

## **Java**

### **Assignment 3.1**



#### **Prepared by:**

Name of Student : Chaitanya Dalvi

Roll No: 19

Batch: 2023-27

Dept. of CSE

**Name of Student:** Chaitanya Dalvi

**Roll Number:** 19

**Experiment No:** 1

---

**Title:** Demonstrate the implementation for the following constructors:

- ArrayList(),
- ArrayList(Collection c),
- ArrayList(int capacity))

• **Code:**

```
import java.util.*;

public class ArrayListEx {
    public static void main(String[] args) {
        // ArrayList()
        List<Integer> list1 = new ArrayList<>();
        list1.add(1);
        list1.add(2);
        list1.add(3);
        System.out.println("ArrayList() : " + list1);

        // ArrayList(Collection c)
        List<Integer> list2 = new ArrayList<>(list1);
        System.out.println("ArrayList(Collection c) : " + list2);

        // ArrayList(int capacity)
        List<Integer> list3 = new ArrayList<>(5);
        list3.add(1);
        list3.add(2);
        list3.add(3);
        System.out.println("ArrayList(int capacity) : " + list3);

        // public boolean add(Object obj)
        list1.add(4);
        System.out.println("public boolean add(Object obj) : " +
list1);

        // public boolean addAll(Collection c)
        list1.addAll(list3);
```

```

        System.out.println("public boolean addAll(Collection c) : "
+ list1);

        // public boolean remove(Object obj)
        list1.remove(4);
        System.out.println("public boolean remove(Object obj) : " +
list1);

        // public boolean removeAll(Collection c)
        list1.removeAll(list3);
        System.out.println("public boolean removeAll(Collection
c) : " + list1);

        // public void clear()
        list1.clear();
        System.out.println("public void clear() : " + list1);

        // public boolean contains(Object obj)
        list1.addAll(list3);
        System.out.println("public boolean contains(Object obj) : "
+ list1.contains(1));

        // public boolean containsAll(Collection c)
        System.out.println("public boolean containsAll(Collection
c) : " + list1.containsAll(list3));

        // public boolean retainAll(Collection c)
        list1.retainAll(list3);
        System.out.println("public boolean retainAll(Collection
c) : " + list1);

        // boolean isEmpty()
        System.out.println("boolean isEmpty() : " +
list1.isEmpty());

        // int size()
        System.out.println("int size() : " + list1.size());

        // void add(int index, E element)
        list1.add(0, 0);
        System.out.println("void add(int index, E element) : " +
list1);

        // boolean add(E e)
        list1.add(5);
        System.out.println("boolean add(E e) : " + list1);

        // boolean addAll(int index, Collection c)
        list1.addAll(1, list3);

```

```

        System.out.println("boolean addAll(int index, Collection c)
: " + list1);

        // E get(int index)
        System.out.println("E get(int index) : " + list1.get(1));

        // int indexOf(Object o)
        System.out.println("int indexOf(Object o) : " +
list1.indexOf(1));

        // boolean contains(Object o)
        System.out.println("boolean contains(Object o) : " +
list1.contains(1));

        // E remove(int index)
        System.out.println("E remove(int index) : " +
list1.remove(1));

        // boolean remove(Object o)
        System.out.println("boolean remove(Object o) : " +
list1.remove((Integer) 1));

        // E set(int index, E element)
        list1.set(0, 1);
        System.out.println("E set(int index, E element) : " +
list1);

        // ListIterator <E> listIterator()
        ListIterator<Integer> it = list1.listIterator();
        System.out.print("ListIterator <E> listIterator() : ");
        while (it.hasNext()) {
            System.out.print(it.next() + " ");
        }

        // List <E>subList(int start, int end)
        List<Integer> subList = list1.subList(0, 2);
        System.out.println("\nList <E>subList(int start, int end) :
" + subList);

        // Iterate ArrayList using for loop
        System.out.print("Iterate ArrayList using for loop : ");
        for (int i = 0; i < list1.size(); i++) {
            System.out.print(list1.get(i) + " ");
        }

        // Iterate ArrayList using for each loop
        System.out.print("\nIterate ArrayList using for each loop :
");
        for (Integer i : list1) {

```

```

        System.out.print(i + " ");
    }

    // Iterate ArrayList using Iterator
    System.out.print("\nIterate ArrayList using Iterator : ");
    Iterator<Integer> itr = list1.iterator();
    while (itr.hasNext()) {
        System.out.print(itr.next() + " ");
    }

    // Iterate ArrayList using ListIterator
    System.out.print("\nIterate ArrayList using ListIterator : ");

    ListIterator<Integer> listItr = list1.listIterator();
    while (listItr.hasNext()) {
        System.out.print(listItr.next() + " ");
    }

    // Use List.of(), List.copyOf(), List.asList() to create a
list
    List<Integer> l1 = List.of(1, 2, 3);
    System.out.println("\nList.of() : " + l1);

    List<Integer> l2 = List.copyOf(list1);
    System.out.println("List.copyOf() : " + l2);

    List<Integer> l3 = List.of(1, 2, 3);
    List<Integer> l4 = List.copyOf(l3);
    System.out.println("List.copyOf() : " + l4);
}
}

```

## Output: (screenshot)

```

lbox/scripts:/Users/chaitanyadalvi/Library/Application Support/JetBrains/Toolbox/scripts:/usr/local/mysql-9.0.1-macos
14-arm64 not found
● chaitanyadalvi@Mac assignment3.1 2 % cd "/Users/chaitanyadalvi/Desktop/Dir1/assignment3.1 2/" && javac ArrayListEx.
java && java ArrayListEx
ArrayList() : [1, 2, 3]
ArrayList(Collection c) : [1, 2, 3]
ArrayList(int capacity) : [1, 2, 3]
public boolean add(Object obj) : [1, 2, 3, 4]
public boolean addAll(Collection c) : [1, 2, 3, 4, 1, 2, 3]
public boolean remove(Object obj) : [1, 2, 3, 4, 2, 3]
public boolean removeAll(Collection c) : [4]
public void clear() : []
public boolean contains(Object obj) : true
public boolean containsAll(Collection c) : true
public boolean retainAll(Collection c) : [1, 2, 3]
boolean isEmpty() : false
int size() : 3
void add(int index, E element) : [0, 1, 2, 3]
boolean add(E e) : [0, 1, 2, 3, 5]
boolean addAll(int index, Collection c) : [0, 1, 2, 3, 1, 2, 3, 5]
E get(int index) : 1
int indexOf(Object o) : 1
boolean contains(Object o) : true
E remove(int index) : 1
boolean remove(Object o) : true
E set(int index, E element) : [1, 2, 3, 2, 3, 5]
ListIterator<E> listIterator() : 1 2 3 2 3 5
List<E>subList(int start, int end) : [1, 2]
Iterate ArrayList using for loop : 1 2 3 2 3 5
Iterate ArrayList using for each loop : 1 2 3 2 3 5
Iterate ArrayList using Iterator : 1 2 3 2 3 5
Iterate ArrayList using ListIterator : 1 2 3 2 3 5
List.of() : [1, 2, 3]
List.copyOf() : [1, 2, 3, 2, 3, 5]
List.copyOf() : [1, 2, 3]
○ chaitanyadalvi@Mac assignment3.1 2 %

