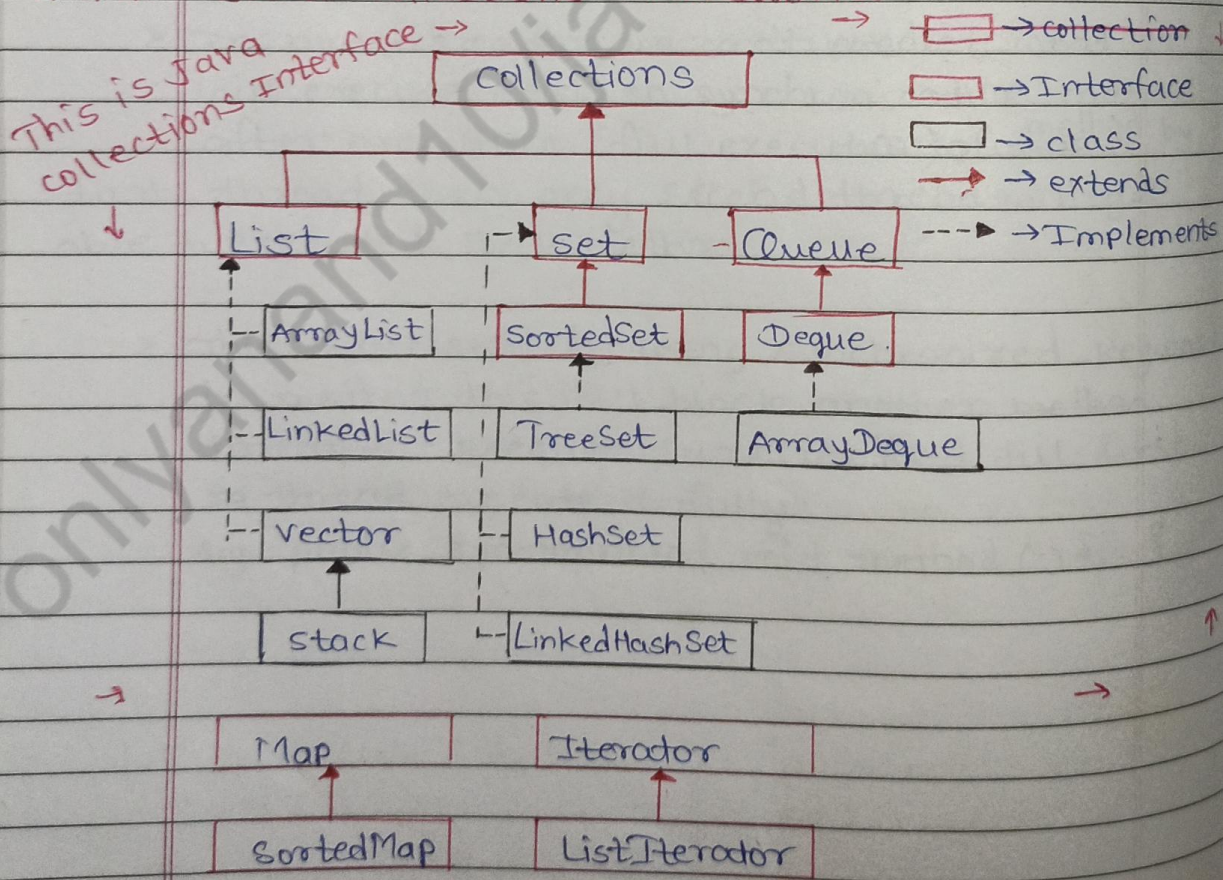
Collections Framework.

- Let's understand little bit about this
In java collections framework provides a set of interfaces and classes to implement various datastructures and algorithms.
For eg.:- The LinkedList class of the collections framework provides the implementation of the doubly-linked list data structure.

- Interfaces of collections framework.
  - The java collections framework provides various interfaces. These interfaces include several methods to perform different operations on collections.

*** Java collections framework.



✱ Java collections interface.

> The collection interface is the root interface of the collections framework hierarchy.

> Java does not provide direct implementation of the Collection interface but provides implementation of its subinterfaces like List, Set & Queue

✱ Collections Framework Vs Collection Interface.

> The collection interface is the root interface of the collections framework.

> The collection framework includes interfaces like collection, Map and Iterator. These interfaces may also have subinterfaces.

