Collections in Java

Prepared by
Rubyeat Islam
Lec, Dept of CSE
rubyeat88@gamil.com

Topic

- Java Collections
- Collection Framework Hierarchy
- Interface
- List
- Queue
- Set
- Map

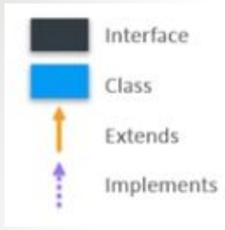
Java Collection

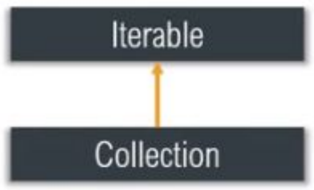
- Collections are the containers that groups multiple items in a single unit. Multi-valued container so they are dynamic containers
- Java collection framework provides an architecture to store and manipulate a group of objects
- Using Java Collections various operations can be performed on the data like searching , sorting , insertion , manipulation deletion etc.
- It provides many interfaces and classes.

Collection Framework Hierarchy

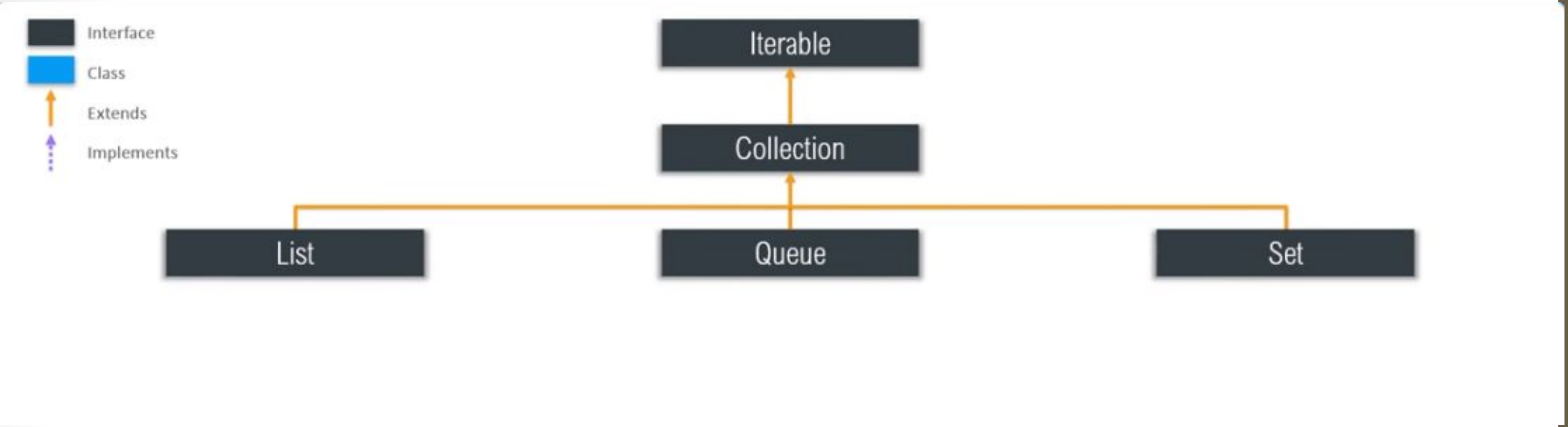
Interface
Class
Extends
Implements
Iterable