```

**Name of Student:** Chaitanya Dalvi

**Roll Number:** 19

**Experiment No:** 2

---

**Title:**WAP to implement the following methods of LinkedList

**Demonstrate the implementation for the following constructors:**

- **LinkedList()**
- **LinkedList(Collection c)**

```
import java.util.*;

public class LinkedListEx {
    public static void main(String[] args) {
        // LinkedList()
        List<Integer> list1 = new LinkedList<>();
        list1.add(1);
        list1.add(2);
        list1.add(3);
        System.out.println("LinkedList() : " + list1);

        // LinkedList(Collection c)
        List<Integer> list2 = new LinkedList<>(list1);
        System.out.println("LinkedList(Collection c) : " + list2);

        // a) to Insert Elements into the LinkedList at the last
        position
        list1.add(4);
        System.out.println("Insert Elements into the LinkedList at
        the last position : " + list1);

        // b) Add an element or collection of elements at a
        specific position of a
        // LinkedList
        list1.add(1, 5);
        System.out
            .println("Add an element or collection of elements
            at a specific position of a LinkedList : " + list1);

        // c) to retrieve the first item from LinkedList
        System.out.println("Retrieve the first item from LinkedList
        : " + list1.get(0));

        // d) to retrieve the First Occurrence of the Specified
        Elements in a
```

```

        // LinkedList
        System.out.println("Retrieve the First Occurrence of the
Specified Elements in a LinkedList : " + list1.get(1));

        // e) to retrieve the position of last occurrence of a
given element in a
        // LinkedList
        System.out.println("Retrieve the position of last
occurrence of a given element in a LinkedList : "
+ list1.lastIndexOf(5));

        // f) to Retrieve but does not Remove, the Last Element of
a LinkedList
        System.out.println(
            "Retrieve but does not Remove, the Last Element of
a LinkedList : " + list1.get(list1.size() - 1));

        // g) to Get the number of elements in a LinkedList
        System.out.println("Get the number of elements in a
LinkedList : " + list1.size());

        // h) to Check if a Particular Element exists in a
LinkedList
        System.out.println("Check if a Particular Element exists in
a LinkedList : " + list1.contains(5));

        // i) to find out whether that element exist in a
LinkedList or not. If it
        // exist retrieve the position of that element?
        System.out.println(
            "Find out whether that element exist in a
LinkedList or not. If it exist retrieve the position of that
element : "
+ list1.indexOf(list1.get(list1.size() -
1)));

        // j) to Iterate through all Elements in a LinkedList
        System.out.println("Iterate through all Elements in a
LinkedList : ");
        for (int i = 0; i < list1.size(); i++) {
            System.out.println(list1.get(i));
        }

        // k) to Iterate a LinkedList in Reverse Order
        System.out.println("Iterate a LinkedList in Reverse Order :
");
        for (int i = list1.size() - 1; i >= 0; i--) {
            System.out.println(list1.get(i));
        }

```

```

        // l) to display the elements and their positions in a
linked list
        System.out.println("Display the elements and their
positions in a linked list : ");
        for (int i = 0; i < list1.size(); i++) {
            System.out.println("Element at position " + i + " is "
+ list1.get(i));
        }

        // m) to test an LinkedList is Empty or Not
        System.out.println("Test an LinkedList is Empty or Not : "
+ list1.isEmpty());

        // n) to Replace an Element in a LinkedList
        list1.set(1, 6);
        System.out.println("Replace an Element in a LinkedList : "
+ list1);

        // o) to Remove and Return the First Element of a
LinkedList
        System.out.println("Remove and Return the First Element of
a LinkedList : " + list1.remove(0));

        // p) to remove a specified element from a linked list
        list1.remove((Integer) 6);
        System.out.println("Remove a specified element from a
linked list : " + list1);

        // q) to remove last element from a linked list
        list1.remove(list1.size() - 1);
        System.out.println("Remove last element from a linked
list : " + list1);

        // r) to remove all the elements from a linked list
        list1.clear();
        System.out.println("Remove all the elements from a linked
list : " + list1);

        // s) to pop items from the stack represented by the
LinkedList
        list1.add(1);
        list1.add(2);
        list1.add(3);
        System.out
            .println("Pop items from the stack represented by
the LinkedList : " + list1.remove(list1.size() - 1));

```



```

        // t) to Check whether an item exists in the LinkedList
collection or not
        System.out.println("Check whether an item exists in the
LinkedList collection or not : " + list1.contains(1));

        // u) to Convert a LinkedList to ArrayList
List<Integer> list3 = new ArrayList<>(list1);
        System.out.println("Convert a LinkedList to ArrayList : " +
list3);

        // v) to join two linked lists
list1.add(1);
list1.add(2);
list1.add(3);
list3.add(4);
list3.add(5);
list3.add(6);
list1.addAll(list3);
        System.out.println("Join two linked lists : " + list1);

        // w) to join an ArrayList at the end of a LinkedList
list1.addAll(list3);
        System.out.println("Join an ArrayList at the end of a
LinkedList : " + list1);

        // x) to Add LinkedList collection into another LinkedList
collection on the
        // specified index
list1.addAll(1, list3);
        System.out.println(
                "Add LinkedList collection into another LinkedList
collection on the specified index : " + list1);
    }
}

