

Collections Day 2

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. Consider the following example.

```
import java.util.*; import java.io.*; import java.lang.*; public class
Main { public static void main(String[] args) { Stack<Integer> stack =
new Stack<>(); stack.push(1); stack.push(2); stack.push(3);
System.out.println(stack.peek()); // returns the top elements of the
stack without removing. stack.pop(); // deletes the element at the top
of the stack. System.out.println(stack.isEmpty()); // returns true if the
stack is empty otherwise false. Iterator itr = stack.iterator(); //
iterator for stack. while(itr.hasNext()){
System.out.println(itr.next()); } System.out.println(stack.size());
//size of the stack. } }
```

Java ▾

Output : 3 false 1 2 2

Java ▾

Time Complexity

⚙ Operation	⌵ Time Complexity
push()	O(1)
pop()	O(1)
peek()	O(1)
isEmpty()	O(1)

COUNT 4

Queue Interface

- 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.

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. Consider the following example.

```
import java.util.*; import java.io.*; import java.lang.*; public class
Main { public static void main(String[] args) { LinkedList<String> list
= new LinkedList<>(); list.add("ankush"); list.add("nnknk");
list.add("kkfko"); for (String i : list){ System.out.print(i + " "); }
System.out.println(); list.remove(); // removes from the head. for
(String i : list){ System.out.print(i + " "); } System.out.println();
list.removeLast(); // removes from the tail. for (String i : list){
System.out.print(i + " "); } } }
```

Java ▾

Output : ankush nnknk kkfko nnknk kkfko nnknk

Java ▾

Set Interface

Set Interface in Java 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.

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.



```
import java.util.*; class Main{ public static void main(String args[]){
HashSet<Integer> map = new HashSet<>(); map.add(1); //function to add
values in the hashset. map.add(2); map.add(3); map.add(0);
map.add(1); //duplicate values are discarded.
System.out.println(map.contains(1)); //function to check if a particular
element is present in the set or not. map.remove(1); //function to remove
element from the set. System.out.println(map.contains(1));
map.remove(4); Iterator itr = map.iterator(); //iterator for the set.
while(itr.hasNext()){ System.out.print(itr.next() + "\t"); }
System.out.print("\n"); System.out.println(map.size()); //size of the
set. } }
```

Java ▾

Output : true false 3 2 0 2

Java ▾

Time Complexity

 Operation	 Time Complexity
<u>add()</u>	O(1)
<u>remove()</u>	O(1)
<u>contains()</u>	O(1)
<u>size()</u>	O(1)

COUNT 4

LinkedHashSet

- The LinkedHashSet is an ordered version of HashSet that maintains a doubly-linked List across all elements. When the iteration order is needed to be maintained this class is used.

- When iterating through a HashSet the order is unpredictable, while a LinkedHashSet lets us iterate through the elements in the order in which they were inserted.
- When cycling through LinkedHashSet using an iterator, the elements will be returned in the order in which they were inserted.

```
import java.util.*; class Main{ public static void main(String args[]){
LinkedHashSet<Integer> map = new LinkedHashSet<>(); map.add(1);//function
to add values in the hashset. map.add(2); map.add(3); map.add(0);
map.add(1);//duplicate values are discarded.
System.out.println(map.contains(1));//function to check if a particular
element is present in the set or not. map.remove(1);//function to remove
element from the set. System.out.println(map.contains(1));
map.remove(4); Iterator itr = map.iterator(); //iterator for the set.
while(itr.hasNext()){ System.out.print(itr.next() + "\t"); }
System.out.print("\n"); System.out.println(map.size());//size of the
set. } }
```

Java ▾

Output : true false 2 3 0 3

Java ▾

Time Complexity

Operation	Time Complexity
add()	O(1)
remove()	O(1)
contains()	O(1)
size()	O(1)

COUNT 4

SortedSet

- The SortedSet interface present in java.util package extends the Set interface present in the collection framework. It is an interface that implements the mathematical set.

- This interface contains the methods inherited from the Set interface and adds a feature that stores all the elements in this interface to be stored in a sorted manner.

