CSE 215: Programming Language II Lab

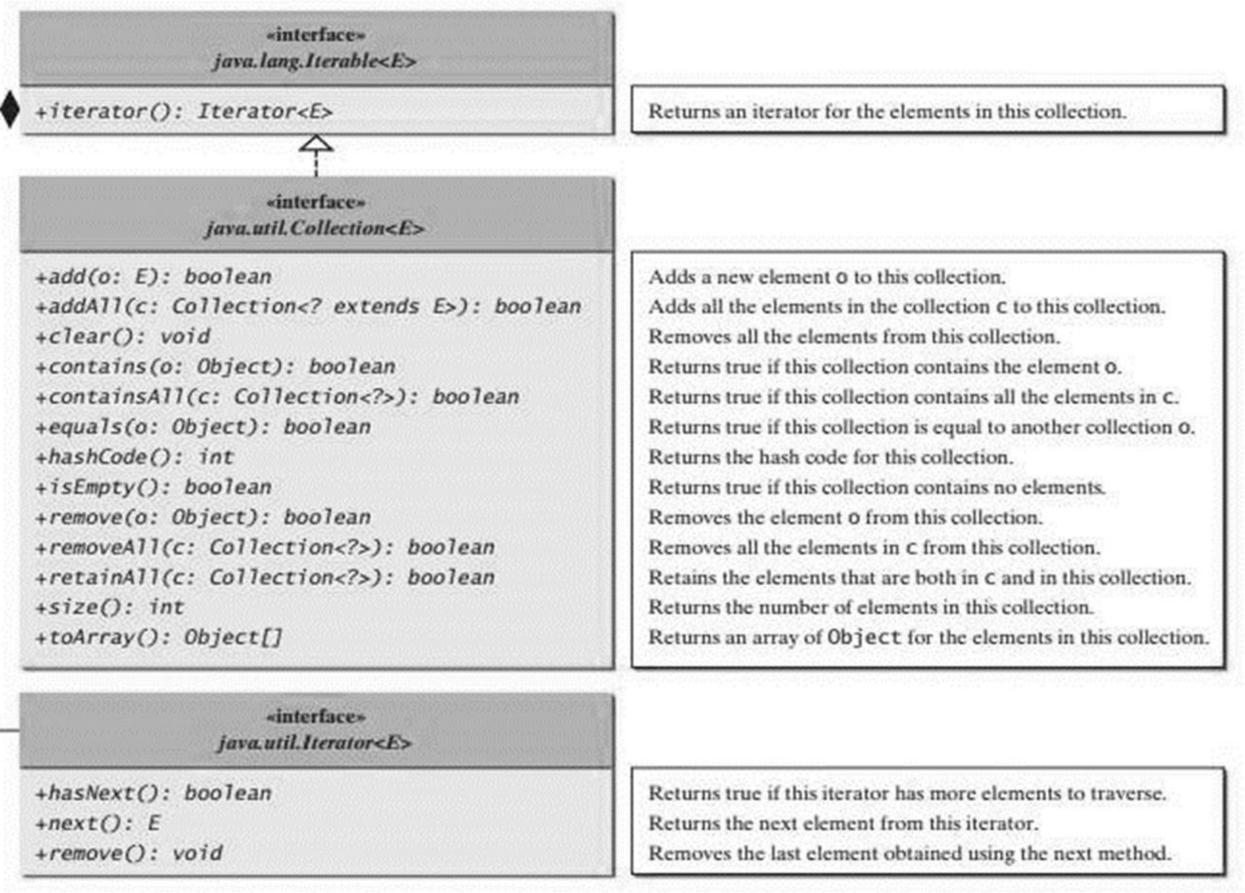
Lab – 15

List, Stack, Queue, Sets and Maps

Objective:
- To learn about stack and queue in java
- To learn to implement those in solving problems

Collection

The Collection interface defines the common operations for lists, vectors, stacks, queues, priority queues, and sets. Lists store an ordered collection of elements.
■ Stacks store objects that are processed in a last-in, first-out fashion.
■ Queues store objects that are processed in a first-in, first-out fashion.
■ PriorityQueues store objects that are processed in the order of their priorities.