Collection Framework Hierarchy

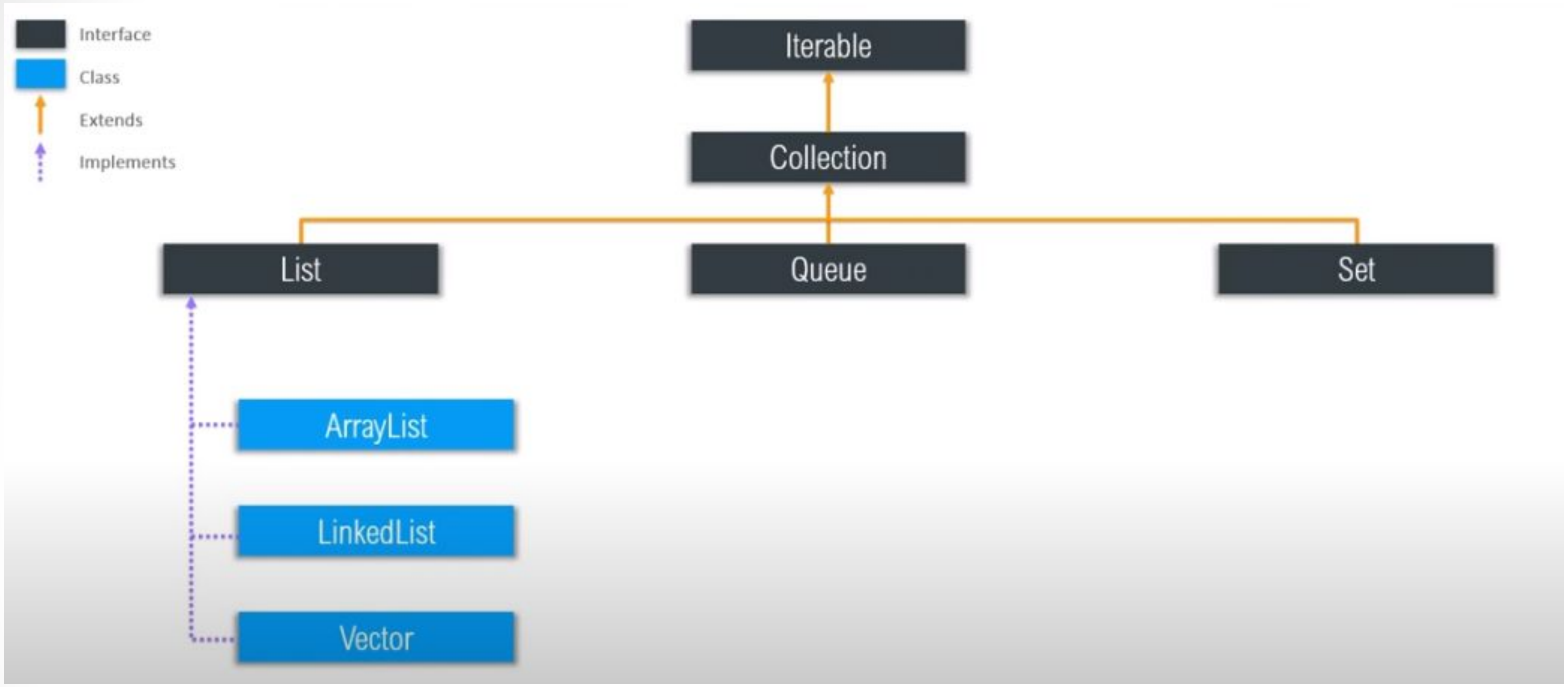




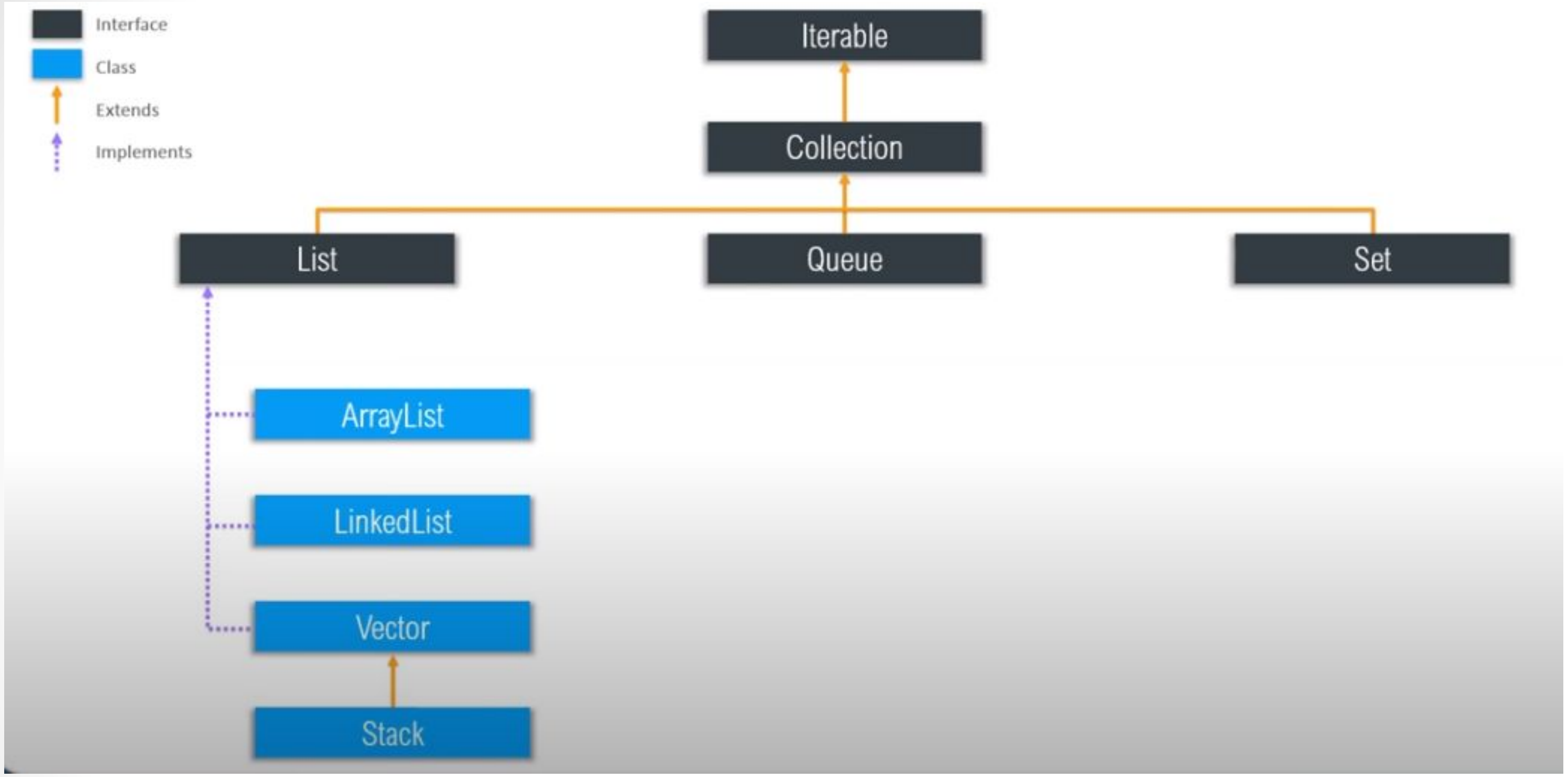
Collection Framework Hierarchy



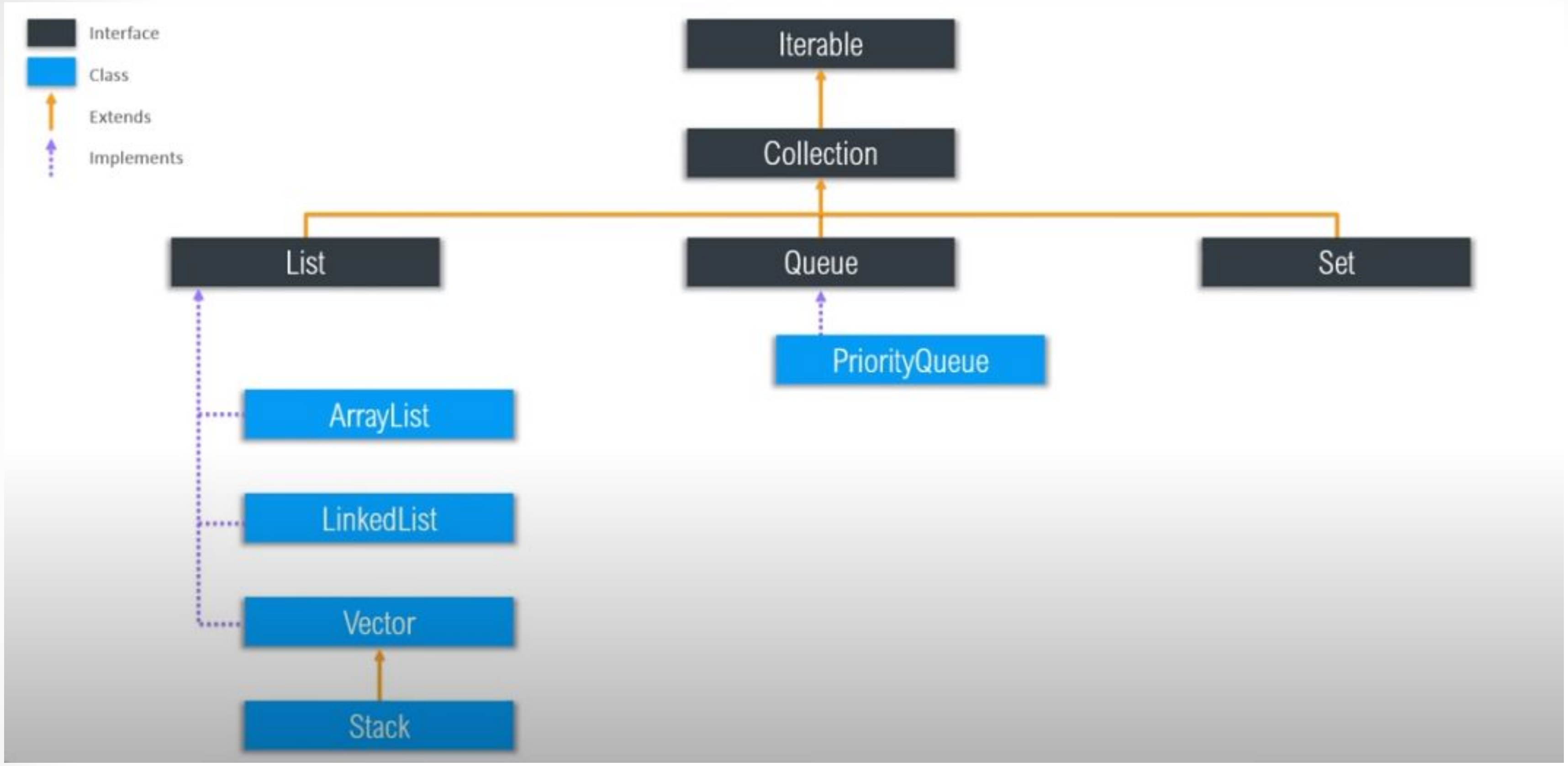
Collection Framework Hierarchy



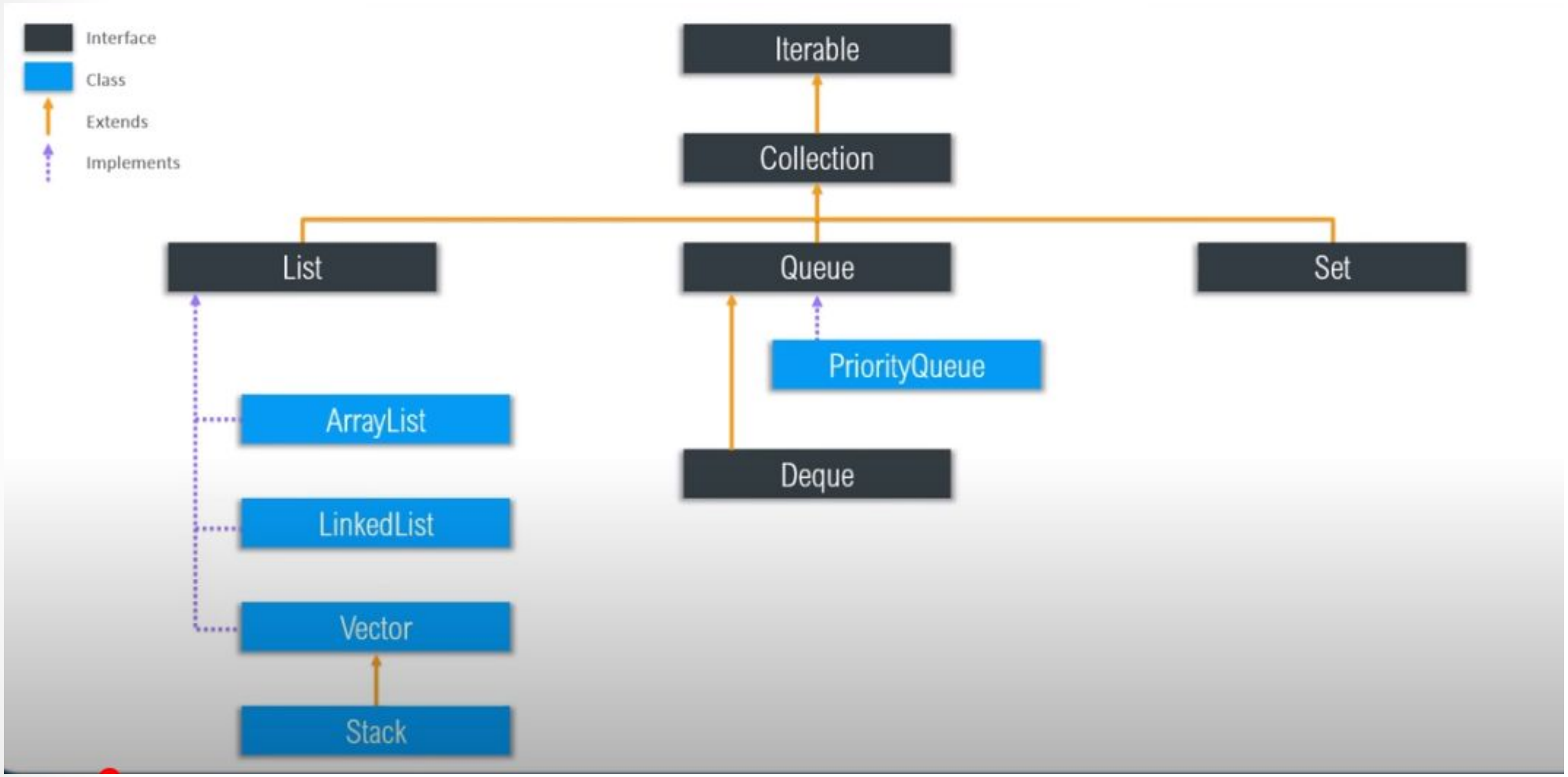
Collection Framework Hierarchy



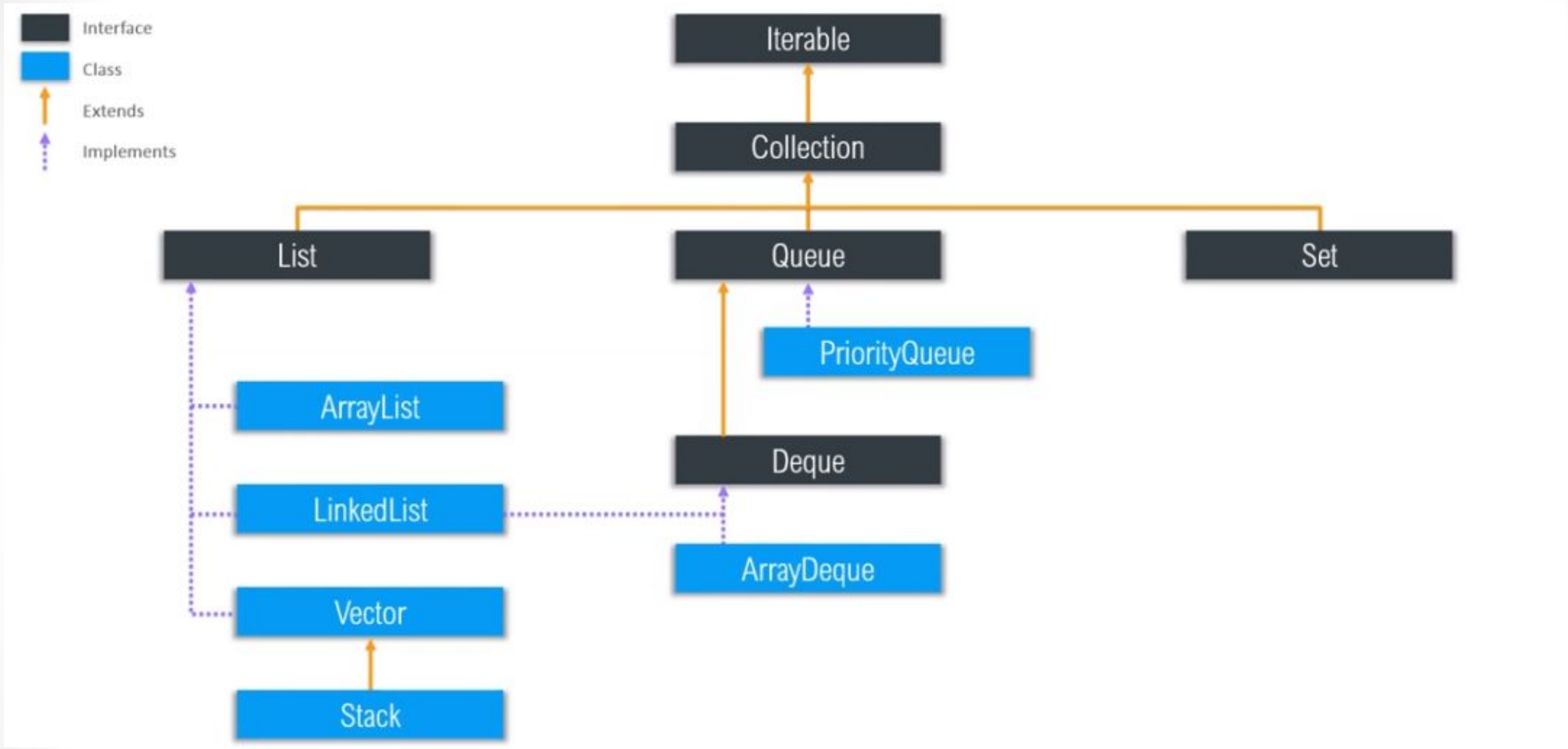
Collection Framework Hierarchy



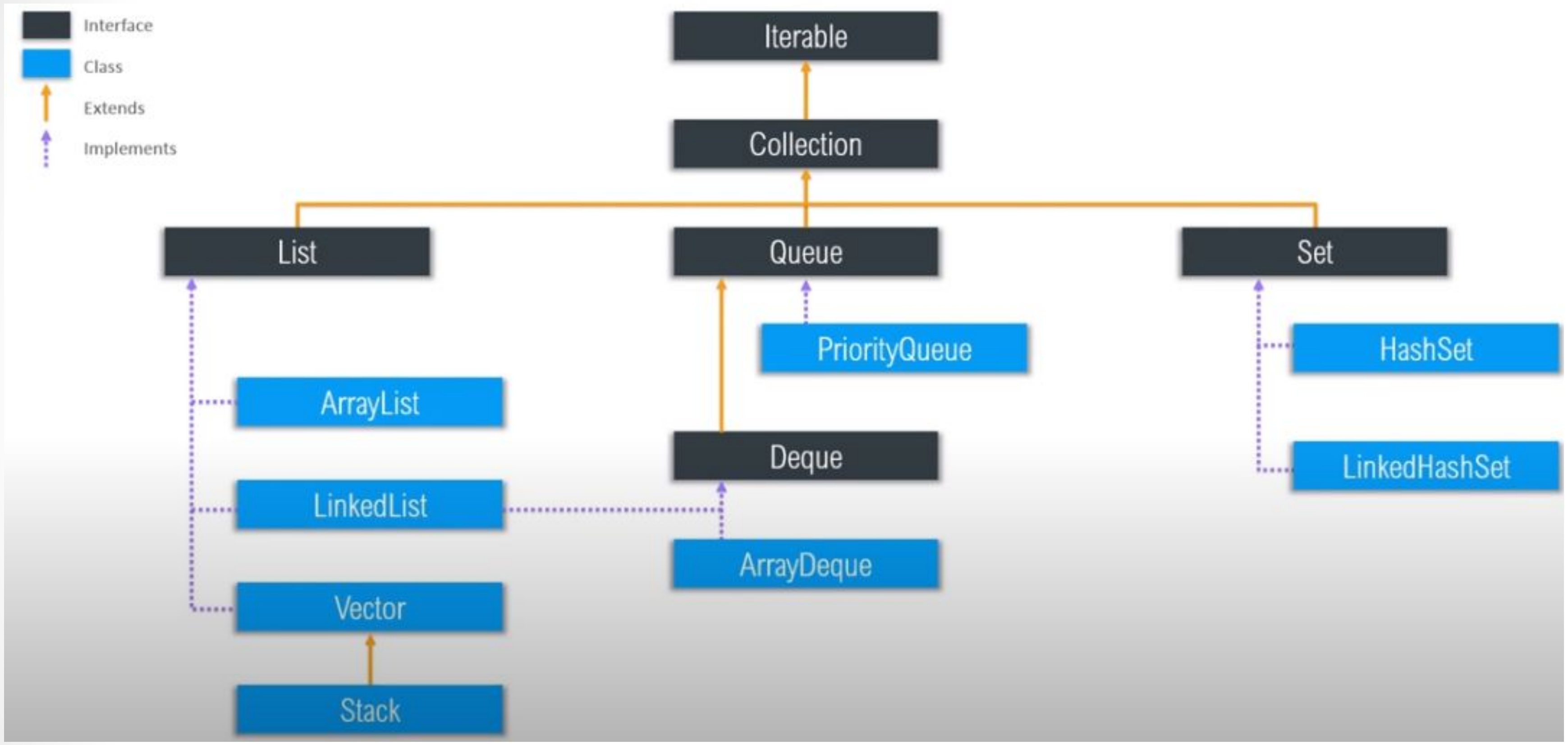