```

## Output: (screenshot)

```
✓ TERMINAL
14-arm64 not found
● chaitanyadalvi@Mac assignment3.1 2 % cd "/Users/chaitanyadalvi/Desktop/Dir1/assignment3.1 2/" && javac LinkedListEx.
java && java LinkedListEx
LinkedList() : [1, 2, 3]
LinkedList(Collection c) : [1, 2, 3]
Insert Elements into the LinkedList at the last position : [1, 2, 3, 4]
Add an element or collection of elements at a specific position of a LinkedList : [1, 5, 2, 3, 4]
Retrieve the first item from LinkedList : 1
Retrieve the First Occurrence of the Specified Elements in a LinkedList : 5
Retrieve the position of last occurrence of a given element in a LinkedList : 1
Retrieve but does not Remove, the Last Element of a LinkedList : 4
Get the number of elements in a LinkedList : 5
Check if a Particular Element exists in a LinkedList : true
Find out whether that element exist in a LinkedList or not. If it exist retrieve the position of that element : 4
Iterate through all Elements in a LinkedList :
1
5
2
3
4
Iterate a LinkedList in Reverse Order :
4
3
2
5
1
Display the elements and their positions in a linked list :
Element at position 0 is 1
Element at position 1 is 5
Element at position 2 is 2
Element at position 3 is 3
Element at position 4 is 4
Test an LinkedList is Empty or Not : false
Replace an Element in a LinkedList : [1, 6, 2, 3, 4]
Remove and Return the First Element of a LinkedList : 1
Remove a specified element from a linked list : [2, 3, 4]
Remove last element from a linked list : [2, 3]
Remove all the elements from a linked list : []
Pop items from the stack represented by the LinkedList : 3
Check whether an item exists in the LinkedList collection or not : true
Convert a LinkedList to ArrayList : [1, 2]
Join two linked lists : [1, 2, 1, 2, 3, 1, 2, 4, 5, 6]
Join an ArrayList at the end of a LinkedList : [1, 2, 1, 2, 3, 1, 2, 4, 5, 6, 1, 2, 4, 5, 6]
Add LinkedList collection into another LinkedList collection on the specified index : [1, 1, 2, 4, 5, 6, 2, 1, 2, 3,
1, 2, 4, 5, 6, 1, 2, 4, 5, 6]
○ chaitanyadalvi@Mac assignment3.1 2 %
```

**Name of Student:** Chaitanya Dalvi

**Roll Number:** 19

**Experiment No:** 3

---

**Title:**WAP to implement the following constructors and following operations on HashSet

```
import java.util.*;

class Box {
    int length;
    int height;
    int width;

    public Box(int length, int height, int width) {
        this.length = length;
        this.height = height;
        this.width = width;
    }

    public String toString() {
        return "Box [height=" + height + ", length=" + length + ",
width=" + width + "]\n";
    }
}

public class HashSetEx {
    public static void main(String[] args) {
        // HashSet()
        Set<Integer> set1 = new HashSet<>();
        set1.add(1);
        set1.add(2);
        set1.add(3);
        System.out.println("HashSet() : " + set1);

        // HashSet(Collection c)
        Set<Integer> set2 = new HashSet<>(set1);
        System.out.println("HashSet(Collection c) : " + set2);

        // 1. to convert a hash set to a List/ArrayList
        List<Integer> list = new ArrayList<>(set1);
        System.out.println("Convert a hash set to a List/
ArrayList : " + list);

        // 2. to convert a hash set to a tree set
```

```

        Set<Integer> treeSet = new TreeSet<>(set1);
        System.out.println("Convert a hash set to a tree set : " +
treeSet);

        // 3. to convert a hash set to an array
        Integer[] arr = new Integer[set1.size()];
        set1.toArray(arr);
        System.out.println("Convert a hash set to an array : " +
Arrays.toString(arr));

        // 4. to test a hash set is empty or not
        System.out.println("Test a hash set is empty or not : " +
set1.isEmpty());

        // 5. to empty an hash set
        set1.clear();
        System.out.println("Empty an hash set : " + set1);

        // 6. to get the number of elements in a hash set
        System.out.println("Get the number of elements in a hash
set : " + set2.size());

        // 7. to iterate through all elements in a hash list
        for (Integer i : set2) {
            System.out.println(i);
        }

        // 8. to add the specified element in hash set
        set2.add(4);
        System.out.println("Add the specified element in hash set :
" + set2);

        // 9. to Remove the specified element from the hashset
        set2.remove(4);
        System.out.println("Remove the specified element from the
hashset : " + set2);

        // 10. Try to add duplicate elements to HashSet
        set2.add(2);
        System.out.println("Try to add duplicate elements to
HashSet : " + set2);

        // 11. to add ArrayList elements to HashSet
        List<Integer> list1 = new ArrayList<>();
        list1.add(5);
        list1.add(6);
        list1.add(7);
        set2.addAll(list1);

```

```

        System.out.println("Add ArrayList elements to HashSet : " +
set2);

        // 12. to Copy Set content to another HashSet
Set<Integer> set3 = new HashSet<>(set2);
System.out.println("Copy Set content to another HashSet : "
+ set3);

        // 13. to Create a HashSet with string items
Set<String> set4 = new HashSet<>();
set4.add("Hello");
set4.add("World");
set4.add("Java");
System.out.println("Create a HashSet with string items : "
+ set4);

        // 14. to Print a HashSet collection using the foreach loop
for (String s : set4) {
    System.out.println(s);
}

        // 15. to Check whether a HashSet contains a specified item
or not
System.out.println("Check whether a HashSet contains a
specified item or not : " + set4.contains("Java"));

        // 16. to Create a set of Box objects using HashSet
Set<Box> boxSet = new HashSet<>();
boxSet.add(new Box(1, 2, 3));
boxSet.add(new Box(4, 5, 6));
boxSet.add(new Box(7, 8, 9));
System.out.println("Create a set of Box objects using
HashSet : " + boxSet);

        // 17. to Find the union of HashSet collections
Set<Integer> set5 = new HashSet<>();
set5.add(1);
set5.add(2);
set5.add(3);
Set<Integer> set6 = new HashSet<>();
set6.add(3);
set6.add(4);
set6.add(5);
set5.addAll(set6);
System.out.println("Find the union of HashSet collections :
" + set5);

        // 18. to Find the intersection of HashSet collection
Set<Integer> set7 = new HashSet<>();

```

```

        set7.add(1);
        set7.add(2);
        set7.add(3);
        Set<Integer> set8 = new HashSet<>();
        set8.add(3);
        set8.add(4);
        set8.add(5);
        set7.retainAll(set8);
        System.out.println("Find the intersection of HashSet
collection : " + set7);

        // 19. to compare two sets and retain elements which are
same on both sets
        Set<Integer> set9 = new HashSet<>();
        set9.add(1);
        set9.add(2);
        set9.add(3);
        Set<Integer> set10 = new HashSet<>();
        set10.add(3);
        set10.add(4);
        set10.add(5);
        set9.retainAll(set10);
        System.out.println("Compare two sets and retain elements
which are same on both sets : " + set9);

        // 20. to compare two hash sets
        System.out.println("Compare two hash sets : " +
set9.equals(set10));

        // 21. to remove all of the elements from a hash set
        set9.clear();
        System.out.println("Remove all of the elements from a hash
set : " + set9);
    }
}