✱ Subinterfaces of Collection.

1) List Interface.

> The List interface is an ordered interface/coll^{n} that allows us to add and remove elements like an array.

2) Set Interface

> The Set Interface allows us to store elements in different sets similar to the set in maths.

> We can't store duplicate elements in Set.

3. Queue Interface.

The Queue Interface is used when we want to store elements in First In First Out (FIFO) manner.

▶ Methods of collection.

> The collection interface provides various methods, these methods are avalible in all subinterfaces.

> Some methods.

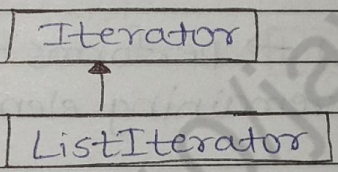
- add() → inserts the specified element to the collection.
- size() → returns the size of collection.
- remove() → removes specified element.
- iterator() → returns an iterator to access elements of the collection.
- removeAll() → removes all the elements of the specified collection from the collection.
- addAll() → adds all the specified collection elements to the collection.
- clear() → removes all the elements of the collection.

Java Map Interface.

> Map interface allows elements to be stored in key/pair key/value pairs.
> keys are the unique names than can be used to access a perticular element in a map
> Each key has a single value associated with it.

Java Iterator Interface.

> Iterator interface provides methods that can be used to access elements of collections.
> It has subinterface ListIterator.



> All Java collections include an iterator() method. This method returns an instance/object of iterator used to iterate over elements of collections.

```
class Main {
    psvm(String[] args) {

  // creating ArrayList
ArrayList<Integer> num = new ArrayList<>();

  // Creating an instance/object of Iterator.
Iterator<Integer> iterate = num.iterator();

    }
}
```

▶ Methods of Iterator.

> The Iterator interface provides 4 methods those are as follow

- hasNext() → return true if there exists an element in the collection.
- next() → return the next element of the collection.
- remove() → removes the last element returned by the next()
- forEachRemaining() → performs the specified action for each remaining element of the collection.

---

* Why the collections Framework?

> We do not have to write code to implement these data structures and algorithms manually.
> Our code will be much more efficient as the collections framework is highly optimized.
> The collections framework allows us to use a specific data structure for a particular type of data. Like
  - If we want our data to be unique, then we can use the `Set` interface provided by the collections framework.
  - To store data in key/value pairs, we can

use the Map interface.

- The ArrayList class provides the functionality of resizable arrays.

> Let's see subclasses of Collection Interface subinterface List interface.

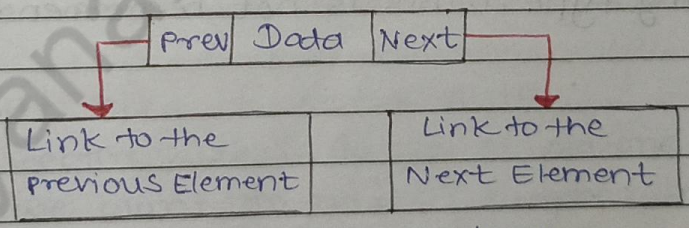
Subclasses of List Interface.

⇒ Java ArrayList Class

* The ArrayList class of java collections framework provides the functionality of resizable-arrays.
* It implements the List interface.

⇒ Java LinkedList

* The LinkedList class provides the functionality of the linked list data structure (doubly linkedlist)



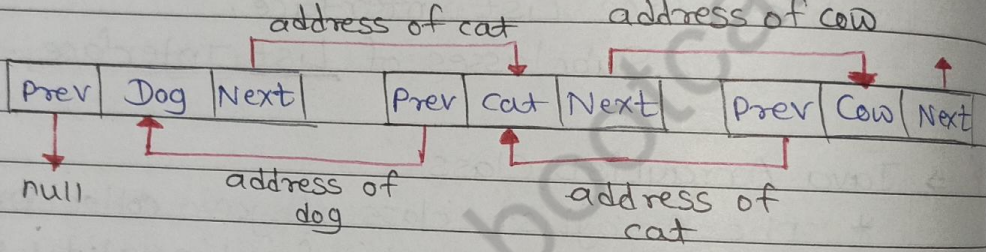
Java Doubly LinkedList

* Each element in a LinkedList is known as Node. It have 3 parts/fields:

- Prev ⇒ stores an address of previous element in the list. It is null for the first element.
- Next ⇒ stores an address of next element in the list. It is null for the last element.
- Data ⇒ stores actual data.