TreeSet

- TreeSet is one of the most important implementations of the SortedSet interface in Java. The ordering of the elements is maintained by a set using their natural ordering.
- It can also be ordered by a Comparator provided at set creation time, depending on which constructor is used.

```
import java.util.*; class Main{ public static void main(String args[]){
TreeSet<Integer> map = new TreeSet<>(); map.add(1); //adding values into
the set. map.add(0); map.add(2); map.add(3); map.add(4);
System.out.println(map.contains(1)); // checking if a particular value
exists in the set; map.remove(1); // removing an element in a set.
Iterator itr = map.iterator(); // iterator for set. while(itr.hasNext())
{ System.out.print(itr.next() + "\t"); } System.out.print("\n");
System.out.println(map.size()); // size of set. } }
```

Java ▾

Output : true 0 2 3 4 4

Java ▾

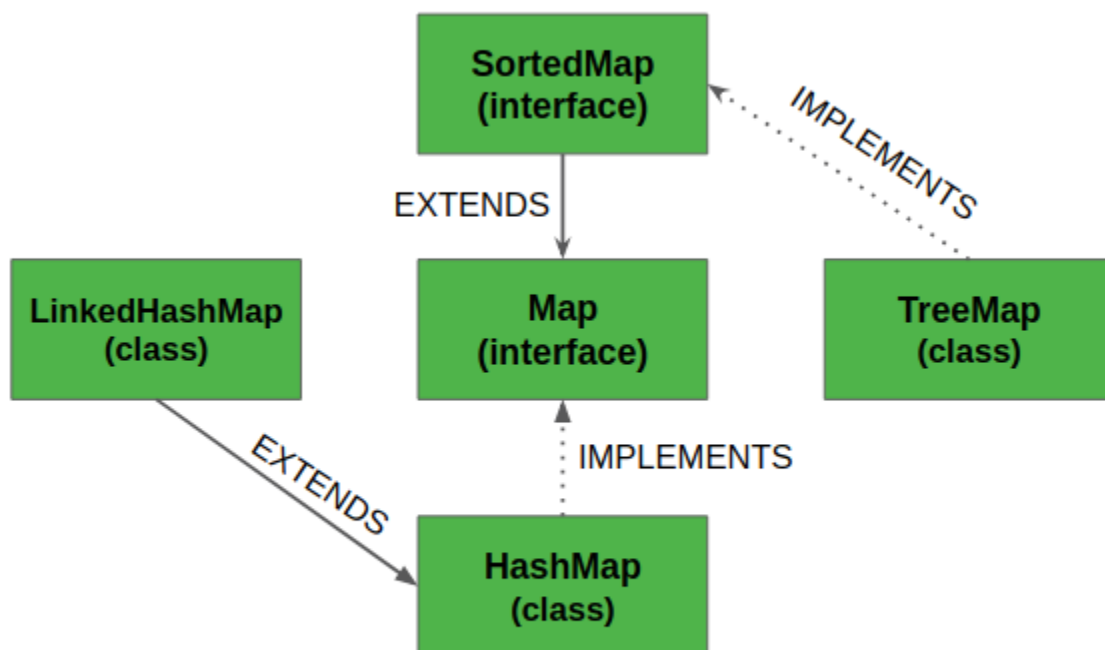
Time Complexity

Operation	Time Complexity
add()	O(log(n))
remove()	O(log(n))
contains()	O(log(n))
size()	O(1)

COUNT 4

Map

- The Map interface present in java.util package represents a mapping between a key and a value.
- The Map interface is not a sub type of the Collection interface. Therefore it behaves a bit differently from the rest of the collection types.
- A map contains unique keys. It behaves as an unordered keys as order in which the keys are entered is not maintained.



MAP Hierarchy in Java

HashMap

- HashMap provides the basic implementation of the Map interface of Java. It stores the data in (Key, Value) pairs.
- To access a value one must know its key. This class uses a technique called Hashing.
- Hashing is a technique of converting a large String to small String that represents the same String. A shorter value helps in indexing and faster searches.

