# Collection

# Collection

- Collections are like containers that groups multiple items in a single unit.
- For example; a jar of chocolates, list of names etc. Collections are used almost in every programming language

# Hashtable

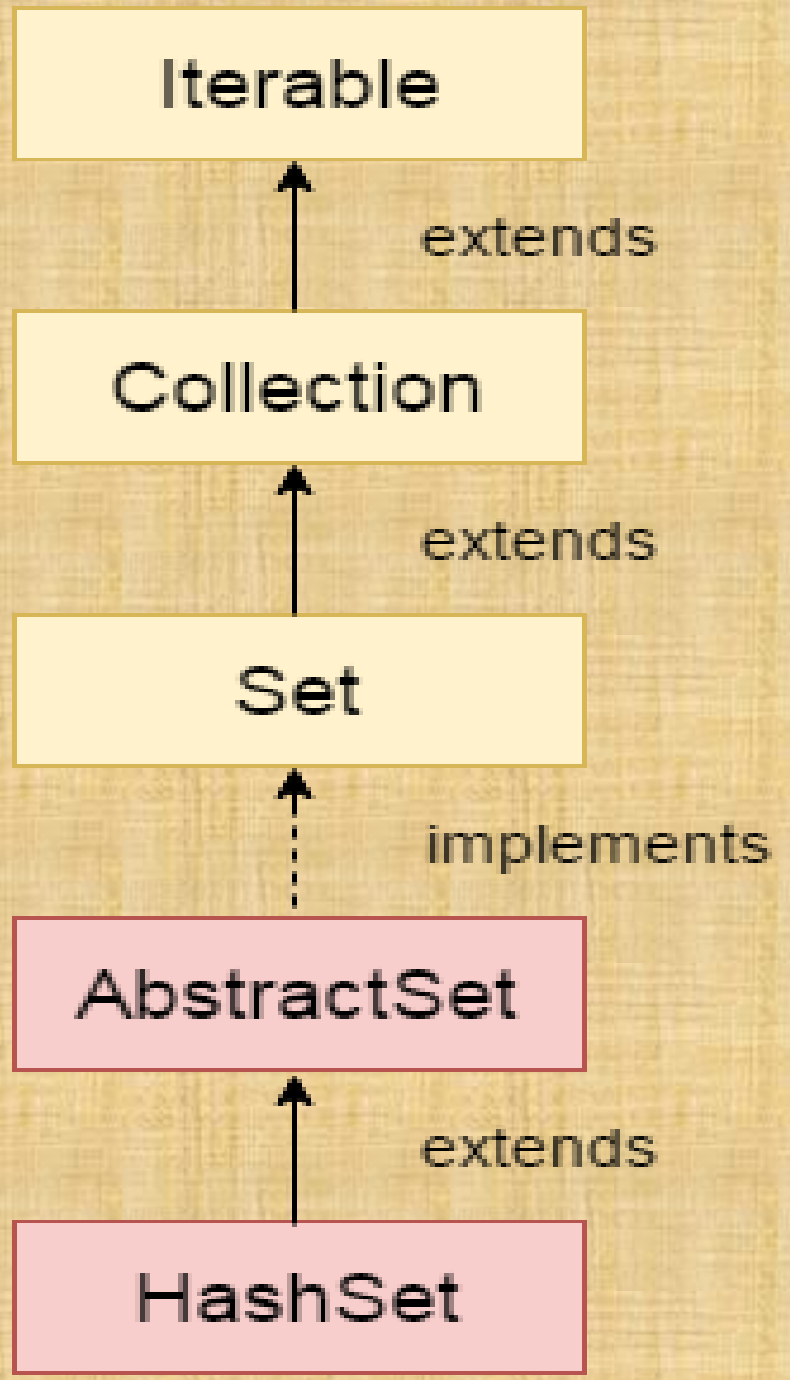
- Hashtable class implements a hashtable, which maps keys to values. Any non-null object can be used as a key or as a value.
- It is similar to HashMap, but is synchronised.
- Hashtable stores key/value pair in hash table.

```
import java.util.*;
class Aug
{
    public static void main(String[] arg)
    { //creating a hash table
        Hashtable h = new Hashtable();
         Hashtable h1 = new Hashtable();
          h.put(3, "Geeks");
        h.put(2, "forGeeks");
        h.put(1, "isBest");
        System.out.println("Value in table " + h);
        // create a clone or shallow copy of hash table h
        h1 = (Hashtable)h.clone();
         // checking clone h1
        System.out.println("values in clone: " + h1);
        // clear hash table h
        h.clear();
       // checking hash table h
        System.out.println("after clearing: " + h);
     }
```

