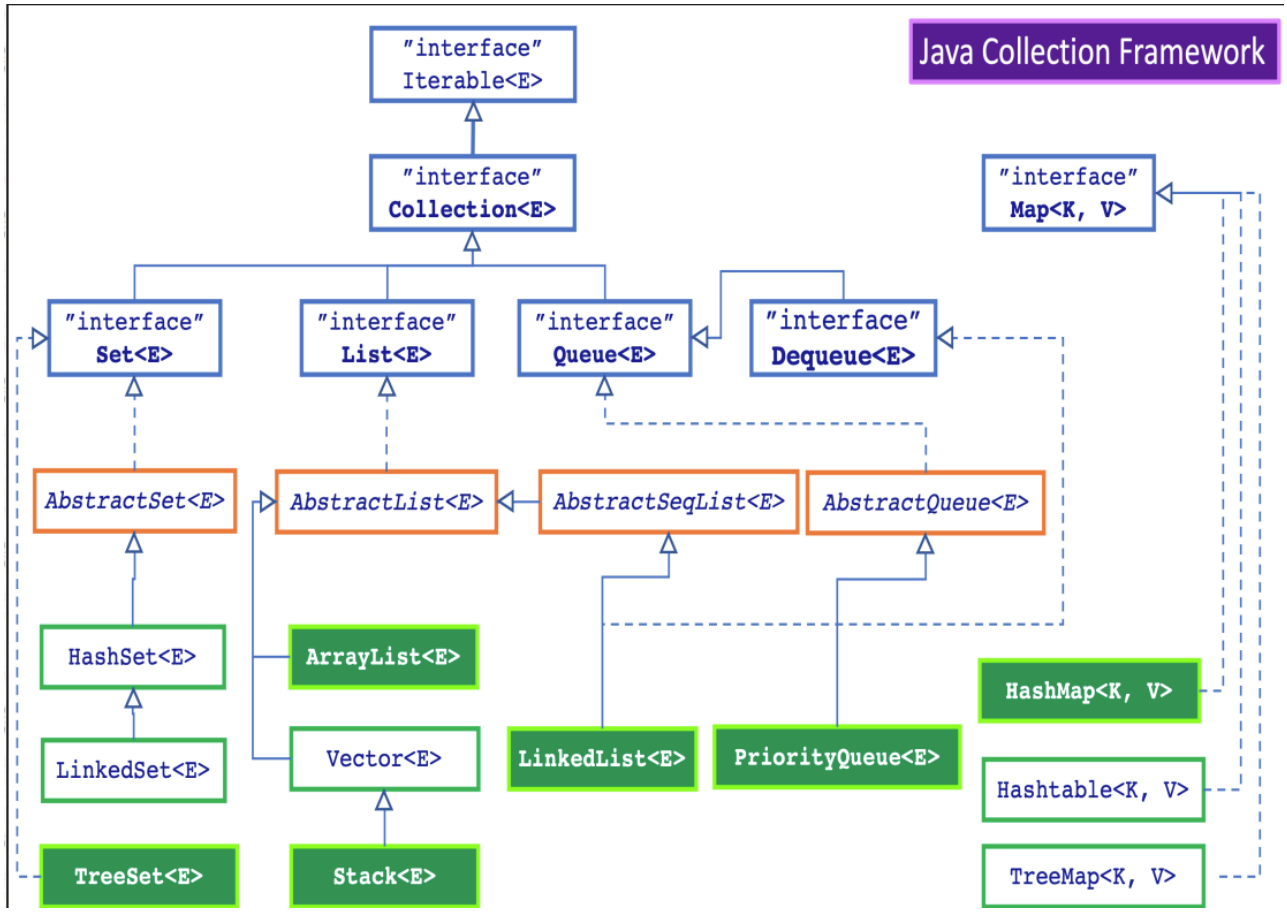
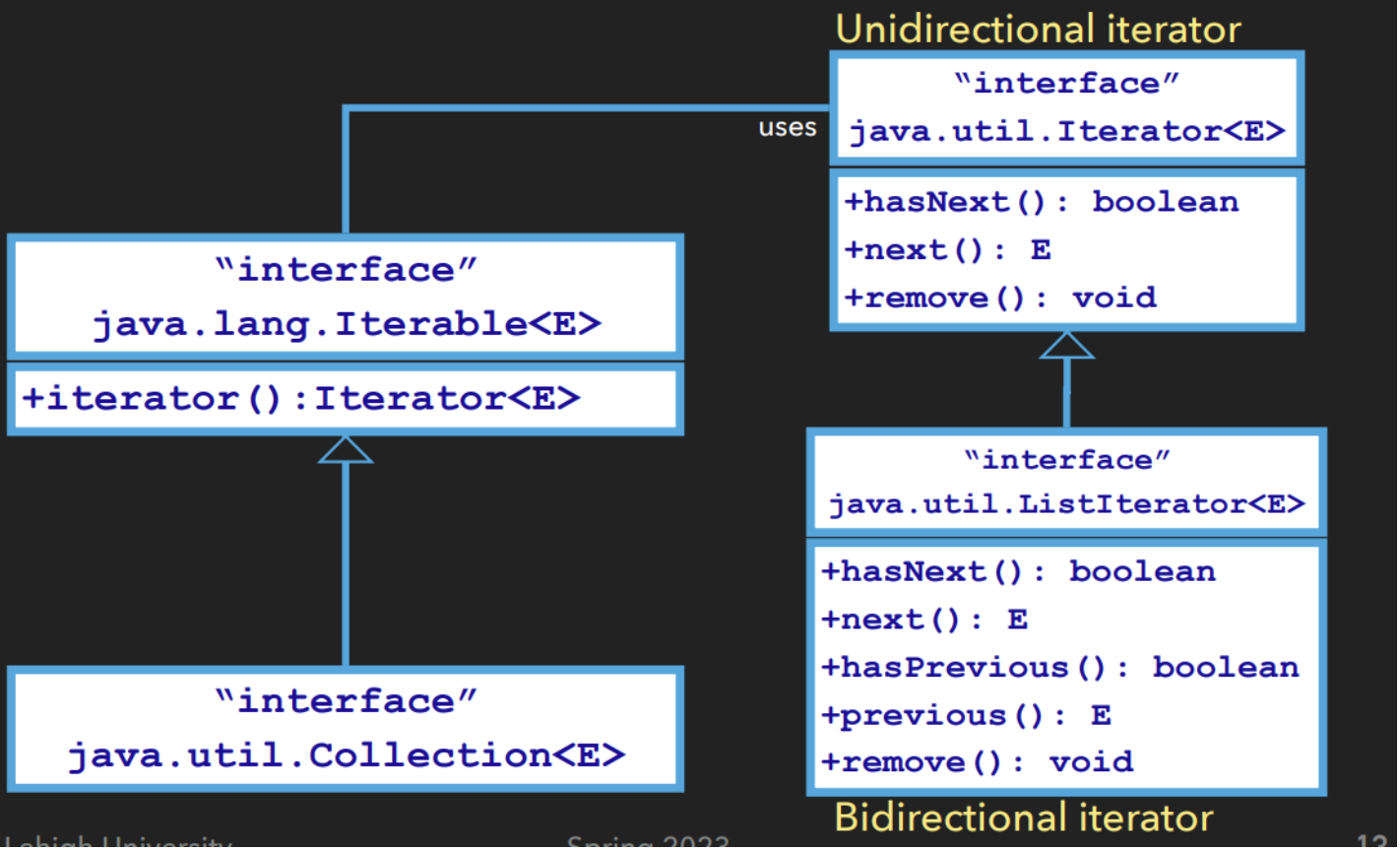