> Working of LinkedList

- Elements in LinkedList are not stored in sequence like array.
- They are scattered and connected through links (Prev and Next)



Java LinkedList Implementation

> Creating a Java LinkedList

```
LinkedList<Type> linkedList = new LinkedList<>();
```

* If we don't specify type we can store data of any type. (This is not recommended)

Java Vector

> The Vector class is an implementation of the List interface that allows us to create resizable-arrays similar to the ArrayList class.

* Vector vs ArrayList

> The vector class synchronizes each individual operation,
> However in Arraylist methods are not synchronized.

Note: It is recommended to use ArrayList in place of Vector because vectors are less efficient.

* Creating a vector.

```
Vector<Type> vector = new Vector< >( );
```

Here we can specify initialsize as well as capacityIncrement
This saves memory wastage.

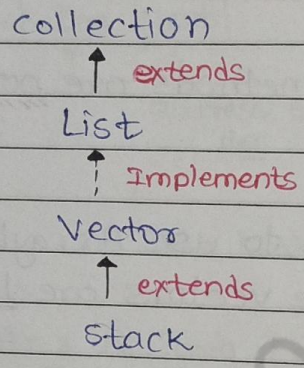
> Default size of vector is 10
> New vector size is double of old one,
If we want to add 11th member in vector the new Vector is created of size oldsize X 2
⇒ 10 X 2 ⇒ 20

> In vector we can check capacity by capacity()

✡ Java Stack Class.

> The Java collections framework has a class named stack that provides a functionality of stack data structure.

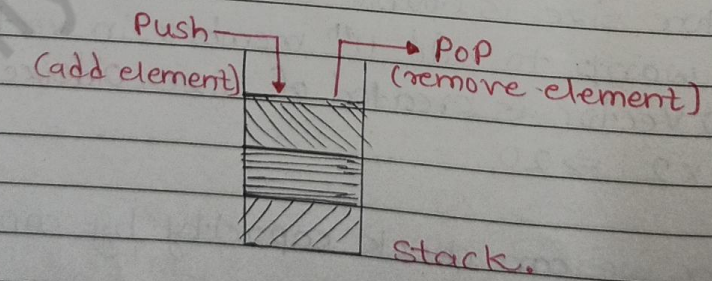
> The stack class extends the vector class.



* Stack implementation:

> In stack, elements are stored and accessed in Last In First Out manner.

> Elements are added to the top of the stack and removed from the top of the stack.



✡ Creating Stack

```
Stack<Type> stacks = new stack<>();
```

! Note: need to import java.util.stack or java.util.*;

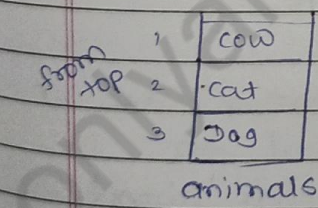
Stack Methods

> As stack extends the vector class, it inherits all the methods of vector.

> Stack class have some unique methods like

- push() → used to add an element to the top of the stack.
- pop() → used to pop an element from the top of the stack.
- peek() → used to get an object/element from the top of the stack.
- search() → used to search an element in the stack.

> It returns the position of the element from the top of the stack.




```
animals.search("cat");
// 2 will be output.
```

- empty() → used to check whether stack is empty or not.

> It returns boolean value (true or false)

> Default capacity of stack is 10

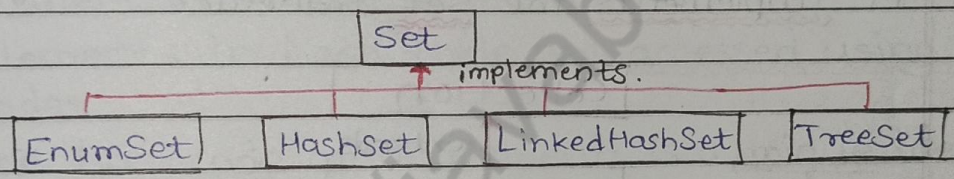
> If we add 11th element then new Stack is formed internally.

