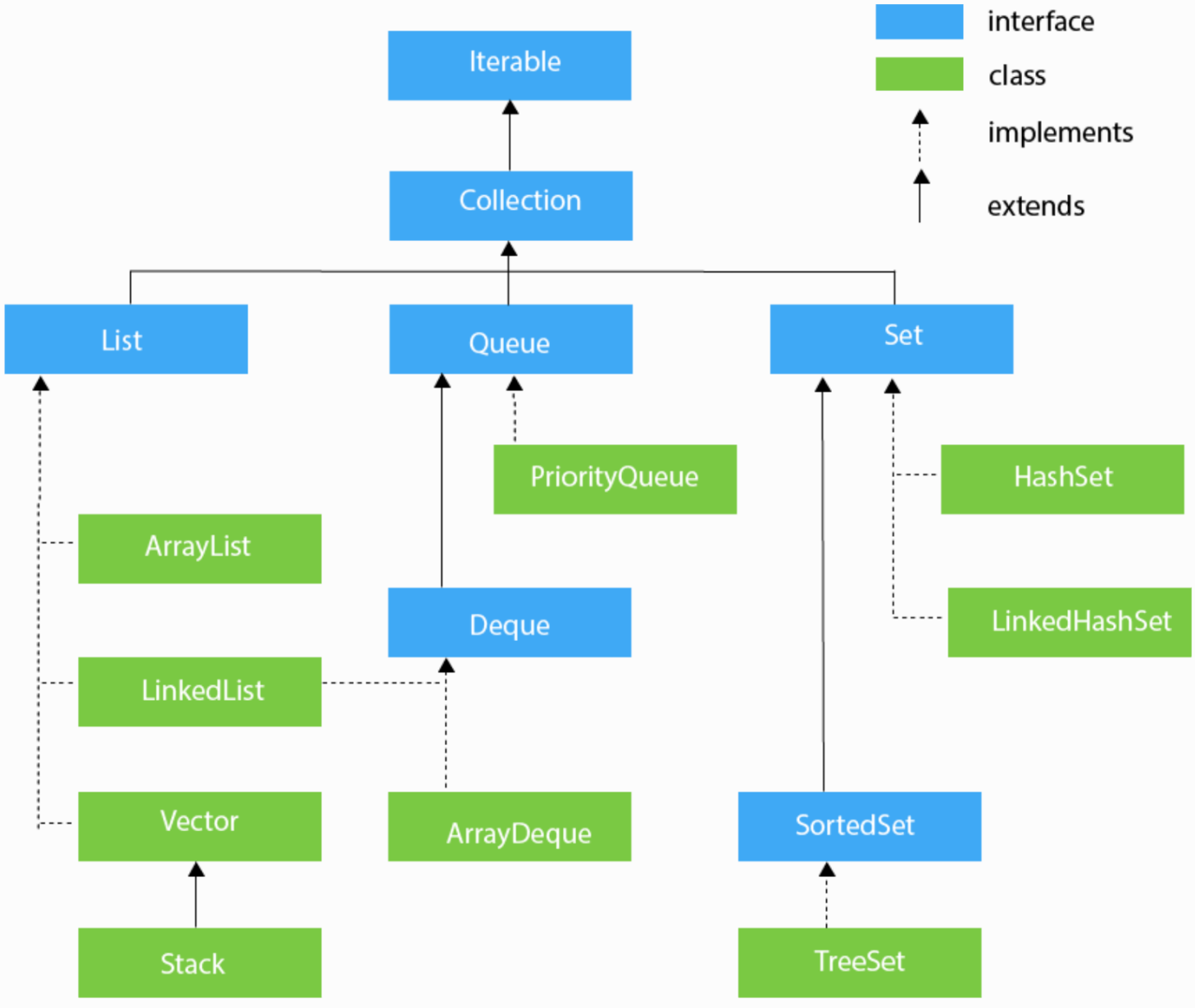

Collection

- framework.

Collections

- class inside Collection framework.
- Collections.sort()
  .reverse().

Collection / framework / API

Sun micro system: "joshua" made Collection framework

why?