```

## Output: (screenshot)



```
cd "/Users/chaitanyadalvi/Desktop/Dir1/assignment3.1 2/" && javac HashSetEx.java && java HashSetEx
/Users/chaitanyadalvi/.zshrc:1: /opt/homebrew/bin:/opt/homebrew/sbin:/usr/local/bin:/System/Cryptexes/App/usr/bin:/usr/bin:/bin:/usr/sbin:/sbin:/var/run/com.apple.security.cryptexd/codex.system/bootstrap/usr/local/bin:/var/run/com.apple.security.cryptexd/codex.system/bootstrap/usr/bin:/var/run/com.apple.security.cryptexd/codex.system/bootstrap/usr/appleinternal/bin:/opt/homebrew/bin:/opt/homebrew/sbin:/Users/chaitanyadalvi/Library/Application Support/JetBrains/Toolbox/scripts:/Users/chaitanyadalvi/Library/Application Support/JetBrains/Toolbox/scripts:/usr/local/mysql-9.0.1-macos14-arm64 not found
● chaitanyadalvi@Mac assignment3.1 2 % cd "/Users/chaitanyadalvi/Desktop/Dir1/assignment3.1 2/" && javac HashSetEx.java && java HashSetEx
HashSet() : [1, 2, 3]
HashSet(Collection c) : [1, 2, 3]
Convert a hash set to a List/ArrayList : [1, 2, 3]
Convert a hash set to a tree set : [1, 2, 3]
Convert a hash set to an array : [1, 2, 3]
Test a hash set is empty or not : false
Empty an hash set : []
Get the number of elements in a hash set : 3
1
2
3
Add the specified element in hash set : [1, 2, 3, 4]
Remove the specified element from the hashset : [1, 2, 3]
Try to add duplicate elements to HashSet : [1, 2, 3]
Add ArrayList elements to HashSet : [1, 2, 3, 5, 6, 7]
Copy Set content to another HashSet : [1, 2, 3, 5, 6, 7]
Create a HashSet with string items : [Java, Hello, World]
Java
Hello
World
Check whether a HashSet contains a specified item or not : true
Create a set of Box objects using HashSet : [Box [height=2, length=1, width=3], Box [height=5, length=4, width=6], Box [height=8, length=7, width=9]]
Find the union of HashSet collections : [1, 2, 3, 4, 5]
Find the intersection of HashSet collection : [3]
Compare two sets and retain elements which are same on both sets : [3]
Compare two hash sets : false
Remove all of the elements from a hash set : []
○ chaitanyadalvi@Mac assignment3.1 2 %
```

**Name of Student:** Chaitanya Dalvi

**Roll Number:** 19

**Experiment No:** 4

---

**Title:** WAP to create a TreeSet using following constructors:

- TreeSet( )
- TreeSet(Collection<? extends E> c)
- TreeSet(SortedSet<E> ss)

• **Code:**

```
import java.util.*;

public class TreeSetEx {
    public static void main(String[] args) {
        // TreeSet()
        Set<Integer> set1 = new TreeSet<>();
        set1.add(1);
        set1.add(2);
        set1.add(3);
        System.out.println("TreeSet() : " + set1);

        // TreeSet(Collection c)
        Set<Integer> set2 = new TreeSet<>(set1);
        System.out.println("TreeSet(Collection c) : " + set2);

        // TreeSet(SortedSet ss)
        SortedSet<Integer> sortedSet = new TreeSet<>();
        sortedSet.add(4);
        sortedSet.add(5);
        sortedSet.add(6);
        Set<Integer> set3 = new TreeSet<>(sortedSet);
        System.out.println("TreeSet(SortedSet ss) : " + set3);

        // Set.of()
        Set<Integer> set4 = Set.of(7, 8, 9);
        System.out.println("Set.of() : " + set4);

        // Set.copyOf()
        Set<Integer> set5 = Set.copyOf(set4);
        System.out.println("Set.copyOf() : " + set5);

        // E ceiling(E obj)
        System.out.println("E ceiling(E obj) : " +
            ((TreeSet<Integer>) set1).ceiling(2));
    }
}
```



```

        // Iterator<E> descendingIterator()
        Iterator<Integer> descendingIterator = ((TreeSet<Integer>)
set1).descendingIterator();
        System.out.print("Iterator<E> descendingIterator() : ");
        while (descendingIterator.hasNext()) {
            System.out.print(descendingIterator.next() + " ");
        }
        System.out.println();

        // NavigableSet<E> descendingSet()
        NavigableSet<Integer> descendingSet = ((TreeSet<Integer>)
set1).descendingSet();
        System.out.println("NavigableSet<E> descendingSet() : " +
descendingSet);

        // E floor(E obj)
        System.out.println("E floor(E obj) : " +
((TreeSet<Integer>) set1).floor(2));

        // NavigableSet<E>headSet(E, boolean)
        NavigableSet<Integer> headSet = ((TreeSet<Integer>)
set1).headSet(2, true);
        System.out.println("NavigableSet<E>headSet(E, boolean) : "
+ headSet);

        // NavigableSet<E> subSet(E lowerBound, boolean lowIncl, E
upperBound, boolean
        // highIncl)
        NavigableSet<Integer> subSet = ((TreeSet<Integer>)
set1).subSet(1, true, 2, true);
        System.out.println("NavigableSet<E> subSet(E lowerBound,
boolean lowIncl, E upperBound, boolean highIncl) : "
+ subSet);

        // E higher(E obj)
        System.out.println("E higher(E obj) : " +
((TreeSet<Integer>) set1).higher(2));

        // E lower(E obj)
        System.out.println("E lower(E obj) : " +
((TreeSet<Integer>) set1).lower(2));

        // E pollFirst()
        System.out.println("E pollFirst() : " + ((TreeSet<Integer>)
set1).pollFirst());

        // E pollLast()

```

```

        System.out.println("E pollLast() : " + ((TreeSet<Integer>)
set1).pollLast());

        // NavigableSet<E> tailSet(E lowerBound, boolean incl)
        NavigableSet<Integer> tailSet = ((TreeSet<Integer>)
set1).tailSet(2, true);
        System.out.println("NavigableSet<E> tailSet(E lowerBound,
boolean incl) : " + tailSet);
    }
}

```

## Output: (screenshot)

```

cd "/Users/chaitanyadalvi/Desktop/Dir1/assignment3.1 2/" && javac TreeSetEx.java && java TreeSetEx
/Users/chaitanyadalvi/.zshrc:1: /opt/homebrew/bin:/opt/homebrew/sbin:/usr/local/bin:/System/Cryptexes/App/usr/bin:/usr/bin:/bin:/usr/sbin:/sbin:/var/run/com.apple.security.cryptexd/codex.system/bootstrap/usr/local/bin:/var/run/com.apple.security.cryptexd/codex.system/bootstrap/usr/appleinternal/bin:/opt/homebrew/bin:/opt/homebrew/sbin:/Users/chaitanyadalvi/Library/Application Support/JetBrains/Toolbox/scripts:/Users/chaitanyadalvi/Library/Application Support/JetBrains/Toolbox/scripts:/usr/local/mysql-9.0.1-macos14-arm64 not found
● chaitanyadalvi@Mac assignment3.1 2 % cd "/Users/chaitanyadalvi/Desktop/Dir1/assignment3.1 2/" && javac TreeSetEx.java && java TreeSetEx
TreeSet() : [1, 2, 3]
TreeSet(Collection c) : [1, 2, 3]
TreeSet(SortedSet ss) : [4, 5, 6]
Set.of() : [9, 8, 7]
Set.copyOf() : [9, 8, 7]
E ceiling(E obj) : 2
Iterator<E> descendingIterator() : 3 2 1
NavigableSet<E> descendingSet() : [3, 2, 1]
E floor(E obj) : 2
NavigableSet<E> headSet(E, boolean) : [1, 2]
NavigableSet<E> subSet(E lowerBound, boolean lowIncl, E upperBound, boolean highIncl) : [1, 2]
E higher(E obj) : 3
E lower(E obj) : 1
E pollFirst() : 1
E pollLast() : 3
NavigableSet<E> tailSet(E lowerBound, boolean incl) : [2]
○ chaitanyadalvi@Mac assignment3.1 2 %