Collection Framework Hierarchy



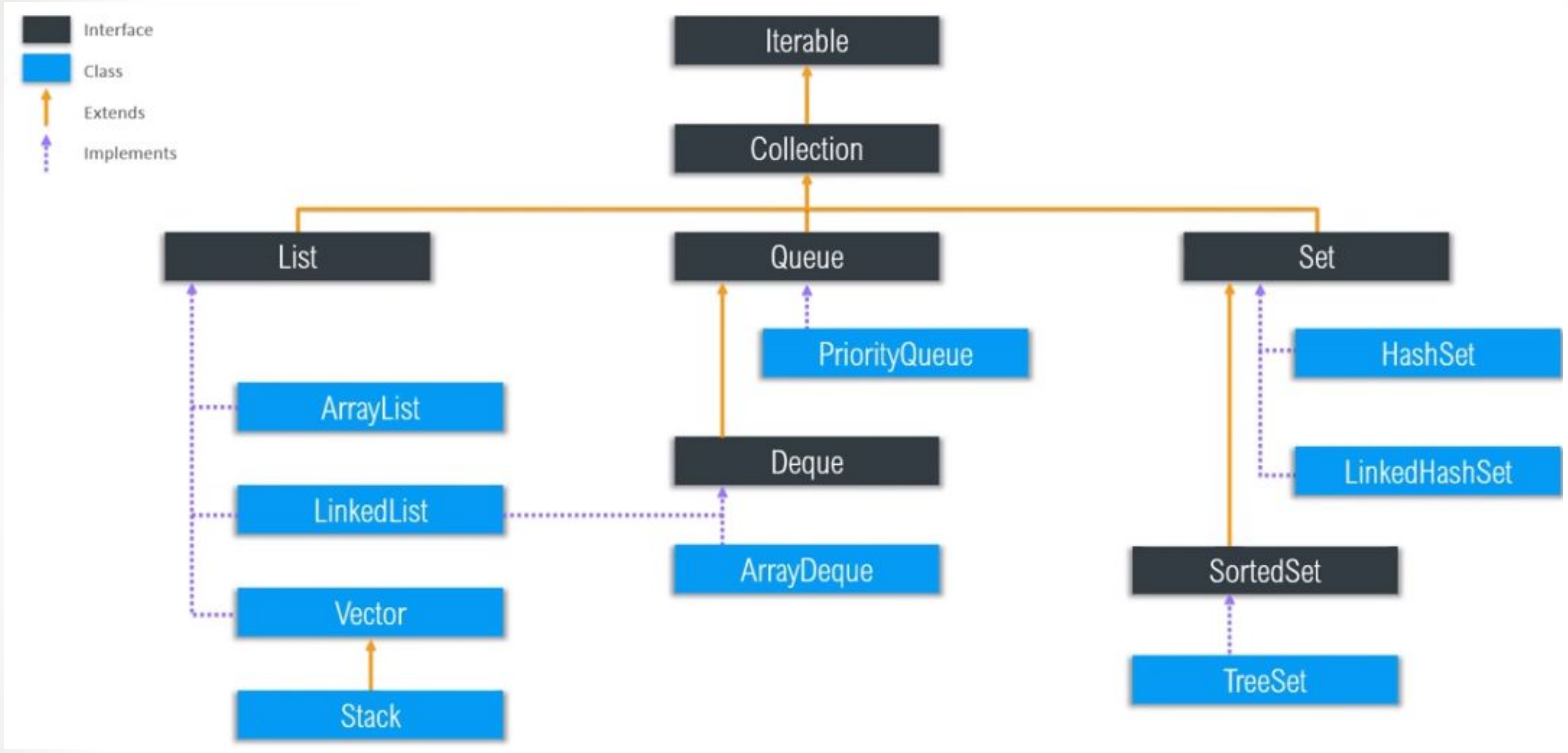
Collection Framework Hierarchy



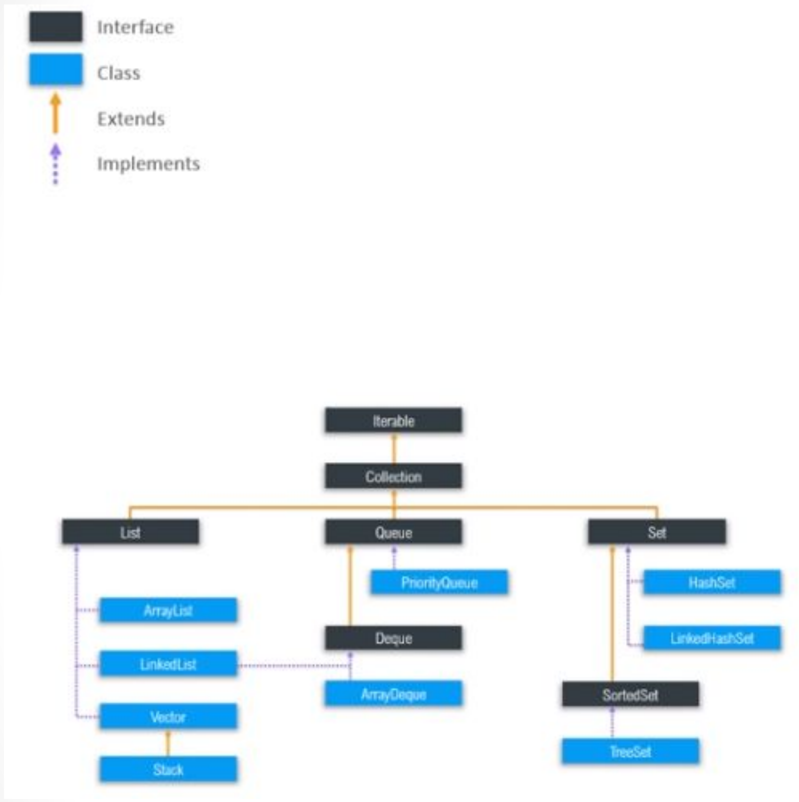
Collection Framework Hierarchy

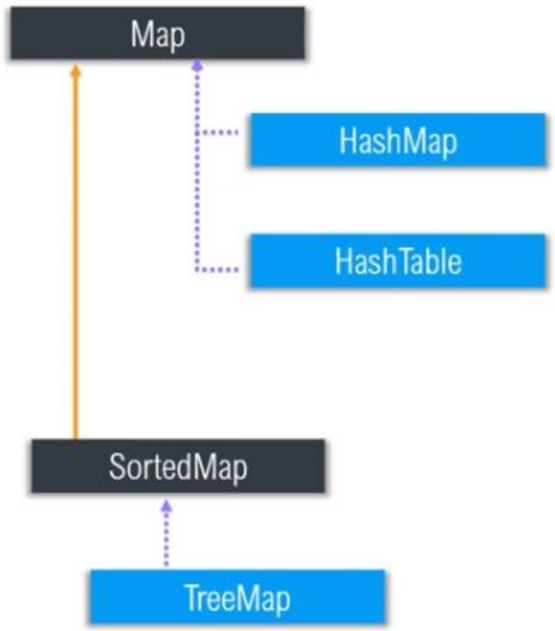


Collection Framework Hierarchy



Collection Framework Hierarchy



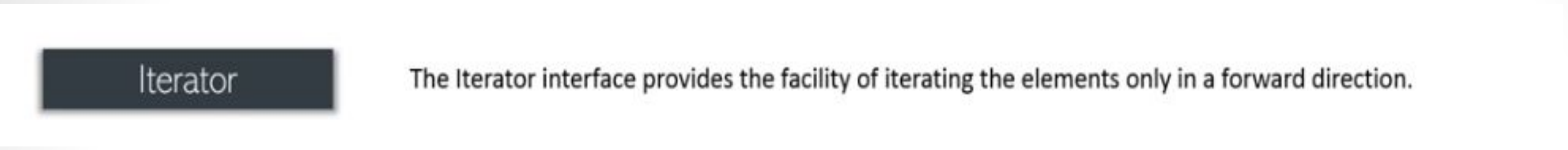


Interface

Interfaces are the reference types similar to classes but only have abstract methods.

- It can not be instantiated
- Do not contain constructors
- Contains only abstract methods
- Is implemented by a class
- Can extend multiple interfaces

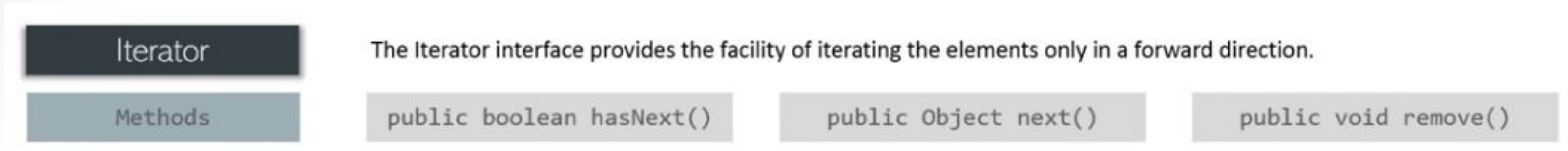
Interface



Iterator

The Iterator interface provides the facility of iterating the elements only in a forward direction.

Interface



Iterator

The Iterator interface provides the facility of iterating the elements only in a forward direction.

