SortedSet Interface

- It is a subinterface of set which ensures elements are stored in sorted order.

- Key Features:

1. Maintains elements in ascending order by default.
2. Does not allow duplicate elements

TreeSet Class

- A class which implements the sortedSet interface.

- Characteristics:

- Thread unsafe: Requires external synchronization in multithreaded environment.
- Elements are stored in ascending order.
- Provides methods to retrieve elements based on position or range.

```
- package sortedSetExamples;

  import java.util.TreeSet;

  public class TreeSetExample {
      public static void main(String[] args) {
          //creating a treeset
          TreeSet<Integer> treeSet = new TreeSet<>();

          //Adding elements
          treeSet.add(60);
          treeSet.add(40);
          treeSet.add(30);
```

```
        treeSet.add(10);

        //Displaying the treeSet
        System.out.println("TreeSet: "+ treeSet);

        //1.Get first element(smallest)
        System.out.println("First Element: "+
treeSet.first());

        //2.Get the last(largest) element
        System.out.println("Last Element: "+
treeSet.last());

        //3. Get elements less than 30
        System.out.println("Elements less than 30: "+
treeSet.headSet(30));

        //4. Get elements greater than or equal to 30
        System.out.println("Elements greater than or
equal to 30: "+ treeSet.tailSet(30));

    }
}
```

O/P:

TreeSet: [10, 30, 40, 60]

First Element: 10

Last Element: 60

Elements less than 30: [10]

Elements greater than or equal to 30: [30, 40, 60]

FAQs:

1.Can we add null elements in treeset?

-> NO, adding null throws NullPointerException.

2. What happens if we add duplicate element?
-> Duplicate elements are ignored.

Queue Interface

- It follows FIFO(First in first out) principle
- Elements are inserted from the rear and removed from front.

PriorityQueue Class

- A class which implements the queue interface and maintains elements in their natural order.

Characteristics:

- Not thread-safe by default

```
package queueExample;

import java.util.PriorityQueue;

public class PriorityQueueExample {
    public static void main(String[] args) {
        PriorityQueue<Integer> pq = new PriorityQueue<>();

        //adding elements
        pq.add(40);
        pq.add(30);
        pq.add(10);
        pq.add(20);

        System.out.println("Priority Queue: "+ pq);

        // Accessing the highest priority element
        System.out.println("Highest Priority Element:
"+pq.peek());
```

```
        // Removing the highest priority element
        System.out.println("Polling: "+ pq.poll());

        //Displaying the priority queue
        System.out.println("Priority Queue after polling: "+
pq);

        //To get the size
        System.out.println("Size: "+ pq.size());
    }
}
```

FAQs:

1. Can PriorityQueue contain null values?
   ➔ No, It throws NullPointerException

2. Is PriorityQueue thread safe?
   ➔ Not thread safe

3. What is the default order of elements in a PriorityQueue?
   ➔ Elements are ordered in ascending order by default.

Map Interface

A collection which is used to store key-value pairs.

Characteristics:

- Each key is unique, but values can be duplicated.
- Allows efficient retrieval of data using a key.

Common Implementations:

1. HashTable
2. HashMap
3. LinkedHasMap
4. TreeMap

HashTable

It is synchronized and thread-safe, which makes it suitable for multithreaded environments.

Characteristics:

- It Does not allow null keys or values
- Performance is slower than hashmap due synchronization.

```java
package mapExample;

import java.util.Hashtable;

public class HashTableExample {
    public static void main(String[] args) {
        //Creating a hashtable
        Hashtable<Integer,String> table = new
Hashtable<>();

        //adding elements
        table.put(1,"JAVA");
        table.put(2,"Python");
        table.put(3,"C#");

        //Displaying Hashtable
        System.out.println("HashTable: "+ table);

        //Retrieve a value by key
        System.out.println("Value of 3rd Key: "+
table.get(3));

        //Check if key exists or not
        System.out.println("Contains key 3? : "+
table.containsKey(3));

        //To remove the key-value pair
        table.remove(3);

        //Displaying table after removal
        System.out.println("After Removal: "+ table);

    }
}
```

FAQs:

1. Can we store null keys or values in HashTable?

➔ No, Does not allow null keys or values.

2. Is HashTable synchronized?

➔ Yes, It is synchronized and thread-safe.

HashMap

HashMap is an unsynchronized map which allows one null key and multiple null values.

Characteristics:

- Faster than hashtable in single threaded environments
- Does not maintain order of elements.

```
package mapExample;

import java.util.HashMap;

public class HashMapExample {
    public static void main(String[] args) {
        //Creating a hashmap
        HashMap<String,Integer> map = new HashMap<>();

        //Adding key-value pairs
        map.put("Apple",100);
        map.put("Banana",50);
        map.put("PineApple", 75);

        //Retrieve value by key
        System.out.println("Price of apple: "+
map.get("Apple"));

        //Check if key exists
        System.out.println("Does cart contain banana? : "+
map.containsKey("Banana"));

        //Get all values
        System.out.println("Values: "+ map.values());

        //To Retrieve All Key-Value Pairs
        System.out.println("All Values: "+ map.entrySet());
    }
}
```

HashMap (Java SE 21 & JDK 21)

FAQs:

Can hashMap store Null keys or values?

- ➔ Yes, It allows one null key and multiple null values.

Is HashMap Synchronized?

- ➔ No, It is not synchronized.

SMS with HashMap