«interface»
java.lang.Iterable<E>

`+iterator(): Iterator<E>`	Returns an iterator for the elements in this collection.

«interface»
java.util.Collection<E>

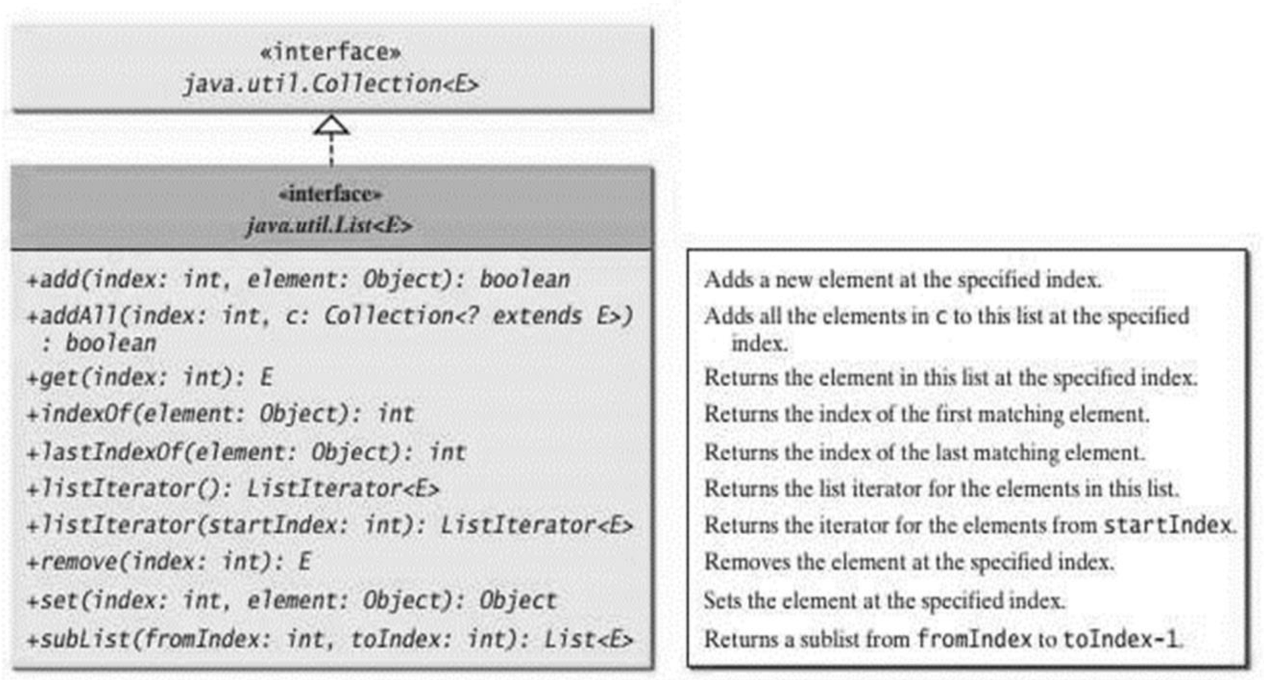
`+add(o: E): boolean`	Adds a new element o to this collection.
`+addAll(c: Collection<? extends E>): boolean`	Adds all the elements in the collection c to this collection.
`+clear(): void`	Removes all the elements from this collection.
`+contains(o: Object): boolean`	Returns true if this collection contains the element o.
`+containsAll(c: Collection<?>): boolean`	Returns true if this collection contains all the elements in c.
`+equals(o: Object): boolean`	Returns true if this collection is equal to another collection o.
`+hashCode(): int`	Returns the hash code for this collection.
`+isEmpty(): boolean`	Returns true if this collection contains no elements.
`+remove(o: Object): boolean`	Removes the element o from this collection.
`+removeAll(c: Collection<?>): boolean`	Removes all the elements in c from this collection.
`+retainAll(c: Collection<?>): boolean`	Retains the elements that are both in c and in this collection.
`+size(): int`	Returns the number of elements in this collection.
`+toArray(): Object[]`	Returns an array of Object for the elements in this collection.