Memory by oldSize*2 ⇒ 10*2 ⇒ 20

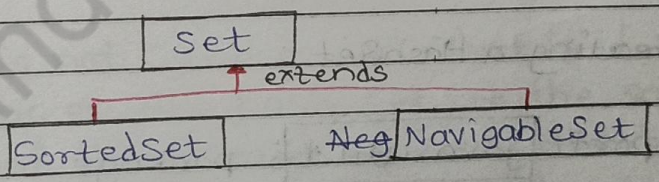
Java Set Interface

> The Set interface of the java Collections framework provides the features of the mathematical set in Java.
> Set interface extends the Collection interface.
> Set can't contain duplicate elements.
> Insertion order is not preserved.

* Classes that implement Set.



* Interfaces that extends Set.



* Set Operations.

The Java Set interface allows us to perform basic mathematical set operations like union, intersection and subset.

- Union ⇒ to get the union of two sets x and y, we can use x.addAll(y)
- Intersection ⇒ to get the intersection of two sets

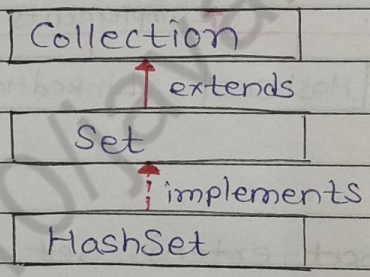
x and y, we can use x.retainAll(y)

- Subset ⇒ to check if x is a subset of y, we can use y.containsAll(x)

✱ Java HashSet class

> The HashSet class of the Java collections framework provides the functionalities of the hash table data structure.

> It implements the set Interface.



✱ Creating a HashSet

> Import java.util.*; package

> HashSet<Integer> num = new HashSet<>();

or

HashSet<Integer> num = new HashSet<>(8, 0.70);

InitialCapacity ← → loadfactor

- InitialCapacity : Initial capacity of HashSet means it can store 8 elements initially.

Default InitialCapacity is 16

- loadFactor : The load-factor here is 0.70 it means as the first HashSet get 70% filled new Hashset will get created or new memory will get allocated.

* Default loadFactor is 0.75

* About HashSet

> Insertion order is not preserved here.

> Hashset is commonly used if we have to access elements randomly.

> Elements in hash table are accessed using hash codes.

> The hashcode of an element is a unique identity that helps to identify the element in a hash table.

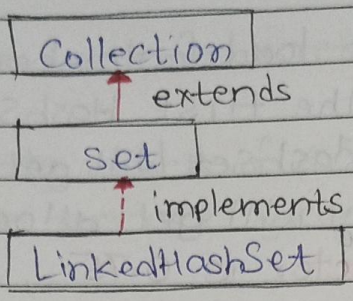
> HashSet can't contain duplicate elements. Hence, each hash set element has a unique hashcode.

Note → HashSet is not synchronized. That is if multiple threads access the hash set at the same time and one of the threads modifies the hash set. Then it must be externally synchronized.

* Java LinkedHashSet class

> The LinkedHashSet class of the Java collections framework provides functionalities of both the hashtable and linked list data structure.

> It implements Set interface.



> Elements of LinkedHastSet are stored in hash tables Similar to HashSet.

> LinkedHastSet maintain a doubly-linked list internally for all of its elements. The linked list defines the order in which elements are inserted in hash tables.

* Create a LinkedHashSet

> Import the java.util.* Package

> LinkedHashSet<Integer> num = new LinkedHashSet<>(8,0.75)
8 is initial capacity default capacity is 16 &
0.75 is load factor default load factor is 0.75

* LinkedHashSet Vs. HashSet
As both these implements set interface.

> LinkedHashSet maintains a linkedlist internally due to this, it maintain the insertion order of its elements.

> The LinkedHashSet class requires more storage than HashSet. This is because LinkedHashSet

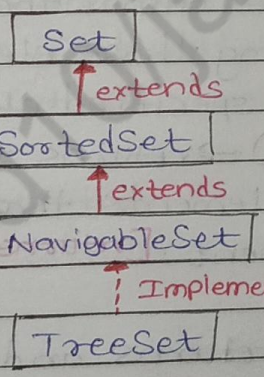
maintains linked list internally.

> The performance of LinkedHashSet internally is slower than HashSet, because of linked lists present in LinkedHashSet.

✱ Java TreeSet Class

> The TreeSet class of Java collections framework provides the functionality of a tree data structure.

> It extends the NavigableSet interface



Creating a TreeSet

> import java.util.*;

> TreeSet<Type> numbers=new TreeSet<>();

! Here we have created a Treeset without any arguments. In this case, the elements in TreeSet are sorted naturally (ascending order).