1. Reduces programming effort
2. provides - in built methods & classes.
3. Reduce operational time
4. Interoperability: - two system using same thing by API.

7 classes:

util

List (I)
1. Array List (Dynamic Array) - O(n)
2. Linked List (Doubly LL) - O(n)

Queue (I)
3. Array Deque (Double ended Queue) - O(n)
4. Priority Queue (min Heap) - O(n)

Set (I)
5. Tree Set (Balanced BST) - Searching O(log n)
6. Hash Set (Hash table — Hash fn, load factor) - O(1)
7. Linked Hash Set — O(1) — maintain order of Insertion also.

Iterator:

1. Iterator (All collection classes)
2. List Iterator (AL, LL)
3. Descending Iterator (LL, ArrayDeque, TreeSet)

```
while (ditr.hasNext())
{ sys (ditr.next()); // 300, 200, 100
}
```

9th dec

Different ways of Accessing DS:

// normal loop

```
for (int i=0 ; i<=al.size() ; i++)
{ sop (al.get(i)); }
```

// for each

```
for (Object obj : al)
    sop (obj);
```

only list based classes

// Iterator


```
Iterator itr = al.iterator();   // method
if (itr.hasNext () == true)
{ itr.next(); }
while (itr.hasNext())
{ Integer i = (Integer) itr.next();   // return Object
  Object o = itr.next();
  sys (itr.next());
}
```

// reverse only for Arraylist & LL

```
ListIterator litr = al.listIterator(al.size());
while (litr.hasPrevious())
{ sys (litr.previous()); }
```

// Descending Iterator (LL, ArrayDeque, TreeSet)

```
LinkedList lt = new LinkedList();
lt.add(100); lt.add(200); lt.add(300);

Iterator ditr = lt.descendingIterator();
```

Enumeration:

└ before collection

we have legacy classes (like vector).

enumeration used before collection.

```
Vector v = new Vector();
v.add(10); v.add(20); v.add(30);

Enumeration em = v.elements();
while(em.hasMoreElements())
{ sys(em.nextElement()); }
```

	for(i)	for-each	iterator	listItr	DesItr	Enum
AL	✓	✓	✓	✓	x	x
LL	✓	✓	✓	✓	✓	x
AD	x	✓	✓	x	✓	x
PQ	x	✓	✓	x	x	x
TS	x	✓	✓	x	✓	x
HS	x	✓	✓	x	x	x
LHS	x	✓	✓	x	x	x

	Internal DS	order of Insertion	permits null value	Duplicate	searching
AL	Dynamic Array	✓	✓	✓	O(n)
LL	Doubly LL	✓	✓	✓	O(n)
AD	Double ended queue	✓	x	✓	O(n)
PQ	min-heap	x	x	✓	O(n)
TS	Balanced BST	x	x	x	O(logn)
HS	HashTable	x	✓	x	O(1)
LHS	HashTable	✓	✓	x	O(1)

Fail Fast Fail safe :

(why iterator not for() & for-each() ?)

Structural modifn : while accessing the data if you are modifying the DS, then technically it is called as structural modifn. or Concurrent modifn.

for() & for-each() can't handle structural modfn, it will run ∞ times.

to remove ∞ time running (illegal logic) we use Iterator. (it'll give, ConcurrentModificationException) & terminate it from ∞ running.

E.g.

```
AL al = new AL();
al.add(10, 20, 30);
for (Object o : al)
{   sys(o);
    al.add(100); // 10 20 30 100 100 ... ∞
}
```

// fail fast

```
Iterator it = al.iterator();
for (it.hasNext())
{   sys(it.next());
    al.add(100); // ConcurModn Exception
}
```

// fail safe

```
CopyOnWriteArrayList cal = new CopyOn...();
cal.add(100, 200, 300, 400);
Iterator itr = cal.iterator();
while (itr.hasNext())
{   sys(itr.next()); // 100, 200, 300, 400
    cal.add(1234); // no exception
}
```