«interface»
java.util.Iterator<E>

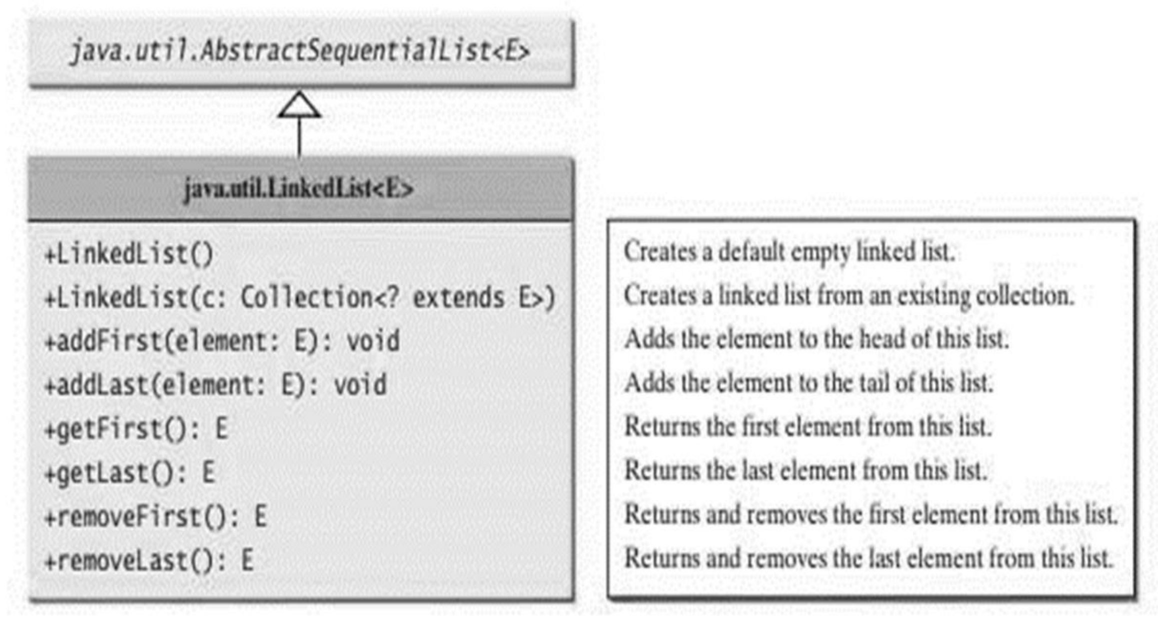
`+hasNext(): boolean`	Returns true if this iterator has more elements to traverse.
`+next(): E`	Returns the next element from this iterator.
`+remove(): void`	Removes the last element obtained using the next method.

List

The List interface extends the Collection interface and defines a collection for storing elements in a sequential order. To create a list, use one of its two concrete classes: ArrayList or LinkedList.