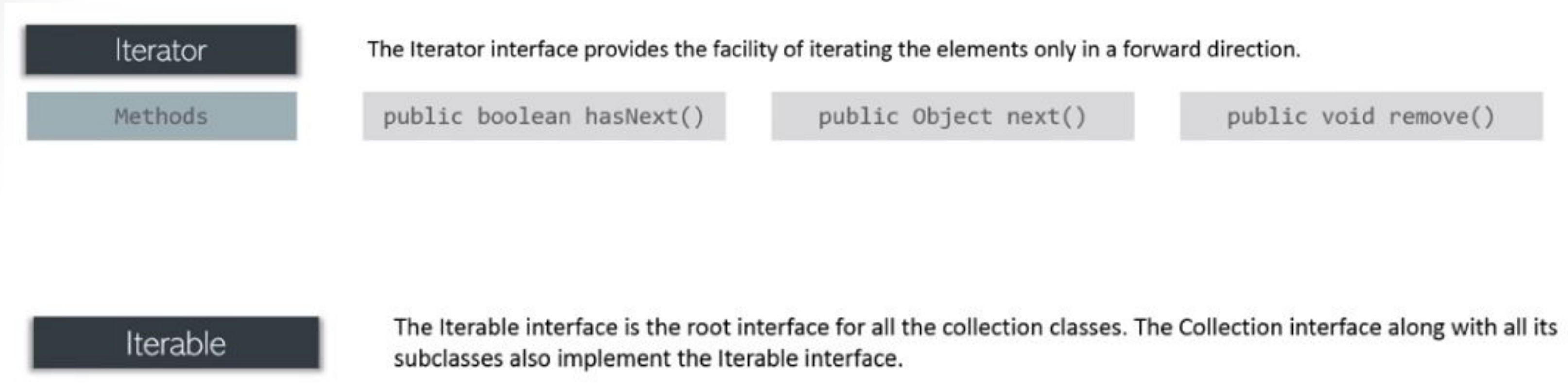
Methods

`public boolean hasNext()`

`public Object next()`

`public void remove()`

Interface



Iterator

The Iterator interface provides the facility of iterating the elements only in a forward direction.

Methods

`public boolean hasNext()`

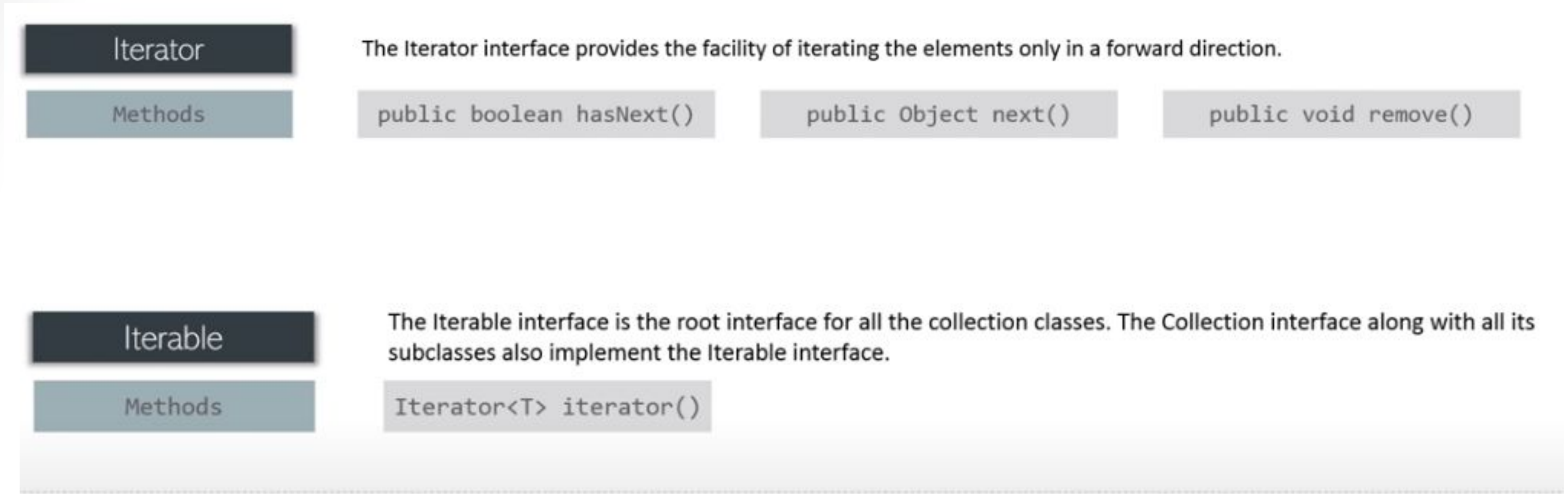
`public Object next()`

`public void remove()`

Iterable

The Iterable interface is the root interface for all the collection classes. The Collection interface along with all its subclasses also implement the Iterable interface.

Interface



Iterator

The Iterator interface provides the facility of iterating the elements only in a forward direction.

Methods

```
public boolean hasNext()
public Object next()
public void remove()
```

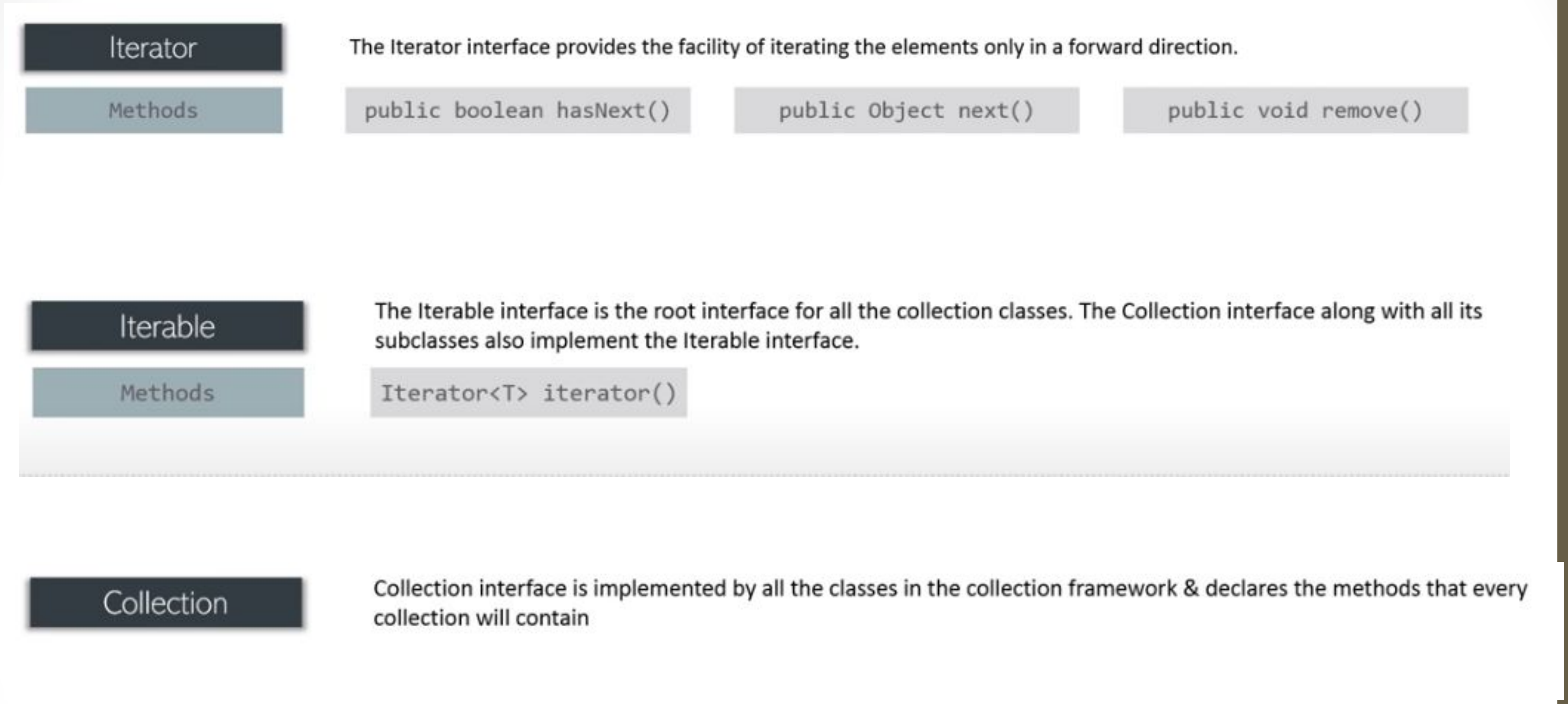
Iterable

The Iterable interface is the root interface for all the collection classes. The Collection interface along with all its subclasses also implement the Iterable interface.