Map Interface

Interface inside Interface? ✓

Eg: Student has A bike (loosly coupled) → (aggregation), can also be access after the student passed.
has A heart (tightly coupled) → (composition) student gone heart gone

```
interface Map
{   interface Entry
    {
    }
}
```

<10, "Hi"> - Entry

Map<key, value> <20, "H">
<30, "Y">

- HashMap<K,V>
- key must be unique
- no iteration()
- LinkedHashMap<K,V>
- maintain order of Insertion

.tostring()

```
sop(obj ref) // HashCode / address of obj
sop(obj ref) // value
```

```
// View                    return
Hm.entrySet();             // key, value
Hm.keySet();               // key
Hm.values();               // value
```

```
Collection c = Hm.values();
Iterator itr = c.iterator();
for(itr.hasNext())
  { sys(itr.next());
  }
```

HashMap

31 December 2022 20:48

```
public class HashMap<K,V> extends AbstractMap<K,V> implements Map<K,V>, Cloneable, Serializable
```

-

Constructor and Description
HashMap() Constructs an empty HashMap with the default initial capacity (16) and the default load factor (0.75).
HashMap(int initialCapacity) Constructs an empty HashMap with the specified initial capacity and the default load factor (0.75).
HashMap(int initialCapacity, float loadFactor) Constructs an empty HashMap with the specified initial capacity and load factor.
HashMap(Map<? extends K,? extends V> m) Constructs a new HashMap with the same mappings as the specified Map.

Constructors

- ***Method Summary***

Modifier and Type	Method and Description
void	clear() Removes all of the mappings from this map.
Object	clone() Returns a shallow copy of this HashMap instance: the keys and values themselves are not cloned.
V	compute(K key, BiFunction<? super K,? super V,? extends V> remappingFunction) Attempts to compute a mapping for the specified key and its current mapped value (or null if there is no current mapping).
V	computeIfAbsent(K key, Function<? super K,? extends V> mappingFunction) If the specified key is not already associated with a value (or is mapped to null), attempts to compute its value using the given mapping function and enters it into this map unless null.
V	computeIfPresent(K key, BiFunction<? super K,? super V,? extends V> remappingFunction) If the value for the specified key is present and non-null, attempts to compute a new mapping given the key and its current mapped value.
boolean	containsKey(Object key) Returns true if this map contains a mapping for the specified key.
boolean	containsValue(Object value) Returns true if this map maps one or more keys to the specified value.
Set<Map.Entry<K,V>>	entrySet() Returns a Set view of the mappings contained in this map.
void	forEach(BiConsumer<? super K,? super V> action) Performs the given action for each entry in this map until all entries have been processed or the action throws an exception.
V	get(Object key) Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key.

V	getOrDefault(Object key, V defaultValue) Returns the value to which the specified key is mapped, or defaultValue if this map contains no mapping for the key.
boolean	isEmpty() Returns true if this map contains no key-value mappings.
Set<K>	keySet() Returns a Set view of the keys contained in this map.
V	merge(K key, V value, BiFunction<? super V,? super V,? extends V> remappingFunction) If the specified key is not already associated with a value or is associated with null, associates it with the given non-null value.
V	put(K key, V value) Associates the specified value with the specified key in this map.
void	putAll(Map<? extends K,? extends V> m) Copies all of the mappings from the specified map to this map.
V	putIfAbsent(K key, V value) If the specified key is not already associated with a value (or is mapped to null) associates it with the given value and returns null, else returns the current value.
V	remove(Object key) Removes the mapping for the specified key from this map if present.
boolean	remove(Object key, Object value) Removes the entry for the specified key only if it is currently mapped to the specified value.
V	replace(K key, V value) Replaces the entry for the specified key only if it is currently mapped to some value.
boolean	replace(K key, V oldValue, V newValue) Replaces the entry for the specified key only if currently mapped to the specified value.
void	replaceAll(BiFunction<? super K,? super V,? extends V> function) Replaces each entry's value with the result of invoking the given function on that entry until all entries have been processed or the function throws an exception.
int	size() Returns the number of key-value mappings in this map.
Collection<V>	values() Returns a Collection view of the values contained in this map.