```

**Name of Student:** Chaitanya Dalvi

**Roll Number:** 19

**Experiment No:** 5

---

**Title:**WAP to create HashMaps using following constructors:

- **HashMap( )**
- **HashMap(Map<? extends K, ? extends V> m) HashMap(int capacity)**
- **HashMap(int capacity, float fillRatio)**

```
import java.util.*;

public class HashMapEx {
    public static void main(String[] args) {
        // HashMap()
        Map<Integer, String> map1 = new HashMap<>();
        map1.put(1, "One");
        map1.put(2, "Two");
        map1.put(3, "Three");
        System.out.println("HashMap() : " + map1);

        // HashMap(Map m)
        Map<Integer, String> map2 = new HashMap<>(map1);
        System.out.println("HashMap(Map m) : " + map2);

        // HashMap(int capacity)
        Map<Integer, String> map3 = new HashMap<>(10);
        map3.put(4, "Four");
        map3.put(5, "Five");
        map3.put(6, "Six");
        System.out.println("HashMap(int capacity) : " + map3);

        // HashMap(int capacity, float fillRatio)
        Map<Integer, String> map4 = new HashMap<>(10, 0.75f);
        map4.put(7, "Seven");
        map4.put(8, "Eight");
        map4.put(9, "Nine");
        System.out.println("HashMap(int capacity, float
fillRatio) : " + map4);

        // Map.of()
        Map<Integer, String> map5 = Map.of(10, "Ten", 11, "Eleven",
12, "Twelve");
```

```

System.out.println("Map.of() : " + map5);

// Map.copyOf()
Map<Integer, String> map6 = Map.copyOf(map5);
System.out.println("Map.copyOf() : " + map6);

// V get(Object key)
System.out.println("V get(Object key) : " + map1.get(1));

// V getOrDefault(Object key, V defaultValue)
System.out.println("V getOrDefault(Object key, V
defaultValue) : " + map1.getOrDefault(2, "Default"));

// V put(K key, K value)
map1.put(1, "Uno");
System.out.println("V put(K key, K value) : " + map1);

// int hashCode()
System.out.println("int hashCode() : " + map1.hashCode());

// boolean isEmpty()
System.out.println("boolean isEmpty() : " +
map1.isEmpty());

// boolean containsKey(Object k)
System.out.println("boolean containsKey(Object k) : " +
map1.containsKey(1));

// boolean containsValue (Object v)
System.out.println("boolean containsValue (Object v) : " +
map1.containsValue("Uno"));

// V remove(Object k)
System.out.println("V remove(Object k) : " +
map1.remove(1));

// boolean remove(Object key, Object value)
System.out.println("boolean remove(Object key, Object
value) : " + map1.remove(2, "Two"));

// putIfAbsent(K key, V value)
map1.putIfAbsent(2, "Two");
System.out.println("putIfAbsent(K key, V value) : " +
map1);

// V replace(K key, V value)
map1.replace(2, "Dos");
System.out.println("V replace(K key, V value) : " + map1);

```

```

        // boolean replace(K key, V oldValue, V newValue)
        System.out.println("boolean replace(K key, V oldValue, V
newValue) : " + map1.replace(2, "Dos", "Two"));

        // void clear()
        map1.clear();
        System.out.println("void clear() : " + map1);

        // int size()
        System.out.println("int size() : " + map2.size());

        // Collection values()
        System.out.println("Collection values() : " +
map2.values());

        // Set<K> keySet()
        System.out.println("Set<K> keySet() : " + map2.keySet());

        // Set<Map.Entry<K,V>> entrySet()
        System.out.println("Set<Map.Entry<K,V>> entrySet() : " +
map2.entrySet());

        // K getKey() of Map.Entry
        System.out.println("K getKey() of Map.Entry : " +
map2.entrySet().iterator().next().getKey());

        // V getValue() of Map.Entry
        System.out.println("V getValue() of Map.Entry : " +
map2.entrySet().iterator().next().getValue());
    }
}

```

## Output: (screenshot)



```
cd "/Users/chaitanyadalvi/Desktop/Dir1/assignment3.1 2/" && javac HashMapEx.java && java HashMapEx
/Users/chaitanyadalvi/.zshrc:1: /opt/homebrew/bin:/opt/homebrew/sbin:/usr/local/bin:/System/Cryptexes/App/usr/bin:/usr/bin:/bin:/usr/sbin:/sbin:/var/run/com.apple.security.cryptexd/codex.system/bootstrap/usr/local/bin:/var/run/com.apple.security.cryptexd/codex.system/bootstrap/usr/appleinternal/bin:/opt/homebrew/bin:/opt/homebrew/sbin:/Users/chaitanyadalvi/Library/Application Support/JetBrains/Toolbox/scripts:/Users/chaitanyadalvi/Library/Application Support/JetBrains/Toolbox/scripts:/usr/local/mysql-9.0.1-macos14-arm64 not found
● chaitanyadalvi@Mac assignment3.1 2 % cd "/Users/chaitanyadalvi/Desktop/Dir1/assignment3.1 2/" && javac HashMapEx.java && java HashMapEx
HashMap() : {1=One, 2=Two, 3=Three}
HashMap(Map m) : {1=One, 2=Two, 3=Three}
HashMap(int capacity) : {4=Four, 5=Five, 6=Six}
HashMap(int capacity, float fillRatio) : {7=Seven, 8=Eight, 9=Nine}
Map.of() : {12=Twelve, 11=Eleven, 10=Ten}
Map.copyOf() : {12=Twelve, 11=Eleven, 10=Ten}
V get(Object key) : One
V getOrDefault(Object key, V defaultValue) : Two
V put(K key, K value) : {1=Uno, 2=Two, 3=Three}
int hashCode() : 80956546
boolean isEmpty() : false
boolean containsKey(Object k) : true
boolean containsValue (Object v) : true
V remove(Object k) : Uno
boolean remove(Object key, Object value) : true
putIfAbsent(K key, V value) : {2=Two, 3=Three}
V replace(K key, V value) : {2=Dos, 3=Three}
boolean replace(K key, V oldValue, V newValue) : true
void clear() : {}
int size() : 3
Collection values() : [One, Two, Three]
Set<K> keySet() : [1, 2, 3]
Set<Map.Entry<K,V>> entrySet() : [1=One, 2=Two, 3=Three]
K getKey() of Map.Entry : 1
V getValue() of Map.Entry : One
○ chaitanyadalvi@Mac assignment3.1 2 %
```