- The put(key,pair) function is used to insert values into the HashMap, and the get(key) function is used to access value for a particular key. The remove(key) function is used to remove a particular key value pair.

```
import java.util.*; import java.io.*; import java.lang.*; public class
Main { public static void main(String[] args) { HashMap<Integer,Integer>
map = new HashMap<>(); map.put(1,1);//adding values in a hashmap.
map.put(2,2); map.put(3,3); for (Integer i : map.keySet()){ // for each
loop to iterate through the map. System.out.println(i + "\t" +
map.get(i)); } map.remove(1); // remove a particular key-value pair. for
(Integer i : map.keySet()){ System.out.println(i + "\t" + map.get(i)); }
map.put(3,4);//Duplicate keys are not allowed. The key is mapped to the
most recent value. for (Integer i : map.keySet()){ System.out.println(i
+ "\t" + map.get(i)); } System.out.println(map.containsKey(1));//checks
if the map contains a particular key.
System.out.println(map.containsValue(3));//checks if the map contains a
particular value. } }
```

Java ▾

Output : 1 1 2 2 3 3 2 2 3 3 2 2 3 4 false true

Java ▾

- Custom classes can also be used as keys, and are quite helpful in a lot of different scenarios. The unique key property holds true in all conditions.
- The function keySet() returns the entire set of keys in the map. The function values() returns the entire value set in the HashMap.



```
import java.util.*; import java.io.*; import java.lang.*; public class
Main { static class Pair{ int x; int y; Pair(int x,int y){ this.x = x;
this.y = y; } } public static void main(String[] args) {
HashMap<Pair,Integer> map = new HashMap<>(); map.put(new Pair(1,2),1);
map.put(new Pair(2,1),2); map.put(new Pair(2,3),3); map.put(new
Pair(3,2),4); for (Pair i : map.keySet()){ System.out.println(i.x + " "
+ i.y); } System.out.println(map.values()); } }
```

Java ▾

Output : 2 3 1 2 2 1 3 2 [3, 1, 2, 4]

Java ▾

Time Complexity

 Operations	 Time Complexity
<code>put()</code>	O(1)
<code>containsKey()</code>	O(1)
<code>get()</code>	O(1)
<code>containsValue()</code>	O(N)
<code>size()</code>	O(1)

COUNT 5

LinkedHashMap

- The LinkedHashMap is just like HashMap with an additional feature of maintaining an order of elements inserted into it.
- HashMap provided the advantage of quick insertion, search, and deletion but it never maintained the track and order of insertion which the LinkedHashMap provides where the elements can be accessed in their insertion order.
- LinkedHashMap extends the HashMap class therefore supports all its functions.

```
import java.util.*; import java.io.*; import java.lang.*; public class
Main { static class Pair{ int x; int y; Pair(int x,int y){ this.x = x;
this.y = y; } } public static void main(String[] args) {
HashMap<Pair,Integer> map = new LinkedHashMap<>(); map.put(new
Pair(1,2),1); map.put(new Pair(2,1),2); map.put(new Pair(3,1),3);
map.put(new Pair(1,3),1); map.put(new Pair(1,4),5); for (Pair i :
map.keySet()){ System.out.println(i.x + "\t" + i.y + "\t" + map.get(i));
} } }
```

Java ▾

Output : 1 2 1 2 1 2 3 1 3 1 3 1 1 4 5

Java ▾

Time Complexity

 Operation	 Time Complexity
<code>put()</code>	$O(1)$
<code>get()</code>	$O(1)$
<code>containsKey()</code>	$O(1)$
<code>containsValue()</code>	$O(N)$
<code>size()</code>	$O(1)$

COUNT 5

SortedMap

- SortedMap is an interface in the collection framework. This interface extends the Map interface and provides a total ordering of its elements.
- The elements can be traversed in sorted order of keys.
- The class that implements this interface is TreeMap.

TreeMap

- The map is sorted according to the natural ordering of its keys, or by a Comparator provided at map creation time, depending on which constructor is used.
- This proves to be an efficient way of sorting and storing the key-value pairs. The storing order maintained by the TreeMap must be consistent with equals just like any other sorted map, irrespective of the explicit comparators. Consider the example below.



```
import java.util.*; import java.io.*; public class Main { public static
void main(String[] args) { TreeMap<Integer,Integer> map = new TreeMap<>
(); map.put(1,1); map.put(3,2); map.put(2,3); map.put(4,8);
map.put(0,1); for (Integer i : map.keySet()){ System.out.println(i +
"\t" + map.get(i)); } } }
```

Java ▾

Output : 0 1 1 1 2 3 3 2 4 8

Java ▾

Time Complexity

 Operation	 Time Complexity
<u>put()</u>	$O(\log(N))$
<u>get()</u>	$O(\log(N))$
<u>containsKey()</u>	$O(\log(N))$
<u>containsValue()</u>	$O(N)$
<u>size()</u>	$O(1)$

COUNT 5

- In a lot of scenarios, a custom comparator, is quite useful.