All Methods Instance Methods Concrete Methods

- **Methods inherited from class java.util.AbstractMap**
  equals, hashCode, toString
- **Methods inherited from class java.lang.Object**
  finalize, getClass, notify, notifyAll, wait, wait, wait
- **Methods inherited from interface java.util.Map**
  equals, hashCode
- ***Nested Class Summary***
- **Nested classes/interfaces inherited from class java.util.AbstractMap**
  AbstractMap.SimpleEntry<K,V>, AbstractMap.SimpleImmutableEntry<K,V>
- **Nested classes/interfaces inherited from interface java.util.Map**
  Map.Entry<K,V>

ArrayList: Indexed based Ds. (Dynamic Array)

(Heterogenous data)

```
ArrayList<class> l = new AL<>();
    l.add(1)
    l.add("Hi"); // autoboxing "all elements stored as obj only"
    l.add('t');
  so(l); // [1, Hi, t]
  l.get(0); // 1
      L
    index
  l.size(); // 3
```

Virtual Capacity: (debug you can see it. PC=D)

VC

PC = Physical capacity

VC = 10

Default initial capacity is 10.

```
// typical for loop:
for (i=0 ; i<student.size() : i++)
        so(l.get(i));

// for-each
for (String s: l)
    so(s);

// Java-8 - lamda fn with streams
l.stream().forEach(ele -> sysout(ele));

// iterator:
Iterator<String> it = l.iterator();
while (it.hasNext())
{   Sysout(it.next());
}
```

// Initialization

```
AL<Integer> l = new AL<>(Arrays.asList
                (1, 2, 3));
```

Arraylist

11 December 2022 13:02

Constructor and Description
ArrayList() Constructs an empty list with an initial capacity of ten.
ArrayList(**Collection**<? extends **E**> c) Constructs a list containing the elements of the specified collection, in the order they are returned by the collection's iterator.
ArrayList(int initialCapacity) Constructs an empty list with the specified initial capacity.

Modifier and Type	Method and Description
boolean	**add**(**E** e) Appends the specified element to the end of this list.
void	**add**(int index, **E** element) Inserts the specified element at the specified position in this list.
boolean	**addAll**(**Collection**<? extends **E**> c) Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's Iterator.
boolean	**addAll**(int index, **Collection**<? extends **E**> c) Inserts all of the elements in the specified collection into this list, starting at the specified position.
void	**clear**() Removes all of the elements from this list.
Object	**clone**() Returns a shallow copy of this ArrayList instance.
boolean	**contains**(**Object** o) Returns true if this list contains the specified element.
void	**ensureCapacity**(int minCapacity) Increases the capacity of this ArrayList instance, if necessary, to ensure that it can hold at least the number of elements specified by the minimum capacity argument.
void	**forEach**(**Consumer**<? super **E**> action) Performs the given action for each element of the Iterable until all elements have been processed or the action throws an exception.
E	**get**(int index) Returns the element at the specified position in this list.

- **Methods inherited from class java.util.AbstractList**
  equals, hashCode
- **Methods inherited from class java.util.AbstractCollection**
  containsAll, toString
- **Methods inherited from class java.lang.Object**
- finalize, getClass, notify, notifyAll, wait, wait, wait
- **Methods inherited from interface java.util.List**
  containsAll, equals, hashCode
- **Methods inherited from interface java.util.Collection**
  parallelStream, stream

