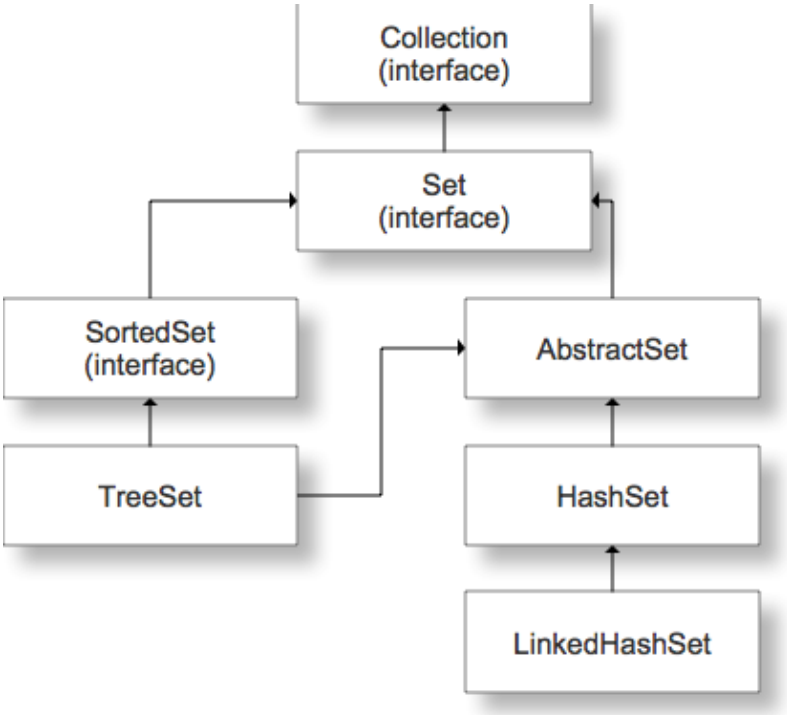
# Set and Map

Set Interface is used to implement classes TreeSet, HashSet, and LinkedHashSet.




Set Objects do not allow duplicates and are not required to maintain their objects in the order they are added.

| Set | Maintains Items |
|---|---|
| HashSet | in an order designed for fast look up. |
| LinkedHashSet | in the order they are added while also maintain hashing for fast look up. |
| TreeSet | in a sorted order |

HashSet

- No duplicate entries
- Objects in the HashSet set
  - must implement equals method
- can contain no more than one null entry

key methods of Set Interface

- boolean add(Object o) (false if object was not added to the set)
- void clear() (removes all items from the Set)

- bolean contains(Object o) (false if object was not in the set)
- boolean isEmpty() (check to se if set is empty)
- boolean remove(Object o) (false if object was not removed)
- int size() (returns the size of the set)
- Object [] toArray()
- Iterator iterator()

```
Set<String> names = new HashSet<String>();
names.add("John");
...
names.remove("Tom");
...
if(names.contains("John"))
...
```

Using iterators

```
Iterator<String> iter = names.iterator();
while(iter.hasNext())
{
        String s = iter.next();
        ...
        (do something)
}
or using for each loop
for ( String name : names)
        (do something with name)
```

TreeSet

Sorting items in the set

```
TreeSet tStu = new TreeSet(how to sort the items);
Implement the method CompareTo
public class Student implements Comparable
{   public int compareTo(Object o)
        {
            ...
```

```
            return ___;
        }
}
```

The technique of implementing the method compareTo above is all need to sort only on one field.

Sorting m different fields

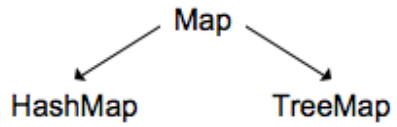
```
public class Student implements Compareable
{
        public static final Comparator SOR_ID_ASC = new Comparator()
        {
            public int comparteTo(Object o)
            {
                ...
            }
        };
        public static final Comparator SOR_LAST_ASC = new Comparator()
        {
            public int compareTo(Object o)
            {
                ...
            }
        };
        ....
}
```

Create a TreeSet object

```
TreeSet tStru = new TreeSet(Student.SOR_ID_ASC);
```

Map

- An object tha maps keys to values
- A Map cannot contain duplicate keys
- A map might contain duplicate values
- Each key can map to a t most one value




HashMap

HashMap(K,E)

ex. HashMap<String, String> hmap = new HashMap<String, String>();

get(akey)

get("123") → "John"

put(aKey, aValue)

put("999", "Mike")

containsKey(aKey)

containsValue(aValue)

remove(aKey)

values() – returns a set of all values

keyset() – returns a set of all keys

TreeMap

- Sorts the table with respect to the keys
- Otherwise, same as HashMap

HashTable

- Uses a hashing function that computes an integer value ( hash code) from an object.
- The hash code must have the property that different objects are likely to yield different hash codes.

Initialize a HashSet capacity

HashSet( *initial capacity*, *loading factor 0→1*)

initial capacity: 100

loading factor: .75
Reach 75 entries → double capacity 200

Overriding methods equals and hashCode

```
public class Student
{
        private int id;
        private String last
        private String first;
        ….
        public boolean equals(Object o)
        {
                return this.id == ((Student) o).id;
        }
        public int hashCode()
        {
                return id.hashCode();
        }
}
```