Green = concrete class

"interface"
Java.util.Collection<E>

```
+add(E): boolean
+addAll(Collection<? Extends E>):boolean    (Set Union)
+clear(): void
+contains(Object): boolean
+containsAll(Collection<?>): boolean
+equals(Object): boolean
+remove(Object): boolean
+removeAll(Collection<?>): boolean    (Set difference)
+retainAll(Collection<?>): boolean    (Set intersection)
+size(): int
+toArray(): Object[]
+toArray(T[]): T[]
+iterator():Iterator<E>
```

Java Collection Framework (**Iterators**)



Java Collection Framework (**Algorithms**)

Java.util.Collections

+sort(List): void

+binarySearch(List, Object): int

+reverse(List): void

+shuffle(List): void

+copy(List, List): void

+fill(List, Object): List

+swap(List, int, int):void

TYPES OF DATA STRUCTIRES

Generic Class Ex: `java.util.ArrayList;`

```
+ArrayList()
+ArrayList(int capacity)
+add(int index, E item): void
+add(E item): void
+get(int index): E
+set(int index, E item): E
+remove(int index): boolean
+size(): int
+isEmpty(): boolean
+clear(): void
+contains(Object obj): boolean
+indexOf(Object obj): int
+lastIndexOf(Object obj): int
+remove(Object obj): boolean
+remove(int index): boolean
```