int	**indexOf**(**Object** o) Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element.
boolean	**isEmpty**() Returns true if this list contains no elements.
Iterator<**E**>	**iterator**() Returns an iterator over the elements in this list in proper sequence.
int	**lastIndexOf**(**Object** o) Returns the index of the last occurrence of the specified element in this list, or -1 if this list does not contain the element.
ListIterator<**E**>	**listIterator**() Returns a list iterator over the elements in this list (in proper sequence).
ListIterator<**E**>	**listIterator**(int index) Returns a list iterator over the elements in this list (in proper sequence), starting at the specified position in the list.
E	**remove**(int index) Removes the element at the specified position in this list.
boolean	**remove**(**Object** o) Removes the first occurrence of the specified element from this list, if it is present.
boolean	**removeAll**(**Collection**<?> c) Removes from this list all of its elements that are contained in the specified collection.
boolean	**removeIf**(**Predicate**<? super **E**> filter) Removes all of the elements of this collection that satisfy the given predicate.
protected void	**removeRange**(int fromIndex, int toIndex) Removes from this list all of the elements whose index is between fromIndex, inclusive, and toIndex, exclusive.
void	**replaceAll**(**UnaryOperator**<**E**> operator) Replaces each element of this list with the result of applying the operator to that element.
boolean	**retainAll**(**Collection**<?> c) Retains only the elements in this list that are contained in the specified collection.
E	**set**(int index, **E** element) Replaces the element at the specified position in this list with the specified element.
int	**size**() Returns the number of elements in this list.
void	**sort**(**Comparator**<? super **E**> c) Sorts this list according to the order induced by the specified **Comparator**.
Spliterator<**E**>	**spliterator**() Creates a ***late-binding*** and *fail-fast* **Spliterator** over the elements in this list.
List<**E**>	**subList**(int fromIndex, int toIndex) Returns a view of the portion of this list between the specified fromIndex, inclusive, and toIndex, exclusive.
Object[]	**toArray**() Returns an array containing all of the elements in this list in proper sequence (from first to last element).
<T> T[]	**toArray**(T[] a) Returns an array containing all of the elements in this list in proper sequence (from first to last element); the runtime type of the returned array is that of the specified array.
void	**trimToSize**() Trims the capacity of this ArrayList instance to be the list's current size.

Linked list: (doubly LL)

- Heterogeneous / homogeneous DS.
- non-contiguous m/o allocation
  dispersed
- Can perform iteration at any given position.
- Stores Data as object
- List (I) & Deque (I)
- list.offerFirst(4); // may or may not get added

  list.addFirst(4); // must be added even m/o is full.

- Index based adding & removal allowed.
- Duplicates allowed

Linked List

11 December 2022 13:48

Constructor and Description
LinkedList() Constructs an empty list.
LinkedList(**Collection**<? extends **E**> c) Constructs a list containing the elements of the specified collection, in the order they are returned by the collection's iterator.

Modifier and Type	Method and Description
boolean	**add**(**E** e) Appends the specified element to the end of this list.
void	**add**(int index, **E** element) Inserts the specified element at the specified position in this list.
boolean	**addAll**(**Collection**<? extends **E**> c) Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator.
boolean	**addAll**(int index, **Collection**<? extends **E**> c) Inserts all of the elements in the specified collection into this list, starting at the specified position.
void	**addFirst**(**E** e) Inserts the specified element at the beginning of this list.
void	**addLast**(**E** e) Appends the specified element to the end of this list.
void	**clear**() Removes all of the elements from this list.
Object	**clone**() Returns a shallow copy of this LinkedList.
boolean	**contains**(**Object** o) Returns true if this list contains the specified element.
Iterator<**E**>	**descendingIterator**() Returns an iterator over the elements in this deque in reverse sequential order.
E	**element**() Retrieves, but does not remove, the head (first element) of this list.
E	**get**(int index) Returns the element at the specified position in this list.
E	**getFirst**() Returns the first element in this list.
E	**getLast**() Returns the last element in this list.
int	**indexOf**(**Object** o) Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element.
int	**lastIndexOf**(**Object** o) Returns the index of the last occurrence of the specified element in this list, or -1 if this list does not contain the element.