Methods

```
Iterator<T> iterator()
```

Interface



Iterator

The Iterator interface provides the facility of iterating the elements only in a forward direction.

Methods

```
public boolean hasNext()
public Object next()
public void remove()
```

Iterable

The Iterable interface is the root interface for all the collection classes. The Collection interface along with all its subclasses also implement the Iterable interface.

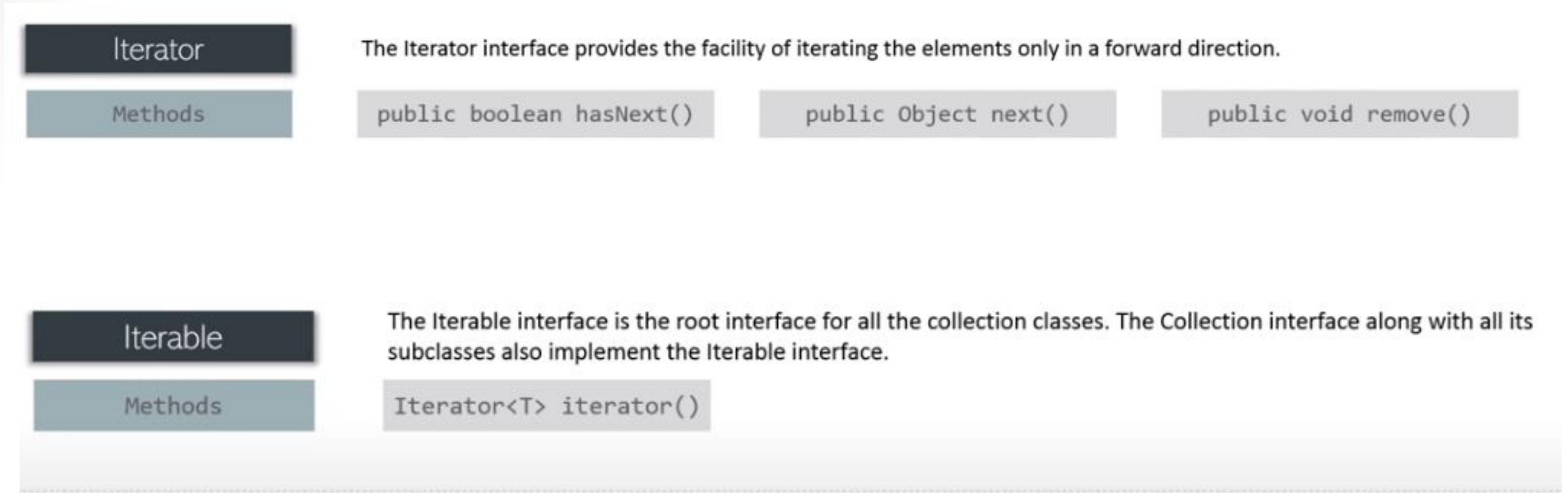
Methods

```
Iterator<T> iterator()
```

Collection

Collection interface is implemented by all the classes in the collection framework & declares the methods that every collection will contain

Interface



Iterator

The Iterator interface provides the facility of iterating the elements only in a forward direction.

Methods

`public boolean hasNext()`

`public Object next()`

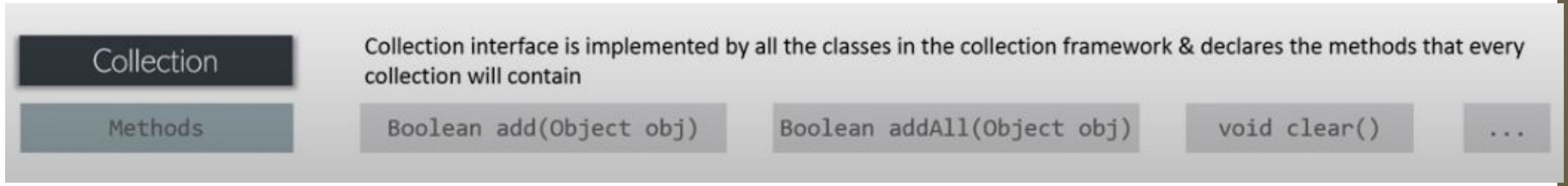
`public void remove()`

Iterable

The Iterable interface is the root interface for all the collection classes. The Collection interface along with all its subclasses also implement the Iterable interface.

Methods

`Iterator<T> iterator()`



Collection

Collection interface is implemented by all the classes in the collection framework & declares the methods that every collection will contain

Methods

`Boolean add(Object obj)`

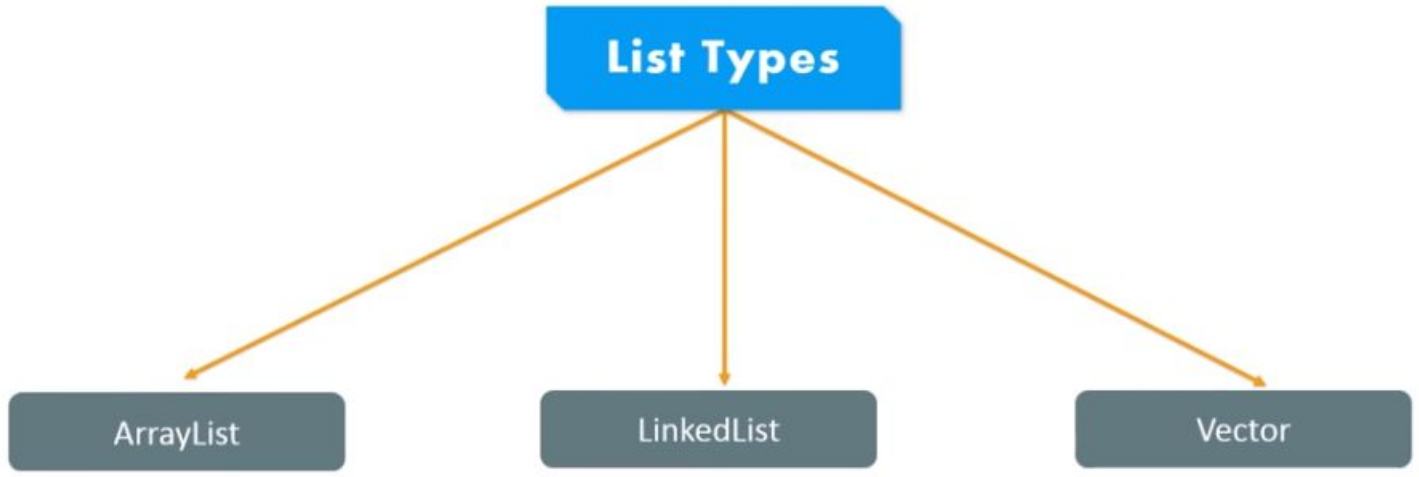
`Boolean addAll(Object obj)`

`void clear()`

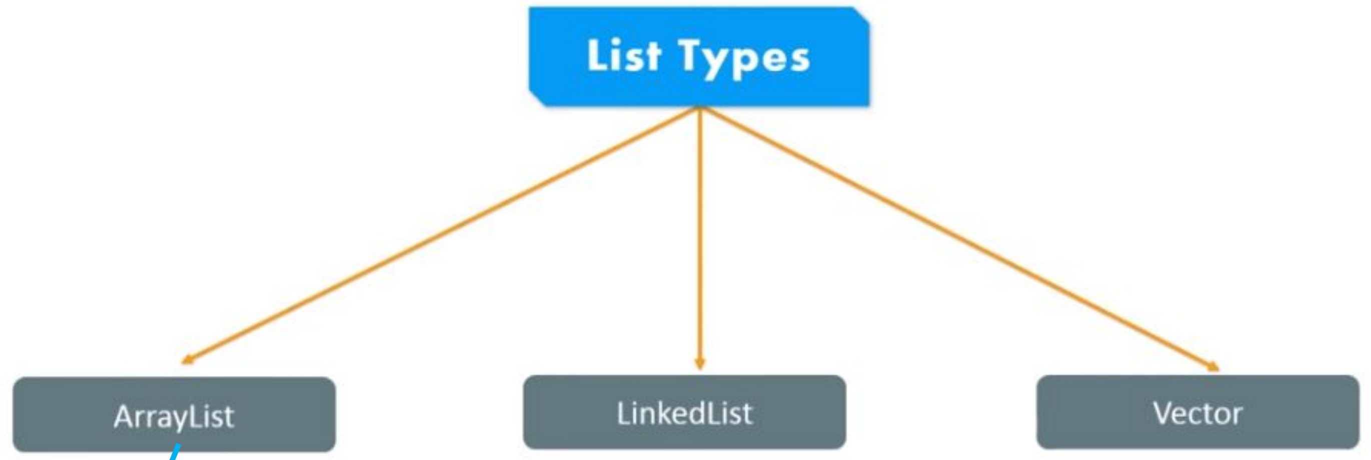
`...`

Java Lists

- List is an interface that extends the Collection interface and contains elements of ordered collection including duplicate elements.
- Cares about which position each object is in