Comparator

> We can customize sorting using Comparator interface.
- Using comparator we can also customize the ordering of elements.
- We can create own methods as well as we can use predefined methods like reverseOrder() which will store/show elements in reverse order.

```
TreeSet<Type> num = new TreeSet<>(Comparator.reverseOrder());
```

* Some Methods of TreeSet

*TreeSet class implements NavigableSet, so these are methods :-

> first() → returns the first element of the set
> last() → returns the last element of the set
> higher(element) → returns the ~~greatest~~ lowest element among those elements that are greater than specified element [closest higher number]
> lower(element) → returns the greatest element among those elements that are less than the specified element. [returns closest lower number]

> pollfirst() → returns and removes the first element from the set.
> pollLast() → returns and removes the last element from the set.

> subSet(e1, bV1, e2, bV2)
- The subSet() method returns the elements between e1 and e2, including e1.
- The bV1 and bV2 are optional parameters by default bV1 is True and bV2 is false.
- If false is passed as bV1, then method

returns all the elements between e1 and e2 without including e1.
- If ~~false~~ true is passed as bV2 the method will returns all the elements between e1 and e2 without excluding e2. (including e2)

TreeSet vs HashSet

> Both implements hash interface.
> Unlike Hashset, elements in Treeset are stored in some order. It is because TreeSet implements the SortedSet interface as well.
> Treeset provides some methods for easy navigation like first(), last(), headSet(), tailSet(), etc. It is because TreeSet also implements the NavigableSet interface.
> HashSet is faster than the TreeSet for basic operations like add, remove, contains and size.

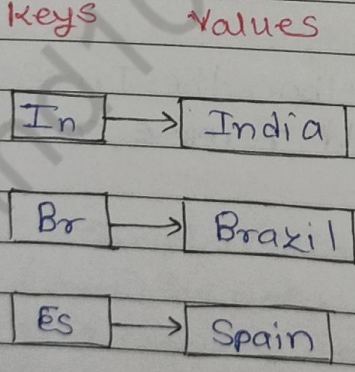
Java Map Interface.

The Map interface of the Java collections framework provides the functionality of the map data structure.

* Working of Map.

> Elements of Map are stored in key/value pairs. keys are unique values which are associated with individual values.

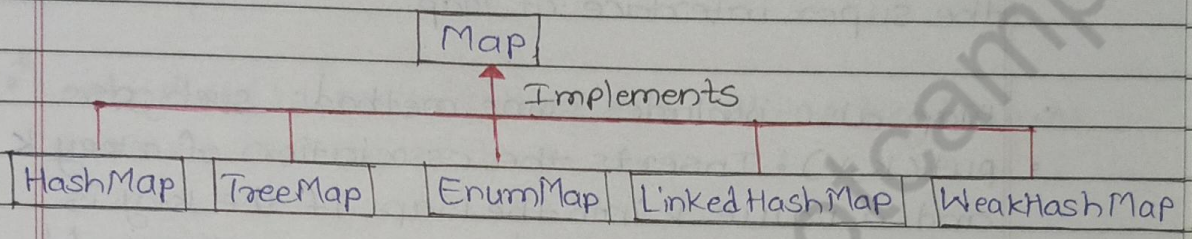
> A map can't contain duplicate keys. And, each key is associated with a single value.



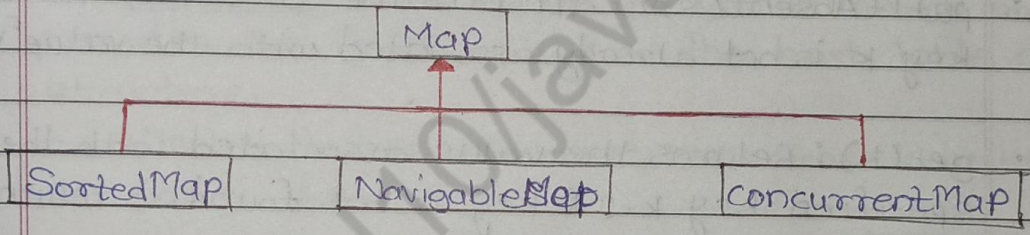
> We can access and modify values using the keys associated with them.