Methods

- **Methods inherited from class java.util.AbstractSequentialList**
  iterator
- **Methods inherited from class java.util.AbstractList**
  equals, hashCode, listIterator, removeRange, subList
- **Methods inherited from class java.util.AbstractCollection**
  containsAll, isEmpty, removeAll, retainAll, toString
- **Methods inherited from class java.lang.Object**
  finalize, getClass, notify, notifyAll, wait, wait, wait
- **Methods inherited from interface java.util.List**
  containsAll, equals, hashCode, isEmpty, iterator, listIterator, removeAll, retainAll, subList
- **Methods inherited from interface java.util.Deque**
  iterator

ListIterator<**E**>	**listIterator**(int index) Returns a list-iterator of the elements in this list (in proper sequence), starting at the specified position in the list.
boolean	**offer**(**E** e) Adds the specified element as the tail (last element) of this list.
boolean	**offerFirst**(**E** e) Inserts the specified element at the front of this list.
boolean	**offerLast**(**E** e) Inserts the specified element at the end of this list.
E	**peek**() Retrieves, but does not remove, the head (first element) of this list.
E	**peekFirst**() Retrieves, but does not remove, the first element of this list, or returns null if this list is empty.
E	**peekLast**() Retrieves, but does not remove, the last element of this list, or returns null if this list is empty.
E	**poll**() Retrieves and removes the head (first element) of this list.
E	**pollFirst**() Retrieves and removes the first element of this list, or returns null if this list is empty.
E	**pollLast**() Retrieves and removes the last element of this list, or returns null if this list is empty.
E	**pop**() Pops an element from the stack represented by this list.
void	**push**(**E** e) Pushes an element onto the stack represented by this list.
E	**remove**() Retrieves and removes the head (first element) of this list.
E	**remove**(int index) Removes the element at the specified position in this list.
boolean	**remove**(**Object** o) Removes the first occurrence of the specified element from this list, if it is present.
E	**removeFirst**() Removes and returns the first element from this list.
boolean	**removeFirstOccurrence**(**Object** o) Removes the first occurrence of the specified element in this list (when traversing the list from head to tail).
E	**removeLast**() Removes and returns the last element from this list.
boolean	**removeLastOccurrence**(**Object** o) Removes the last occurrence of the specified element in this list (when traversing the list from head to tail).
E	**set**(int index, **E** element) Replaces the element at the specified position in this list with the specified element.
int	**size**() Returns the number of elements in this list.
Object[]	**toArray**() Returns an array containing all of the elements in this list in proper sequence (from first to last element).
<T> T[]	**toArray**(T[] a) Returns an array containing all of the elements in this list in proper sequence (from first to last element); the runtime type of the returned array is that of the specified array.

Array Deque :

- Double ended queue
- insertion front / deletion rear
- Duplicates allowed.
- index based access not allowed.
- Queue (I)

ArrayDeque

11 December 2022 13:58

Constructor and Description
ArrayDeque() Constructs an empty array deque with an initial capacity sufficient to hold 16 elements.
ArrayDeque(Collection<? extends E> c) Constructs a deque containing the elements of the specified collection, in the order they are returned by the collection's iterator.
ArrayDeque(int numElements) Constructs an empty array deque with an initial capacity sufficient to hold the specified number of elements.

Modifier and Type	Method and Description
boolean	**add**(E e) Inserts the specified element at the end of this deque.
void	**addFirst**(E e) Inserts the specified element at the front of this deque.
void	**addLast**(E e) Inserts the specified element at the end of this deque.
void	**clear**() Removes all of the elements from this deque.
ArrayDeque<E>	**clone**() Returns a copy of this deque.
boolean	**contains**(Object o) Returns true if this deque contains the specified element.
Iterator<E>	**descendingIterator**() Returns an iterator over the elements in this deque in reverse sequential order.
E	**element**() Retrieves, but does not remove, the head of the queue represented by this deque.
E	**getFirst**() Retrieves, but does not remove, the first element of this deque.
E	**getLast**() Retrieves, but does not remove, the last element of this deque.
boolean	**isEmpty**() Returns true if this deque contains no elements.
Iterator<E>	**iterator**() Returns an iterator over the elements in this deque.