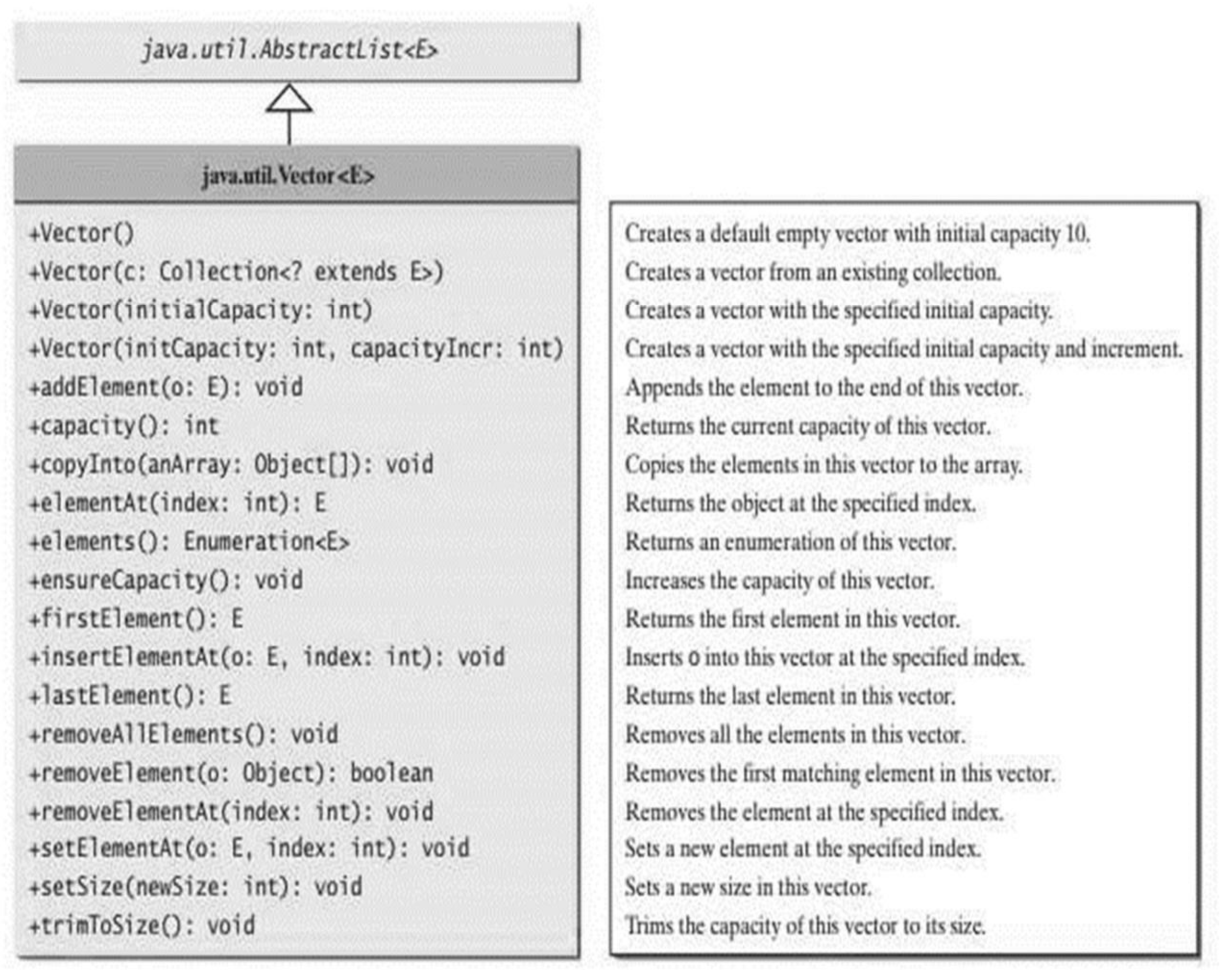

«interface»
java.util.Collection<E>
«interface»
java.util.List<E>
+add(index: int, element: Object): boolean
Adds a new element at the specified index.
+addAll(index: int, c: Collection<? extends E>) : boolean
Adds all the elements in c to this list at the specified index.
+get(index: int): E
Returns the element in this list at the specified index.
+indexOf(element: Object): int
Returns the index of the first matching element.
+lastIndexOf(element: Object): int
Returns the index of the last matching element.
+listIterator(): ListIterator<E>
Returns the list iterator for the elements in this list.
+listIterator(startIndex: int): ListIterator<E>
Returns the iterator for the elements from startIndex.
+remove(index: int): E
Removes the element at the specified index.
+set(index: int, element: Object): Object
Sets the element at the specified index.
+subList(fromIndex: int, toIndex: int): List<E>
Returns a sublist from fromIndex to toIndex-1.

The ArrayList class and the LinkedList class are two concrete implementations of the List interface. ArrayList stores elements in an array. LinkedList stores elements in a linked list.