- Output:-
- value in table: {3=Geeks, 2=forGeeks, 1=isBest}
- values in clone: {3=Geeks, 2=forGeeks, 1=isBest}
- after clearing: {}

# HashSet

- The HashSet class implements the Set interface, backed by a hash table which is actually a HashMap instance.
- HashSets are used to store a collection of *unique* elements.
- HashSet cannot contain duplicate values.
- HashSet is an unordered collection. It does not maintain the order in which the elements are inserted.
- HashSet is not thread-safe.



```
import java.util.HashSet;
import java.util.Set;

public class HashSt {
    public static void main(String[] args) {
        // Creating a HashSet
        Set<String> daysOfWeek = new HashSet<>();

        // Adding new elements to the HashSet
        daysOfWeek.add("Monday");
        daysOfWeek.add("Tuesday");
        daysOfWeek.add("Wednesday");
        daysOfWeek.add("Thursday");
        daysOfWeek.add("Friday");
        daysOfWeek.add("Saturday");
        daysOfWeek.add("Sunday");

        // Adding duplicate elements will be ignored
        daysOfWeek.add("Monday");

        System.out.println(daysOfWeek);
    }
}
```

```
# Output
[Monday, Thursday, Friday, Sunday, Wednesday, Tuesday, Saturday]
```

```
import java.util.HashSet;
import java.util.Set;

public class HashSt {
    public static void main(String[] args) {
        Set<Integer> numbers = new HashSet<>();
        numbers.add(2);
        numbers.add(3);
        numbers.add(4);
        numbers.add(5);
        numbers.add(6);
        numbers.add(7);
        numbers.add(8);
        numbers.add(9);
        numbers.add(10);
        System.out.println("numbers : " + numbers);

        // Remove an element from a HashSet (The remove() method returns false if the element does not exist in the HashSet)
        boolean isRemoved = numbers.remove(10);
        System.out.println(isRemoved);
        System.out.println("After remove(10) => " + numbers);
        }
```

```
# Output
numbers : [2, 3, 4, 5, 6, 7, 8, 9, 10]
After remove(10) => [2, 3, 4, 5, 6, 7, 8, 9]
```

# LinkedHashSet

- A LinkedHashSet is an ordered version of HashSet that maintains a doubly-linked List across all elements.
- **Syntax:**
- LinkedHashSet<String> hs = new LinkedHashSet<String>();
- Maintains insertion order.