- **Methods inherited from class java.util.AbstractCollection**
  addAll, containsAll, removeAll, retainAll, toString
- **Methods inherited from class java.lang.Object**
  equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait
- **Methods inherited from interface java.util.Collection**
  addAll, containsAll, equals, hashCode, removeAll, retainAll

boolean	**offer**(E e) Inserts the specified element at the end of this deque.
boolean	**offerFirst**(E e) Inserts the specified element at the front of this deque.
boolean	**offerLast**(E e) Inserts the specified element at the end of this deque.
E	**peek**() Retrieves, but does not remove, the head of the queue represented by this deque, or returns null if this deque is empty.
E	**peekFirst**() Retrieves, but does not remove, the first element of this deque, or returns null if this deque is empty.
E	**peekLast**() Retrieves, but does not remove, the last element of this deque, or returns null if this deque is empty.
E	**poll**() Retrieves and removes the head of the queue represented by this deque (in other words, the first element of this deque), or returns null if this deque is empty.
E	**pollFirst**() Retrieves and removes the first element of this deque, or returns null if this deque is empty.
E	**pollLast**() Retrieves and removes the last element of this deque, or returns null if this deque is empty.
E	**pop**() Pops an element from the stack represented by this deque.
void	**push**(E e) Pushes an element onto the stack represented by this deque.
E	**remove**() Retrieves and removes the head of the queue represented by this deque.
boolean	**remove**(Object o) Removes a single instance of the specified element from this deque.
E	**removeFirst**() Retrieves and removes the first element of this deque.
boolean	**removeFirstOccurrence**(Object o) Removes the first occurrence of the specified element in this deque (when traversing the deque from head to tail).
E	**removeLast**() Retrieves and removes the last element of this deque.
boolean	**removeLastOccurrence**(Object o) Removes the last occurrence of the specified element in this deque (when traversing the deque from head to tail).
int	**size**() Returns the number of elements in this deque.
Object[]	**toArray**() Returns an array containing all of the elements in this deque in proper sequence (from first to last element).
<T> T[]	**toArray**(T[] a) Returns an array containing all of the elements in this deque in proper sequence (from first to last element); the runtime type of the returned array is that of the specified array.

TreeSet

13 December 2022 15:22

Constructor and Description
TreeSet() Constructs a new, empty tree set, sorted according to the natural ordering of its elements.
TreeSet(Collection<? extends E> c) Constructs a new tree set containing the elements in the specified collection, sorted according to the *natural ordering* of its elements.
TreeSet(Comparator<? super E> comparator) Constructs a new, empty tree set, sorted according to the specified comparator.
TreeSet(SortedSet<E> s) Constructs a new tree set containing the same elements and using the same ordering as the specified sorted set.

- **Method Summary**

Modifier and Type	Method and Description
boolean	add(E e) Adds the specified element to this set if it is not already present.
boolean	addAll(Collection<? extends E> c) Adds all of the elements in the specified collection to this set.
E	ceiling(E e) Returns the least element in this set greater than or equal to the given element, or null if there is no such element.
void	clear() Removes all of the elements from this set.
Object	clone() Returns a shallow copy of this TreeSet instance.
Comparator<? super E>	comparator() Returns the comparator used to order the elements in this set, or null if this set uses the natural ordering of its elements.
boolean	contains(Object o) Returns true if this set contains the specified element.
Iterator<E>	descendingIterator() Returns an iterator over the elements in this set in descending order.
NavigableSet<E>	descendingSet() Returns a reverse order view of the elements contained in this set.
E	first() Returns the first (lowest) element currently in this set.
E	floor(E e) Returns the greatest element in this set less than or equal to the given element, or null if there is no such element.
SortedSet<E>	headSet(E toElement) Returns a view of the portion of this set whose elements are strictly less than toElement.
NavigableSet<E>	headSet(E toElement, boolean inclusive) Returns a view of the portion of this set whose elements are less than (or equal to, if inclusive is true) toElement.
E	higher(E e) Returns the least element in this set strictly greater than the given element, or null if there is no such element.
boolean	isEmpty() Returns true if this set contains no elements.
Iterator<E>	iterator() Returns an iterator over the elements in this set in ascending order.
E	last() Returns the last (highest) element currently in this set.
E	lower(E e) Returns the greatest element in this set strictly less than the given element, or null if there is no such element.
E	pollFirst() Retrieves and removes the first (lowest) element, or returns null if this set is empty.
E	pollLast() Retrieves and removes the last (highest) element, or returns null if this set is empty.
boolean	remove(Object o) Removes the specified element from this set if it is present.
int	size() Returns the number of elements in this set (its cardinality).
NavigableSet<E>	subSet(E fromElement, boolean fromInclusive, E toElement, boolean toInclusive) Returns a view of the portion of this set whose elements range from fromElement to toElement.
SortedSet<E>	subSet(E fromElement, E toElement) Returns a view of the portion of this set whose elements range from fromElement, inclusive, to toElement, exclusive.
SortedSet<E>	tailSet(E fromElement) Returns a view of the portion of this set whose elements are greater than or equal to fromElement.
NavigableSet<E>	tailSet(E fromElement, boolean inclusive) Returns a view of the portion of this set whose elements are greater than (or equal to, if inclusive is true) fromElement.