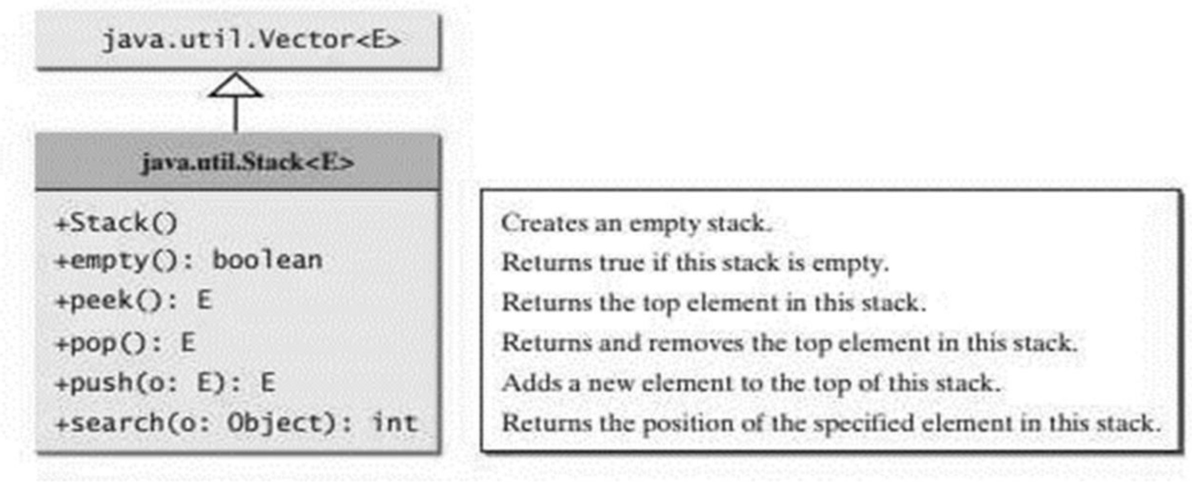

java.util.AbstractSequentialList<E>
java.util.LinkedList<E>
+LinkedList()
Creates a default empty linked list.
+LinkedList(c: Collection<? extends E>)
Creates a linked list from an existing collection.
+addFirst(element: E): void
Adds the element to the head of this list.
+addLast(element: E): void
Adds the element to the tail of this list.
+getFirst(): E
Returns the first element from this list.
+getLast(): E
Returns the last element from this list.
+removeFirst(): E
Returns and removes the first element from this list.
+removeLast(): E
Returns and removes the last element from this list.

Stack

Vector is a subclass of AbstractList, and Stack is a subclass of Vector in the.



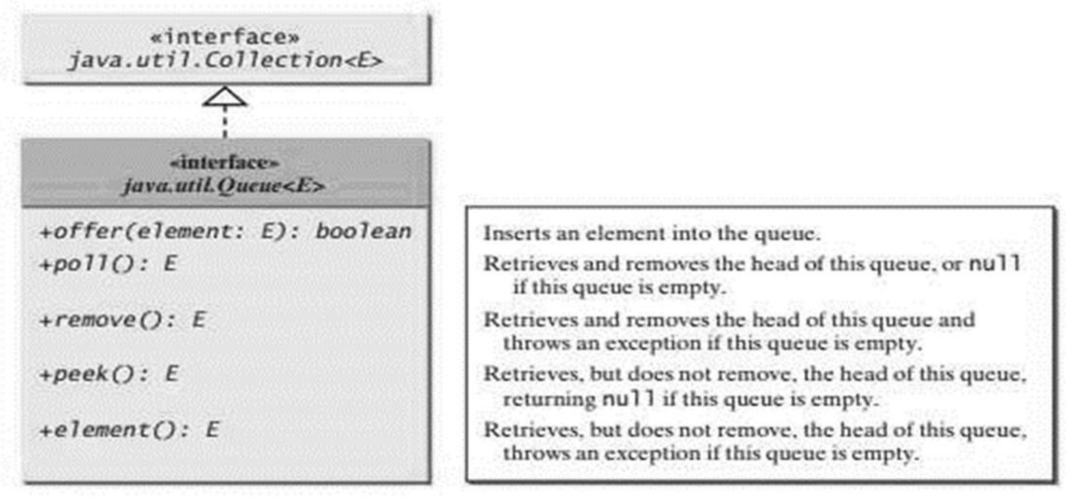
+Vector()	Creates a default empty vector with initial capacity 10.
+Vector(c: Collection<? extends E>)	Creates a vector from an existing collection.
+Vector(initialCapacity: int)	Creates a vector with the specified initial capacity.
+Vector(initCapacity: int, capacityIncr: int)	Creates a vector with the specified initial capacity and increment.
+addElement(o: E): void	Appends the element to the end of this vector.
+capacity(): int	Returns the current capacity of this vector.
+copyInto(anArray: Object[]): void	Copies the elements in this vector to the array.
+elementAt(index: int): E	Returns the object at the specified index.
+elements(): Enumeration<E>	Returns an enumeration of this vector.
+ensureCapacity(): void	Increases the capacity of this vector.
+firstElement(): E	Returns the first element in this vector.
+insertElementAt(o: E, index: int): void	Inserts o into this vector at the specified index.
+lastElement(): E	Returns the last element in this vector.
+removeAllElements(): void	Removes all the elements in this vector.
+removeElement(o: Object): boolean	Removes the first matching element in this vector.
+removeElementAt(index: int): void	Removes the element at the specified index.
+setElementAt(o: E, index: int): void	Sets a new element at the specified index.
+setSize(newSize: int): void	Sets a new size in this vector.
+trimToSize(): void	Trims the capacity of this vector to its size.