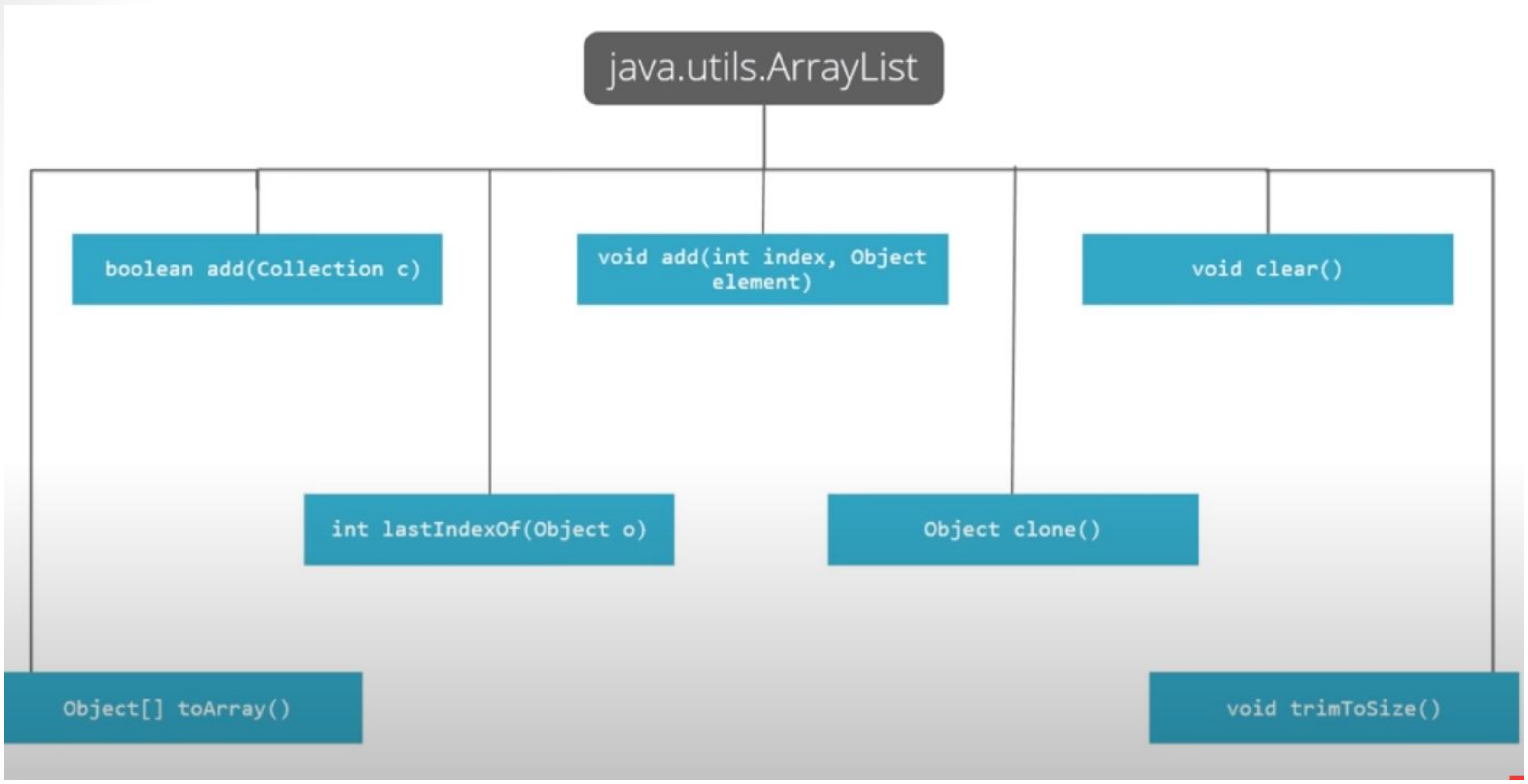


Java Lists

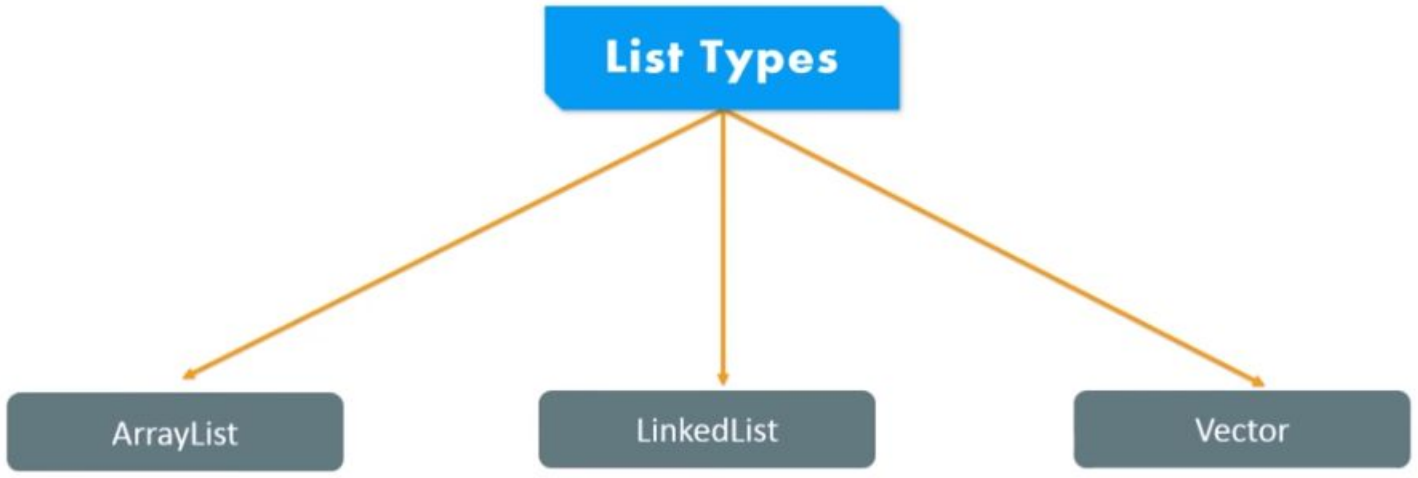


- ArrayList is the implementation of List interface where the elements can be dynamically added or removed from the list
- The size of the list is increased Dynamically if the elements are added more than the initial size
- Insertion and deletion is slower than Compared to linked List but Iteration is faster

ArrayList object = new **ArrayList()**;


java.utils.ArrayList
boolean add(Collection c)
void add(int index, Object element)
void clear()
int lastIndexOf(Object o)
Object clone()
Object[] toArray()
void trimToSize()

Java Lists



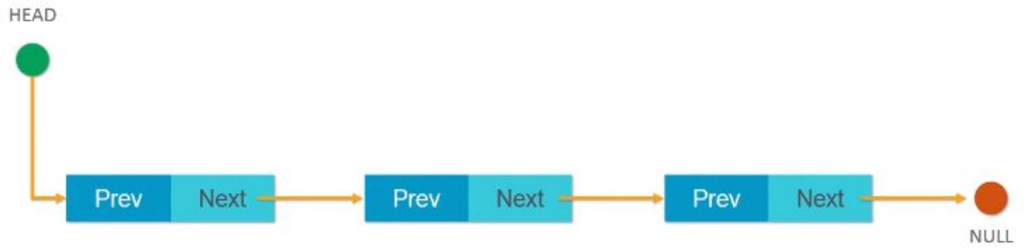
- Linked List is a sequence of
links which contains items
- Each link contains a
connection to another link
- Insertion and deletion is faster than
Compared to ArrayList but Iteration is slower

Linkedlist object = new **Linkedlist()** ;

Singly Linked List

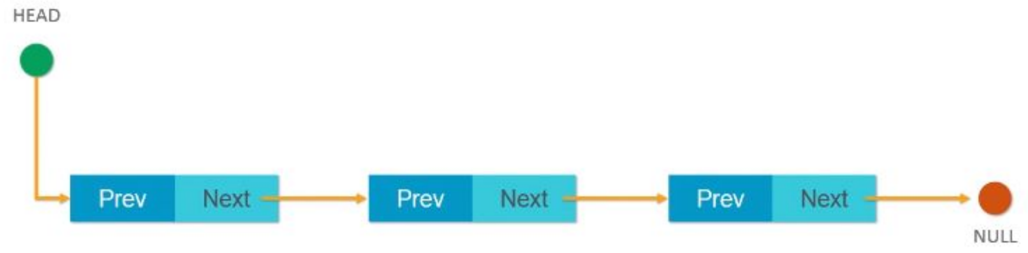
Doubly Linked List

Each node in this list stores the data of the node and a pointer or reference to the next node in the list