**Name of Student:** Chaitanya Dalvi

**Roll Number:** 19

**Experiment No:** 6

---

**Title:** WAP to create TreeMapS using following constructors:

- **TreeMap( )**
- **TreeMap(Comparator<? super K> comp)**
- **TreeMap(Map<? extends K, ? extends V> m)**
- **TreeMap(SortedMap<K, ? extends V> sm)**

```
import java.util.*;

public class TreeMapEx {
    public static void main(String[] args) {
        // TreeMap()
        Map<Integer, String> map1 = new TreeMap<>();
        map1.put(1, "One");
        map1.put(2, "Two");
        map1.put(3, "Three");
        System.out.println("TreeMap() : " + map1);

        // TreeMap(Comparator comp)
        Map<Integer, String> map2 = new
TreeMap<>(Comparator.reverseOrder());
        map2.put(1, "One");
        map2.put(2, "Two");
        map2.put(3, "Three");
        System.out.println("TreeMap(Comparator comp) : " + map2);

        // TreeMap(Map m)
        Map<Integer, String> map3 = new TreeMap<>(map1);
        System.out.println("TreeMap(Map m) : " + map3);

        // TreeMap(SortedMap sm)
        SortedMap<Integer, String> sortedMap = new TreeMap<>();
        sortedMap.put(4, "Four");
        sortedMap.put(5, "Five");
        sortedMap.put(6, "Six");
        Map<Integer, String> map4 = new TreeMap<>(sortedMap);
        System.out.println("TreeMap(SortedMap sm) : " + map4);

        // K firstKey()
```

```

        System.out.println("K firstKey() : " + ((TreeMap<Integer,
String>) map1).firstKey());

        // SortedMap<K, V> headMap(K end)
        System.out.println("SortedMap<K, V> headMap(K end) : " +
((TreeMap<Integer, String>) map1).headMap(2));

        // K lastKey()
        System.out.println("K lastKey() : " + ((TreeMap<Integer,
String>) map1).lastKey());

        // SortedMap<K, V> subMap(K start, K end)
        System.out
            .println("SortedMap<K, V> subMap(K start, K end) :
" + ((TreeMap<Integer, String>) map1).subMap(1, 3));

        // SortedMap<K, V> tailMap(K start)
        System.out.println("SortedMap<K, V> tailMap(K start) : " +
((TreeMap<Integer, String>) map1).tailMap(2));

        // Map.Entry<K,V> ceilingEntry(K obj)
        System.out.println("Map.Entry<K,V> ceilingEntry(K obj) : "
+ ((TreeMap<Integer, String>) map1).ceilingEntry(2));

        // K ceilingKey(K obj)
        System.out.println("K ceilingKey(K obj) : " +
((TreeMap<Integer, String>) map1).ceilingKey(2));

        // Map.Entry<K,V> floorEntry(K obj)
        System.out.println("Map.Entry<K,V> floorEntry(K obj) : " +
((TreeMap<Integer, String>) map1).floorEntry(2));

        // K floorKey(K obj)
        System.out.println("K floorKey(K obj) : " +
((TreeMap<Integer, String>) map1).floorKey(2));

        // Map.Entry<K,V> higherEntry(K obj)
        System.out.println("Map.Entry<K,V> higherEntry(K obj) : " +
((TreeMap<Integer, String>) map1).higherEntry(2));

        // NavigableSet<K> navigableKeySet()
        System.out
            .println("NavigableSet<K> navigableKeySet() : " +
((TreeMap<Integer, String>) map1).navigableKeySet());

        // Map.Entry<K,V> pollFirstEntry()
        System.out.println("Map.Entry<K,V> pollFirstEntry() : " +
((TreeMap<Integer, String>) map1).pollFirstEntry());

```



```

        // Map.Entry<K,V> pollLastEntry()
        System.out.println("Map.Entry<K,V> pollLastEntry() : " +
((TreeMap<Integer, String>) map1).pollLastEntry());

        // NavigableMap<K,V> subMap(K lowerBound, boolean lowIncl,
K upperBound boolean
        // highIncl)
        System.out.println("NavigableMap<K,V> subMap(K lowerBound,
boolean lowIncl, K upperBound boolean highIncl) : "
+ ((TreeMap<Integer, String>) map1).subMap(1, true,
3, true));

        // NavigableMap<K,V> tailMap(K lowerBound, boolean incl)
        System.out.println("NavigableMap<K,V> tailMap(K lowerBound,
boolean incl) : "
+ ((TreeMap<Integer, String>) map1).tailMap(2,
true));

        // NavigableSet<K> descendingKeySet()
        System.out.println(
            "NavigableSet<K> descendingKeySet() : " +
((TreeMap<Integer, String>) map1).descendingKeySet());

        // NavigableMap<K,V> descendingMap()
        System.out.println("NavigableMap<K,V> descendingMap() : " +
((TreeMap<Integer, String>) map1).descendingMap());

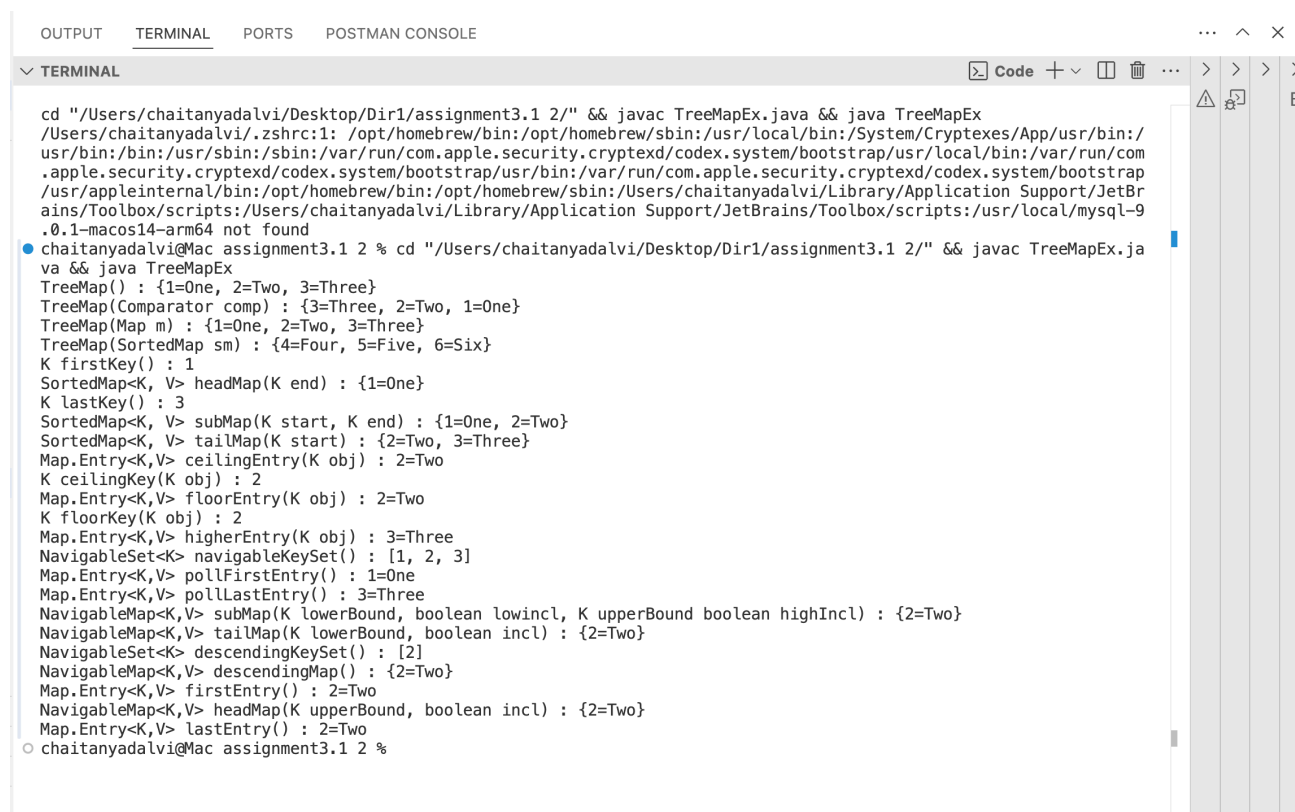
        // Map.Entry<K,V> firstEntry()
        System.out.println("Map.Entry<K,V> firstEntry() : " +
((TreeMap<Integer, String>) map1).firstEntry());

        // NavigableMap<K,V> headMap(K upperBound, boolean incl)
        System.out.println("NavigableMap<K,V> headMap(K upperBound,
boolean incl) : "
+ ((TreeMap<Integer, String>) map1).headMap(2,
true));

        // Map.Entry<K,V> lastEntry()
        System.out.println("Map.Entry<K,V> lastEntry() : " +
((TreeMap<Integer, String>) map1).lastEntry());
    }
}