+Stack()	Creates an empty stack.
+empty(): boolean	Returns true if this stack is empty.
+peek(): E	Returns the top element in this stack.
+pop(): E	Returns and removes the top element in this stack.
+push(o: E): E	Adds a new element to the top of this stack.
+search(o: Object): int	Returns the position of the specified element in this stack.

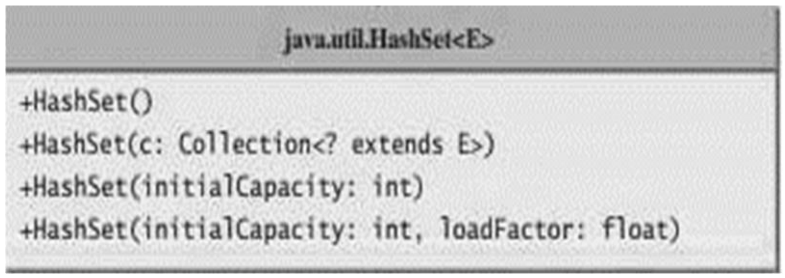
Queue

The Queue interface extends java.util.Collection with additional insertion, extraction, and inspection operations

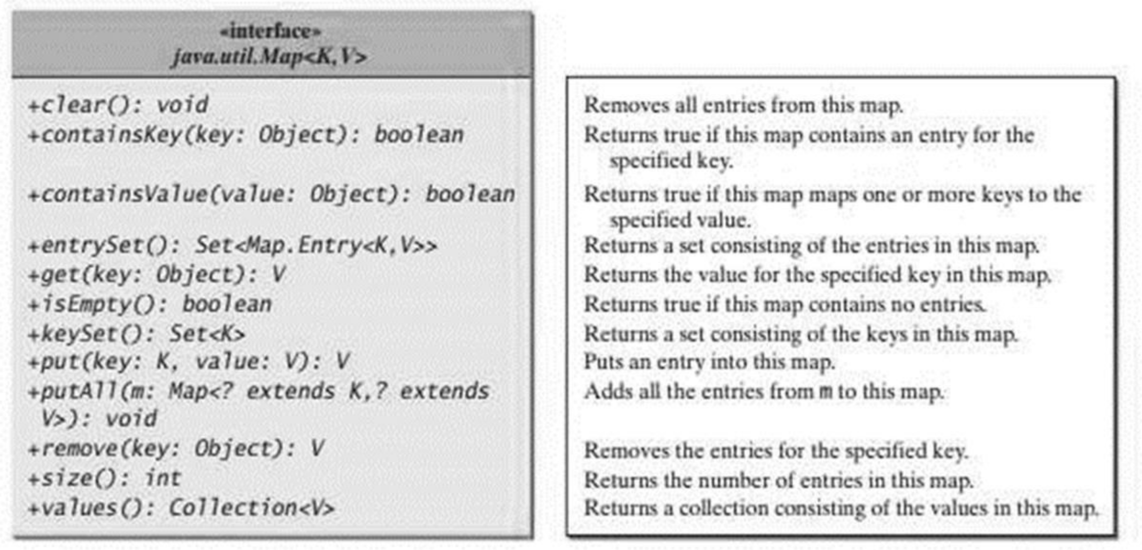

«interface»
java.util.Collection<E>
«interface»
java.util.Queue<E>
+offer(element: E): boolean
+poll(): E
+remove(): E
+peek(): E
+element(): E
Inserts an element into the queue.
Retrieves and removes the head of this queue, or null if this queue is empty.
Retrieves and removes the head of this queue and throws an exception if this queue is empty.
Retrieves, but does not remove, the head of this queue, returning null if this queue is empty.
Retrieves, but does not remove, the head of this queue, throws an exception if this queue is empty.

<u>Sets and Maps</u>

You can create a set using one of its three concrete classes: HashSet, LinkedHashSet, or TreeSet. The Set interface extends the Collection interface. It does not introduce new methods or constants, but it stipulates that an instance of Set contains no duplicate elements. The HashSet class is a concrete class that implements Set.


java.util.HashSet<E>
+HashSet()
+HashSet(c: Collection<? extends E>)
+HashSet(initialCapacity: int)
+HashSet(initialCapacity: int, loadFactor: float)

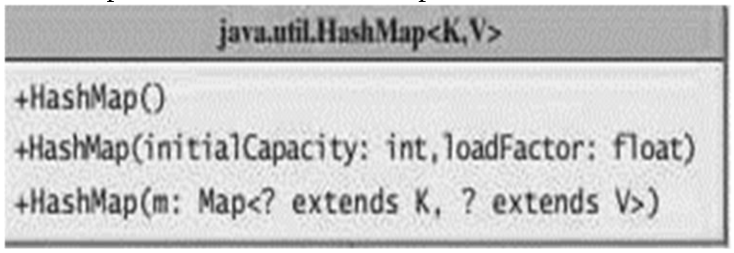
A map is a container object that stores a collection of key/value pairs. It enables fast retrieval, deletion, and updating of the pair through the key. A map stores the values along with the keys. The keys are like indexes.



«interface»
java.util.Map<K, V>

+clear(): void	Removes all entries from this map.
+containsKey(key: Object): boolean	Returns true if this map contains an entry for the specified key.