- 2 Constructors:
  - no-arg creates an array of default size 10
  - One-arg creates an array of size `capacity`
- add (overloaded)
  - **`add(int index, E item)`**: adds `item` at location `index`.
    - All elements from `index` to `size()-1` are pushed one position up
  - **`add(E item)`**: adds `item` at first open location
- **`get(int index)`**: returns `item` at `index`
- **`set(int index, E item)`**: replaces element at location `index` with `item`
  - returns the old value of the `item` at `index`
- **`remove(int index): boolean`**
- **`size()`**: returns the actual size of the array (not `capacity`)
- **`isEmpty()`**: returns true if the array is empty
- **`clear()`**: reset size to 0
- **`contains(Object obj)`**: returns true if `obj` is in the array
- **`indexOf(Object obj)`**: returns the first index of `obj` if found, -1 otherwise
- **`lastIndexOf(Object obj)`**: returns the last index of `obj` if found, -1 otherwise
- Remove (overloaded):
  - **`remove(Object obj)`**: Returns true if `obj` is removed, and false otherwise
  - **`remove(int index)`**: Returns true if `index` is valid and element at `index` removed, false otherwise

Stack

`Java.util.Stack<E>`

```
+Stack(): void
+isEmpty(): boolean
+peek(): E
+pop(): E
+push(E): void
+search(Object): int
```

Priority Queue

✦ Priority Queue uses the natural ordering (`compareTo()` from **`Comparable`**) or a comparator (`compare()`)

`java.util.PriorityQueue<E>`

```
+PriorityQueue()
+PriorityQueue(Comparator<? super E> c)
+offer(E): boolean
+poll(): E
+remove(): E
+peek(): E
```

Linked List

Java.util.LinkedList<E>

```
+LinkedList()
+LinkedList(Collection<? Extends E>)
+addFirst(E): void
+addLast(E): void
+getFirst(): E
+getLast(): E
+removeFirst(): E
+removeLast(): E
+listIterator(): ListIterator<E>
+listIterator(int): ListIterator<E>
```

LinkedList: Node-Based List

- Implemented using linked nodes
- Class node is an inner class to LinkedList

Node

```
+value: E
+next: Node
+Node(E)
```

LinkedListIterator

```
+current: E
+hasNext(): boolean
+next(): E
```

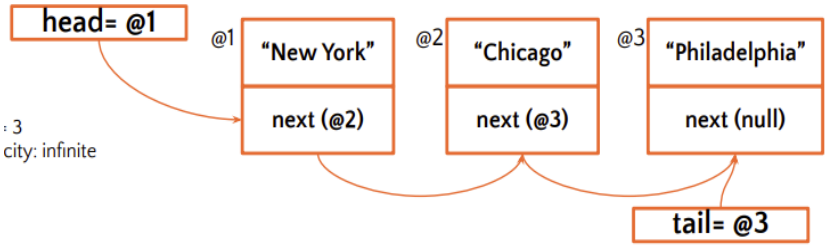
LinkedList<E>

```
-head: Node
-tail: Node
-size: int
+LinkedList()
+addFirst(E): void
+addLast(E): void
+getFirst(): E
+getLast(): E
+removeFirst(): E
+removeLast(): E
+add(E): boolean
+clear(): void
+isEmpty(): boolean
+size(): int
+iterator(): Iterator<E>
```

LinkedList

Traversing the list

```
Node node = head;
while(node != null){
  System.out.println(node.value);
  node = node.next;
}
```



Hash Table Implementation

- Hash Table with chaining
- Array of pointers to linked lists

HashMapEntry<K, V>

```
-key: K
-value: V
+HashMapEntry(K k, V v)
+getKey(): K
+getValue():V
+setKey(K k): void
+setValue(V v): void
+toString(): String
```

HashMap<K, V>

```
-hashTable: LinkedList<HashMapEntry<K,V>>[]
-loadFactor: double
-size: int
+HashMap()
+HashMap(int capacity)
+HashMap(int capacity, double loadFactor)
-trimToPowerOf2(int capacity): int
-hash(int hashCode): int
-rehash(): void
+get(K key): V
+put(K key, V value): V
+remove(K key): void
+containsKey(K key): boolean
+size(): int
+isEmpty(): boolean
+clear(): void
+toString(): String
+toList(): ArrayList<HashMapEntry<K,V>>
```