```

## Output: (screenshot)



```
cd "/Users/chaitanyadalvi/Desktop/Dir1/assignment3.1 2/" && javac TreeMapEx.java && java TreeMapEx
/Users/chaitanyadalvi/.zshrc:1: /opt/homebrew/bin:/opt/homebrew/sbin:/usr/local/bin:/System/Cryptexes/App/usr/bin:/usr/bin:/bin:/usr/sbin:/sbin:/var/run/com.apple.security.cryptexd/codex.system/bootstrap/usr/local/bin:/var/run/com.apple.security.cryptexd/codex.system/bootstrap/usr/bin:/var/run/com.apple.security.cryptexd/codex.system/bootstrap/usr/appleinternal/bin:/opt/homebrew/bin:/opt/homebrew/sbin:/Users/chaitanyadalvi/Library/Application Support/JetBrains/Toolbox/scripts:/Users/chaitanyadalvi/Library/Application Support/JetBrains/Toolbox/scripts:/usr/local/mysql-9.0.1-macos14-arm64 not found
chaitanyadalvi@Mac assignment3.1 2 % cd "/Users/chaitanyadalvi/Desktop/Dir1/assignment3.1 2/" && javac TreeMapEx.java && java TreeMapEx
TreeMap() : {1=One, 2=Two, 3=Three}
TreeMap(Comparator comp) : {3=Three, 2=Two, 1=One}
TreeMap(Map m) : {1=One, 2=Two, 3=Three}
TreeMap(SortedMap sm) : {4=Four, 5=Five, 6=Six}
K firstKey() : 1
SortedMap<K, V> headMap(K end) : {1=One}
K lastKey() : 3
SortedMap<K, V> subMap(K start, K end) : {1=One, 2=Two}
SortedMap<K, V> tailMap(K start) : {2=Two, 3=Three}
Map.Entry<K, V> ceilingEntry(K obj) : 2=Two
K ceilingKey(K obj) : 2
Map.Entry<K, V> floorEntry(K obj) : 2=Two
K floorKey(K obj) : 2
Map.Entry<K, V> higherEntry(K obj) : 3=Three
NavigableSet<K> navigableKeySet() : [1, 2, 3]
Map.Entry<K, V> pollFirstEntry() : 1=One
Map.Entry<K, V> pollLastEntry() : 3=Three
NavigableMap<K, V> subMap(K lowerBound, boolean lowIncl, K upperBound boolean highIncl) : {2=Two}
NavigableMap<K, V> tailMap(K lowerBound, boolean incl) : {2=Two}
NavigableSet<K> descendingKeySet() : [2]
NavigableMap<K, V> descendingMap() : {2=Two}
Map.Entry<K, V> firstEntry() : 2=Two
NavigableMap<K, V> headMap(K upperBound, boolean incl) : {2=Two}
Map.Entry<K, V> lastEntry() : 2=Two
chaitanyadalvi@Mac assignment3.1 2 %
```

**Name of Student:** Chaitanya Dalvi

**Roll Number:** 19

**Experiment No:** 7

---

**Title: Differentiate between Collection and Collections in Java**

Feature	Collection	Collections
Type	Interface	Utility class
Part of	Java Collections Framework	java.util package
Purpose	Defines methods for managing a group of objects (e.g., add, remove)	Provides static methods to manipulate or operate on collections (e.g., sorting, searching)
Usage	Implemented by various collection classes like <code>ArrayList</code> , <code>HashSet</code>	Used to perform utility operations on collections, like <code>Collections.sort()</code> , <code>Collections.shuffle()</code>
Inheritance	Extended by interfaces like <code>List</code> , <code>Set</code> , and <code>Queue</code>	Not extended or implemented by other classes/interfaces
Example	<code>List&lt;String&gt; list = new ArrayList&lt;&gt;();</code>	<code>Collections.sort(list);</code>
Contains Objects?	Can hold objects (like <code>ArrayList</code> , <code>HashSet</code> )	Cannot hold objects, only provides methods