+containsValue(value: Object): boolean	Returns true if this map maps one or more keys to the specified value.
+entrySet(): Set<Map.Entry<K,V>>	Returns a set consisting of the entries in this map.
+get(key: Object): V	Returns the value for the specified key in this map.
+isEmpty(): boolean	Returns true if this map contains no entries.
+keySet(): Set<K>	Returns a set consisting of the keys in this map.
+put(key: K, value: V): V	Puts an entry into this map.
+putAll(m: Map<? extends K,? extends V>): void	Adds all the entries from m to this map.
+remove(key: Object): V	Removes the entries for the specified key.
+size(): int	Returns the number of entries in this map.
+values(): Collection<V>	Returns a collection consisting of the values in this map.

The Map interface except the entrySet() method. The HashMap, LinkedHashMap, and TreeMap classes are three concrete implementations of the Map interface.



java.util.HashMap<K,V>

```
+HashMap()
+HashMap(initialCapacity: int,loadFactor: float)
+HashMap(m: Map<? extends K, ? extends V>)
```

Lab Tasks

1. Write a program that evaluate compound mathematical expressions with multiple operator and parentheses. For simplicity, assume that the operands are integers and the operators are of four types: +, -, *, and /. (e.g., (15 + 2) * 34 – 2)
   Hint: The problem can be solved using two stacks, named operandStack and operatorStack, for storing operands and operators, respectively. Operands and operators are pushed into the stacks before they are processed. When an operator is processed, it is popped from operatorStack and applied to the first two operands from operandStack (the two operands are popped from operandStack). The resultant value is pushed back to operandStack.

2. Write a program that uses a HashMap to store an entry consisting of a word and its count. Hint: For each word, check whether it is already a key in the map. If not, add an entry to the map with the word as the key and value 1. Otherwise, increase the value for the word (key) by 1 in the map. Assume the words are case insensitive; e.g., Good is treated the same as good.