Methods

- **Methods inherited from class java.util.AbstractSet**
  equals, hashCode, removeAll
- **Methods inherited from class java.util.AbstractCollection**
  containsAll, retainAll, toArray, toArray, toString
- **Methods inherited from class java.lang.Object**
  finalize, getClass, notify, notifyAll, wait, wait, wait
- **Methods inherited from interface java.util.Set**
  containsAll, equals, hashCode, removeAll, retainAll, toArray, toArray

HashSet

13 December 2022 15:27

Constructor and Description
HashSet() Constructs a new, empty set; the backing HashMap instance has default initial capacity (16) and load factor (0.75).
HashSet(**Collection**<? extends **E**> c) Constructs a new set containing the elements in the specified collection.
HashSet(int initialCapacity) Constructs a new, empty set; the backing HashMap instance has the specified initial capacity and default load factor (0.75).
HashSet(int initialCapacity, float loadFactor) Constructs a new, empty set; the backing HashMap instance has the specified initial capacity and the specified load factor.

Modifier and Type	Method and Description
boolean	**add**(**E** e) Adds the specified element to this set if it is not already present.
void	**clear**() Removes all of the elements from this set.
Object	**clone**() Returns a shallow copy of this HashSet instance: the elements themselves are not cloned.
boolean	**contains**(**Object** o) Returns true if this set contains the specified element.
boolean	**isEmpty**() Returns true if this set contains no elements.
Iterator<**E**>	**iterator**() Returns an iterator over the elements in this set.
boolean	**remove**(**Object** o) Removes the specified element from this set if it is present.
int	**size**() Returns the number of elements in this set (its cardinality).

Methods

- **Methods inherited from class java.util.AbstractSet**
  equals, hashCode, removeAll
- **Methods inherited from class java.util.AbstractCollection**
  addAll, containsAll, retainAll, toArray, toArray, toString
- **Methods inherited from class java.lang.Object**
  finalize, getClass, notify, notifyAll, wait, wait, wait
- **Methods inherited from interface java.util.Set**
  addAll, containsAll, equals, hashCode, removeAll, retainAll, toArray, toArray

LinkedHashSet

13 December 2022 15:29

Constructor and Description
LinkedHashSet() Constructs a new, empty linked hash set with the default initial capacity (16) and load factor (0.75).
LinkedHashSet(**Collection**<? extends **E**> c) Constructs a new linked hash set with the same elements as the specified collection.
LinkedHashSet(int initialCapacity) Constructs a new, empty linked hash set with the specified initial capacity and the default load factor (0.75).
LinkedHashSet(int initialCapacity, float loadFactor) Constructs a new, empty linked hash set with the specified initial capacity and load factor.

- **Methods inherited from class java.util.HashSet**
  add, clear, clone, contains, isEmpty, iterator, remove, size
- **Methods inherited from class java.util.AbstractSet**
  equals, hashCode, removeAll
- **Methods inherited from class java.util.AbstractCollection**
  addAll, containsAll, retainAll, toArray, toArray, toString
- **Methods inherited from class java.lang.Object**
  finalize, getClass, notify, notifyAll, wait, wait, wait
- **Methods inherited from interface java.util.Set**
  add, addAll, clear, contains, containsAll, equals, hashCode, isEmpty, iterator, remove, removeAll, retainAll, size, toArray, toArray