**Name of Student:** Chaitanya Dalvi

**Roll Number:** 19

**Experiment No:** 8

---

**Title:** Q8. Difference between ArrayList and LinkedList in the Java Collection Framework?

Aspect	Collection	Collections
Type	Interface	Utility class
Belongs to	Part of Java's Collections Framework	Part of <code>java.util</code> package
Purpose	Defines the standard methods for working with a group of objects (e.g., adding, removing)	Provides utility methods for operations on collections (e.g., sorting, searching)
Usage	Extended by other interfaces ( <code>List</code> , <code>Set</code> , <code>Queue</code> ) and implemented by classes ( <code>ArrayList</code> , <code>HashSet</code> )	Contains static methods to operate on collections (e.g., <code>sort()</code> , <code>reverse()</code> , <code>shuffle()</code> )
Common Methods	<code>add()</code> , <code>remove()</code> , <code>size()</code> , <code>contains()</code>	<code>sort()</code> , <code>reverse()</code> , <code>shuffle()</code> , <code>min()</code> , <code>max()</code>
Contains Objects	Can store objects in implementations like <code>ArrayList</code> , <code>HashSet</code>	Does not store objects, only provides static utility methods
Inheritance	Extended by interfaces like <code>List</code> , <code>Set</code> , <code>Queue</code>	Not extended or implemented by other classes/interfaces
Example	<pre>List&lt;String&gt; list = new ArrayList&lt;&gt;();</pre>	<pre>Collections.sort(list);</pre>

**Name of Student:** Chaitanya Dalvi

**Roll Number:** 19

**Experiment No:** 9

---

**Title:** Differentiate between List, Set, and Queue in Java Collection framework

Aspect	List	Set	Queue
Type	Interface	Interface	Interface
Allows Duplicates	Yes, can contain duplicate elements	No, duplicate elements are not allowed	No duplicates in some implementations (e.g., <code>PriorityQueue</code> )
Order of Elements	Maintains insertion order	Does not maintain order (except <code>LinkedHashSet</code> )	Elements are ordered by FIFO (First-In-First-Out) or priority
Null Values	Allows multiple <code>null</code> values (depends on implementation)	Allows a single <code>null</code> value (except <code>TreeSet</code> , which doesn't allow <code>null</code> )	Some implementations allow <code>null</code> (e.g., <code>LinkedList</code> ), while others don't (e.g., <code>PriorityQueue</code> )
Common Implementations	<code>ArrayList</code> , <code>LinkedList</code> , <code>Vector</code>	<code>HashSet</code> , <code>LinkedHashSet</code> , <code>TreeSet</code>	<code>LinkedList</code> , <code>PriorityQueue</code> , <code>ArrayDeque</code>
Primary Purpose	Ordered collection for accessing elements by index	Unordered collection to store unique elements	Used for processing elements in a specific order (FIFO, LIFO, or priority)

**Name of Student:** Chaitanya Dalvi

**Roll Number:** 19

**Experiment No:** 10

---

**Title:** Differentiate between Set and Map in Java Collection framework

Aspect	Set	Map
Type	Interface	Interface
Part of	Java Collections Framework ( <code>java.util</code> )	Java Collections Framework ( <code>java.util</code> )
Purpose	Represents a collection of unique elements	Represents a collection of key-value pairs
Duplicates	Does <b>not</b> allow duplicate elements	Keys must be unique; values can be duplicated
Key-Value Pair	No, it only stores individual elements	Yes, stores elements in key-value pairs
Null Values	Allows at most one <code>null</code> element (for some implementations like <code>HashSet</code> )	Allows one <code>null</code> key and multiple <code>null</code> values (in <code>HashMap</code> )
Common Implementations	<code>HashSet</code> , <code>LinkedHashSet</code> , <code>TreeSet</code>	<code>HashMap</code> , <code>LinkedHashMap</code> , <code>TreeMap</code>
Ordering	Some implementations ( <code>LinkedHashSet</code> , <code>TreeSet</code> ) maintain insertion or sorted order	Some implementations ( <code>LinkedHashMap</code> , <code>TreeMap</code> ) maintain insertion or sorted order of keys
Example Usage	<pre>Set&lt;String&gt; set = new HashSet&lt;&gt;();</pre>	<pre>Map&lt;String, Integer&gt; map = new HashMap&lt;&gt;();</pre>

**Name of Student:** Chaitanya Dalvi

**Roll Number:** 19

**Experiment No:** 11

---

**Title:** Differentiate between Iterator and ListIterator.

Aspect	Iterator	ListIterator
Type	Interface	Interface (extends <code>Iterator</code> )
Applicable To	Works with all collection types ( <code>Set</code> , <code>List</code> , <code>Map</code> via <code>keySet</code> , <code>values</code> , or <code>entrySet</code> )	Works only with <code>List</code> (e.g., <code>ArrayList</code> , <code>LinkedList</code> )
Traversal Direction	Only forward (one-way traversal)	Both forward and backward (two-way traversal)
Traversal Methods	<code>hasNext()</code> , <code>next()</code> , <code>remove()</code>	<code>hasNext()</code> , <code>next()</code> , <code>hasPrevious()</code> , <code>previous()</code> , <code>remove()</code> , <code>add()</code> , <code>set()</code>
Add/Modify Elements	Cannot add or modify elements (can only remove)	Can add, remove, or modify elements during iteration ( <code>add()</code> , <code>set()</code> )
Starting Position	Always starts at the beginning of the collection	Can start at any position in the list (via <code>ListIterator(int index)</code> )
Backward Traversal	Not supported	Supported with <code>hasPrevious()</code> and <code>previous()</code> methods
Index Access	No access to index of the elements during iteration	Provides access to the index of elements using <code>nextIndex()</code> and <code>previousIndex()</code>
Usage Scenario	Used for basic iteration when only forward traversal is needed	Used when bi-directional iteration or element modification is needed during traversal

**Name of Student:** Chaitanya Dalvi

**Roll Number:** 19

**Experiment No:** 12

---

**Title:** Differentiate between TreeSet and TreeMap.

Aspect	TreeSet	TreeMap
Type	Class implementing <code>Set</code> interface	Class implementing <code>Map</code> interface
Key-Value Pair	Stores only unique elements (no key-value pairs)	Stores elements in key-value pairs (keys must be unique)
Underlying Data Structure	Based on a Red-Black Tree	Based on a Red-Black Tree
Order	Elements are sorted in natural order or using a custom <code>Comparator</code>	Keys are sorted in natural order or using a custom <code>Comparator</code>
Duplicates	Does not allow duplicate elements	Does not allow duplicate keys (values can be duplicated)
Null Handling	Does not allow <code>null</code> elements	Does not allow <code>null</code> keys, but can have <code>null</code> values
Traversal	Supports sorted traversal of elements	Supports sorted traversal of keys
Common Methods	<code>add()</code> , <code>remove()</code> , <code>contains()</code> , <code>first()</code> , <code>last()</code>	<code>put()</code> , <code>get()</code> , <code>remove()</code> , <code>firstKey()</code> , <code>lastKey()</code> , <code>values()</code> , <code>keySet()</code>
Performance	$O(\log n)$ time complexity for basic operations (e.g., add, remove, contains)	$O(\log n)$ time complexity for operations like put, get, and remove