> Note: The map interface maintains 3 diff. sets
  > The set of keys
  > The set of values
  > The set of key/value associations (mapping)
Hence we can access them individually.

> Map is an interface we can't create it's objects. In order to implement it we can use some classes of its.



* Interfaces that extend Map.



* Map usage

> Import java.util.Map or java.util.* package.

> Map<key,value> numbers = new HashMap<>();
- we have created map named numbers. We have used the HashMap class to implement the Map interface.

Here,

- key → a unique identifier used to associate each element (value) in a map.
- Value → elements associated by keys in a map.

✱ Method of Map.

> The Map interface includes all the methods of the collection interface. Because collection is the super interface of Map.

> Map also includes some methods such as

- put(K,V): Inserts the association of a key K and a value V into the Map. If the key is already present, the new value replaces old value.

- putIfAbsent(K,V): Inserts the association if the key K is not already associated with the value 'V'

- get(K): Returns the value associated with the specified key K. If key is not found, it returns `null`

- replace(K,V): Replace the value of key K with the new specified value V

- replace(K, oldValue, newValue): Replaces the value of the key K with the new value `newValue` only if the key `K` is associated with the value `oldValue`

- remove(K): Removes entry from the map associated with the key `K`.

- remove(K,V): Removes the entry from the map that has key `K` associated with value `V`

* keySet(): Returns the set of all the keys present in map.

* values(): Returns the set of all the values present in map.

* entrySet(): Returns the set of all the key/values mapping present in map.

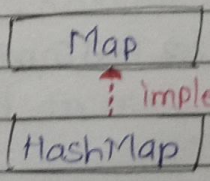
* Java HashMap Set.

> The [HashMap] class of the java collections framework provides the functionality of the hash table data structure.
  - Hash table data structure stores elements in key-value pairs where
    - key: unique integer that is used for indexing the values.
    - value: data that are stored with keys.

Key	Value

  - Same way values are stored map.

>
> The [HashMap] class implements the Map interface.



Create a HashMap

> Import necessary packages
import java.util.* or HashMap

> HashMap<K,V> numbers = new HashMap<>();
HashMap name → numbers
K → key type (String, Integer, etc.)
V → value type (String, Integer, etc.)
Def. capacity → 8*2 → 16
Def. load factor → 0.75

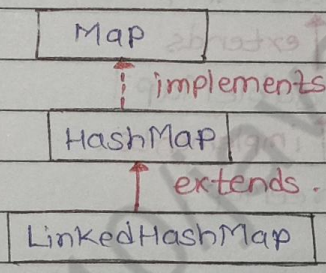
Ex.
```
import java.util.*;

class Main{
    public static void main(String[] args){
        HashMap<Integer, String> first = new HashMap<>();
        // put() method to add elements.
        first.put(1, "Java");
        first.put(2, "JS");
        first.put(3, "python");
        System.out.print(first);
    }
}
```

O/P = {1 = Java, 2 = JS, 3 = Python}

Java LinkedHashMap class.

> The LinkedHashMap class of the Java collections framework provides the hash table and linked list implementation of the Map interface.

> The LinkedHashMap interface extends the HashMap class to store its entries in a hash table. It internally maintains a doubly linked list among all of its entries to order its entries.



⊛ Creating LinkedHashMap

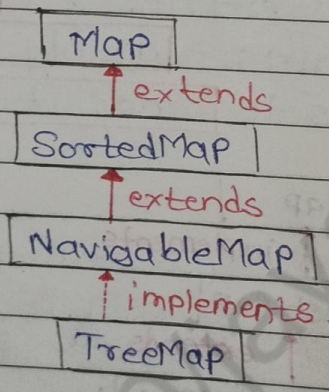
```
LinkedHashMap<Key,Value> num = new LinkedHashMap<>(capacity,
                                    LoadFactor, accessOrder);
```

- accessOrder ⇒ is a boolean value.
- by def. it is false
- In this case entries in the linkedHashMap are ordered on the basis of their insertion order.
- if true is passed as accessOrder, entries in linked hashmap will be ordered from least-recently accessed to most-recently accessed.

✱ Java TreeMap

> The TreeMap class of the Java collection framework provides the tree data structure implementation.

> It implements the NavigableMap interface.



① Creating a TreeMap.

> import java.util.TreeMap and java.util.*;

> TreeMap<key, value> numbers = new TreeMap<>();