
26)	static <T> void	fill()	It is used to replace all of the elements of the specified list with the specified elements.
27)	static int	frequency()	It is used to get the number of elements in the specified collection equal to the specified object.
28)	static int	indexOfSubList()	It is used to get the starting position of the first occurrence of the specified target list within the specified source list. It returns -1 if there is no such occurrence in the specified list.
29)	static int	lastIndexOfSubList()	It is used to get the starting position of the last occurrence of the specified target list within the specified source list. It returns -1 if there is no such occurrence in the specified list.
30)	static <T> ArrayList<T>	list()	It is used to get an array list containing the elements returned by the specified enumeration in the order in which they are returned by the enumeration.
31)	static <T extends Object & Comparable<? super T>> T	max()	It is used to get the maximum value of the given collection, according to the natural ordering of its elements.
32)	static <T extends Object & Comparable<? super T>> T	min()	It is used to get the minimum value of the given collection, according to the natural ordering of its elements.
33)	static <T> List<T>	nCopies()	It is used to get an immutable list consisting of **n** copies of the specified object.
34)	static <E> Set<E>	newSetFromMap()	It is used to return a set backed by the specified map.
35)	static <T> boolean	replaceAll()	It is used to replace all occurrences of one specified value in a list with the other specified value.
36)	static void	reverse()	It is used to reverse the order of the elements in the specified list.
37)	static <T> Comparator<T>	reverseOrder()	It is used to get the comparator that imposes the reverse of the natural ordering on a collection of objects which implement the Comparable interface.
38)	static void	rotate()	It is used to rotate the elements in the specified list by a given distance.
39)	static void	shuffle()	It is used to randomly reorders the specified list elements using a default randomness.

UNICAL ACADEMY

S. No.	Modifier & Type	Methods	Descriptions
40)	static <T> Set<T>	singleton()	It is used to get an immutable set which contains only the specified object.
41)	static <T> List<T>	singletonList()	It is used to get an immutable list which contains only the specified object.
42)	static <K,V> Map<K,V>	singletonMap()	It is used to get an immutable map, mapping only the specified key to the specified value.
43)	static <T extends Comparable<? super T>>void	sort()	It is used to sort the elements presents in the specified list of collection in ascending order.
44)	static void	swap()	It is used to swap the elements at the specified positions in the specified list.
45)	static <T> Collection<T>	synchronizedCollection()	It is used to get a synchronized (thread-safe) collection backed by the specified collection.
46)	static <T> List<T>	synchronizedList()	It is used to get a synchronized (thread-safe) collection backed by the specified list.
47)	static <K,V> Map<K,V>	synchronizedMap()	It is used to get a synchronized (thread-safe) map backed by the specified map.
48)	static <K,V> NavigableMap<K,V>	synchronizedNavigableMap()	It is used to get a synchronized (thread-safe) navigable map backed by the specified navigable map.
49)	static <T> NavigableSet<T>	synchronizedNavigableSet()	It is used to get a synchronized (thread-safe) navigable set backed by the specified navigable set.
50)	static <T> Set<T>	synchronizedSet()	It is used to get a synchronized (thread-safe) set backed by the specified set.
51)	static <K,V> SortedMap<K,V>	synchronizedSortedMap()	It is used to get a synchronized (thread-safe) sorted map backed by the specified sorted map.
52)	static <T> SortedSet<T>	synchronizedSortedSet()	It is used to get a synchronized (thread-safe) sorted set backed by the specified sorted set.
53)	static <T> Collection<T>	unmodifiableCollection()	It is used to get an unmodifiable view of the specified collection.
54)	static <T> List<T>	unmodifiableList()	It is used to get an unmodifiable view of the specified list.
55)	static <K,V> Map<K,V>	unmodifiableMap()	It is used to get an unmodifiable view of the specified map.
56)	static <K,V> NavigableMap<K,V>	unmodifiableNavigableMap()	It is used to get an unmodifiable view of the specified navigable map.
57)	static <T> NavigableSet<T>	unmodifiableNavigableSet()	It is used to get an unmodifiable view of the specified navigable set.
58)	static <T> Set<T>	unmodifiableSet()	It is used to get an unmodifiable view of the specified set.
59)	static <K,V> SortedMap<K,V>	unmodifiableSortedMap()	It is used to get an unmodifiable view of the specified sorted map.
60	static <T> SortedSet<T>	unmodifiableSortedSet()	It is used to get an unmodifiable view of the specified sorted set.

Session Recap

UNICAL ACADEMY