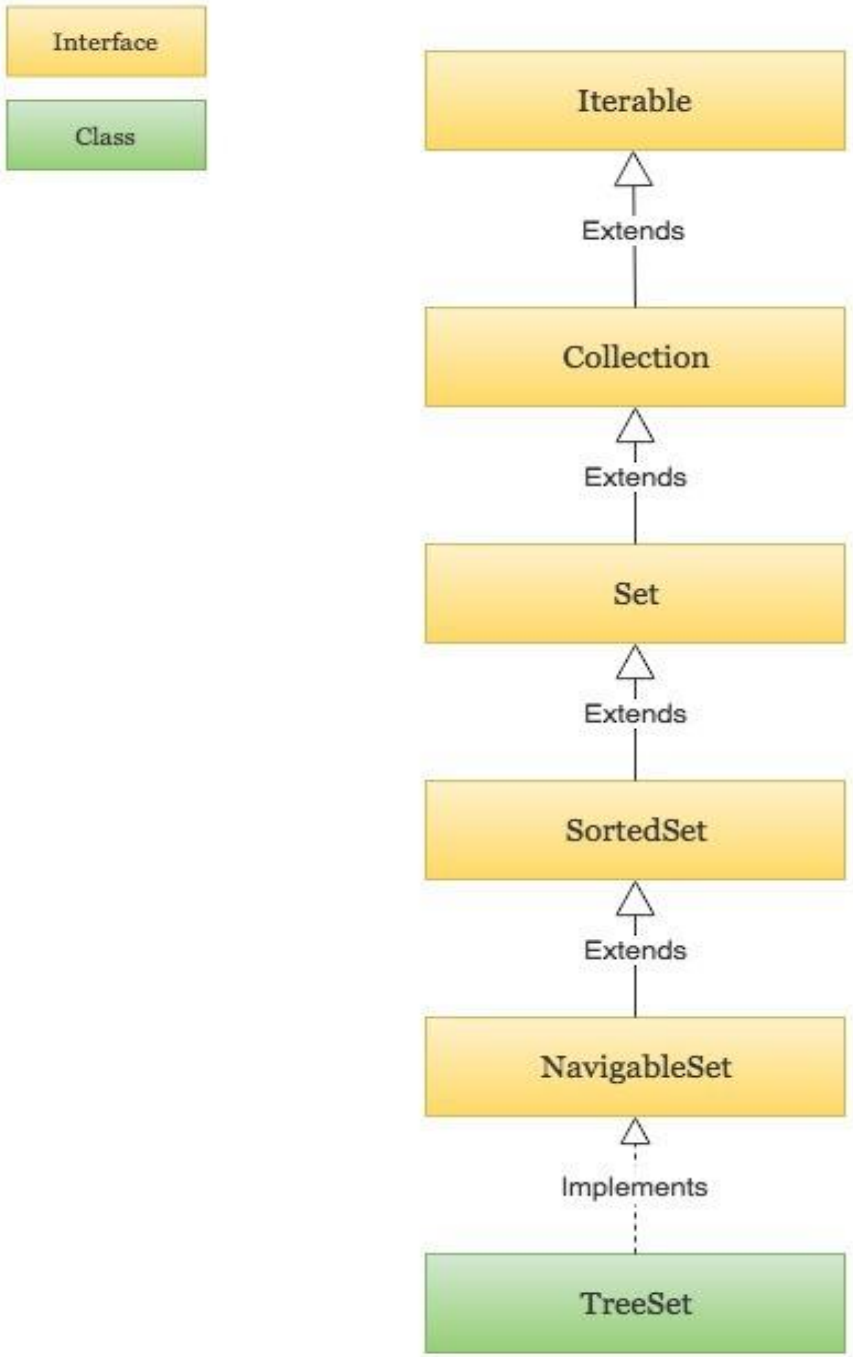
```
{   public static void main(String[] args)
    {
        LinkedHashSet<String> linkedset = new LinkedHashSet<String>();

        // Adding element to LinkedHashSet
        linkedset.add("A");
        linkedset.add("B");
        linkedset.add("C");
        linkedset.add("D");
        // This will not add new element as A already exists
        linkedset.add("A");
        linkedset.add("E");
        System.out.println("Size of LinkedHashSet = " +
                           linkedset.size());
        System.out.println("Original LinkedHashSet:" + linkedset);
        System.out.println("Removing D from LinkedHashSet: " +
                   linkedset.remove("D"));
        System.out.println("Trying to Remove Z which is not "+
                   "present: " + linkedset.remove("Z"));
        System.out.println("Checking if A is present=" +
```

- OutPut:-
- Size of LinkedHashSet=5
- Original LinkedHashSet:[A, B, C, D, E]
- Removing D from LinkedHashSet: true
- Trying to Remove Z which is not present: false
- Checking if A is present=true
- Updated LinkedHashSet: [A, B, C, E]

# TreeSet

- TreeSet is one of the most important implementations of the SortedSet interface in Java that uses a Tree for storage.
- It stores unique elements.
- It sorts the elements in ascending order.
- It’s not thread-safe.

Java TreeSet Class Hierarchy



```
import java.util.SortedSet;
import java.util.TreeSet;

public class CreateTreeSetExample {
    public static void main(String[] args) {
        // Creating a TreeSet
        SortedSet<String> fruits = new TreeSet<>();

        // Adding new elements to a TreeSet
        fruits.add("Banana");
        fruits.add("Apple");
        fruits.add("Pineapple");
        fruits.add("Orange");
        System.out.println("Fruits Set : " + fruits);
        fruits.add("Apple"); // Duplicate elements are ignored
        System.out.println("After adding duplicate element \"Apple\" : " + fruits);
        // This will be allowed because it's in lowercase.
        fruits.add("banana");
        System.out.println("After adding \"banana\" : " + fruits);
    }
}
```

- # Output
- Fruits Set : [Apple, Banana, Orange, Pineapple]
- After adding duplicate element "Apple" : [Apple, Banana, Orange, Pineapple]
- After adding "banana" : [Apple, Banana, Orange, Pineapple, banana]