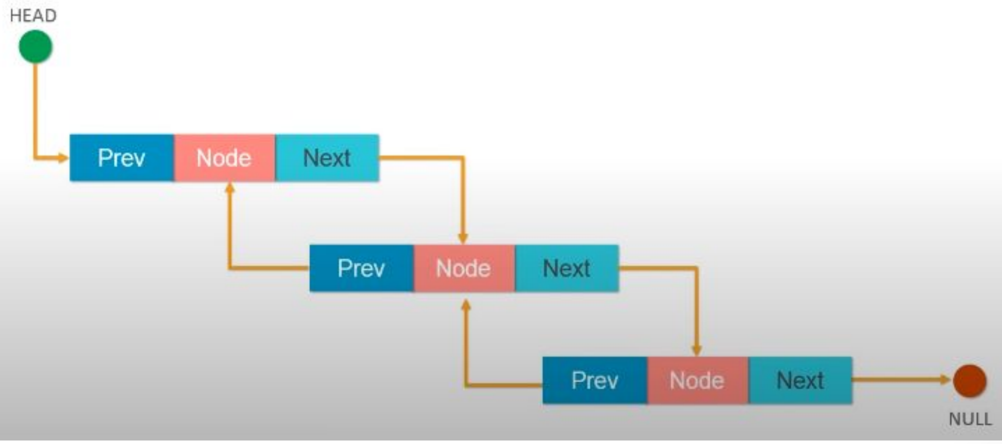
Singly Linked List | Doubly Linked List

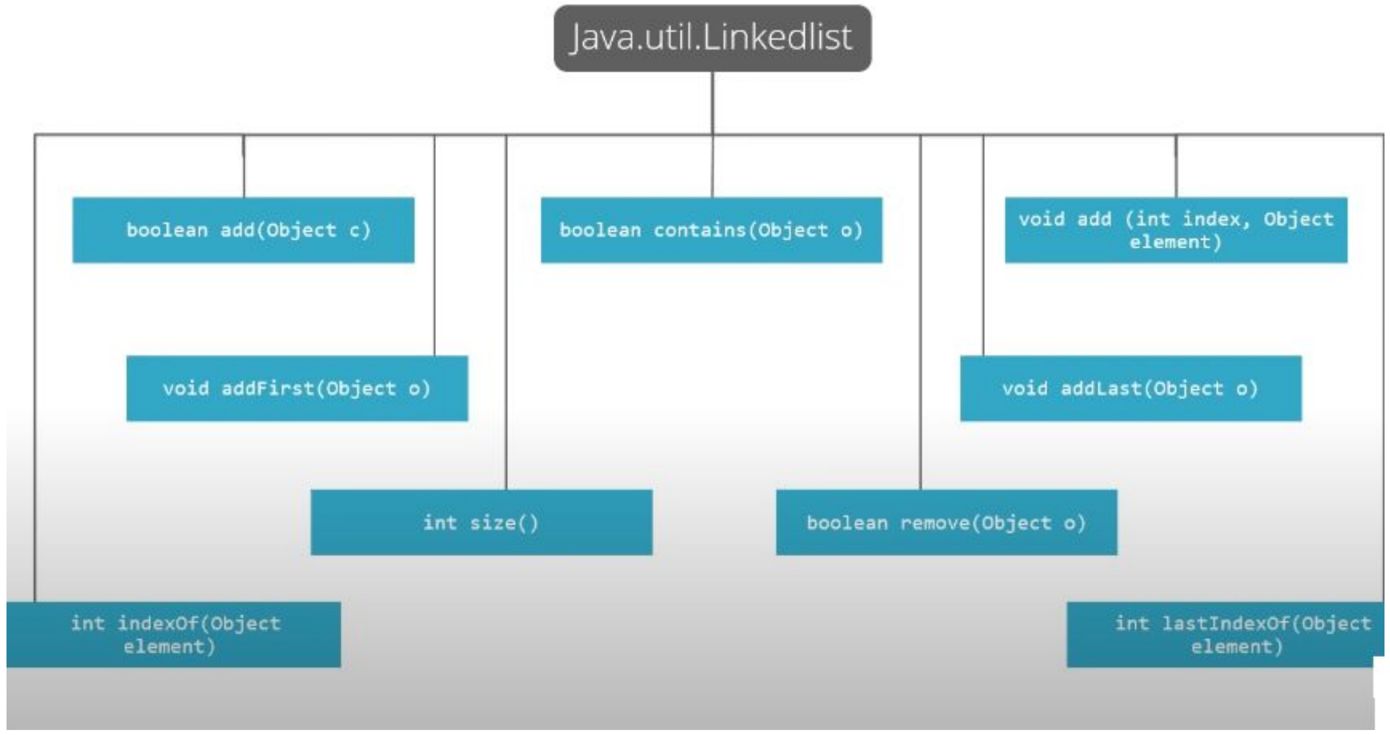
Each node in this list stores the data of the node and a pointer or reference to the next node in the list



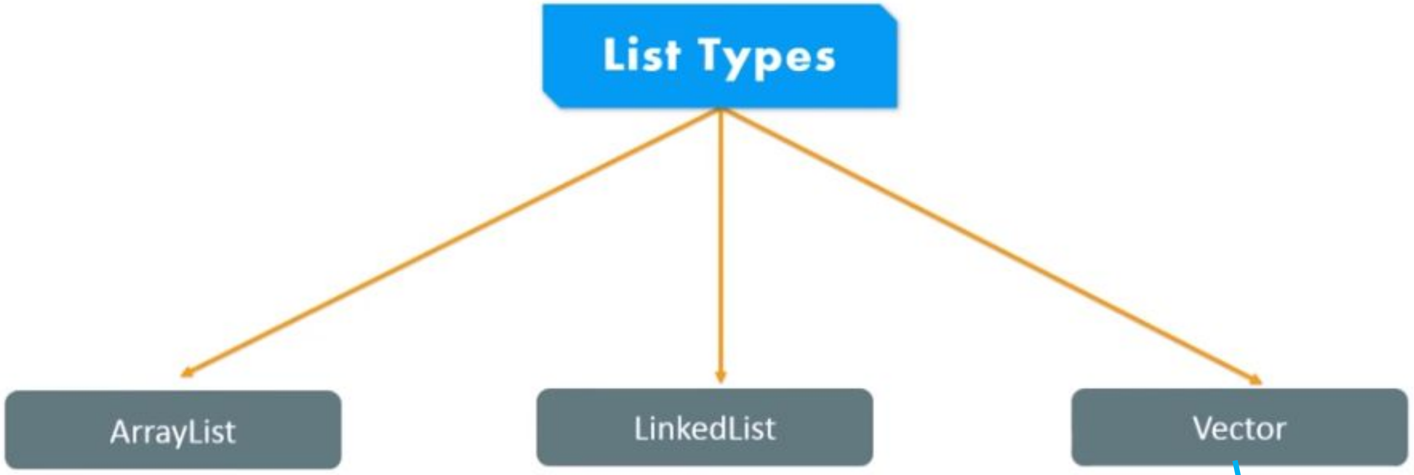
Singly Linked List | **Doubly Linked List**

Doubly Linked list has two references: one to the next node and another to previous node




Java.util.Linkedlist
boolean add(Object c)
boolean contains(Object o)
void add (int index, Object element)
void addFirst(Object o)
void addLast(Object o)
int size()
boolean remove(Object o)
int indexOf(Object element)
int lastIndexOf(Object element)

Java Lists



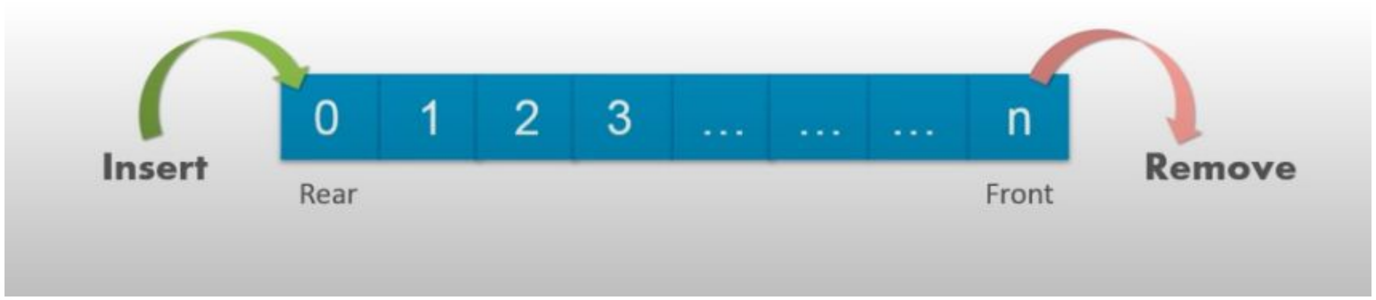
- Vectors are similar to arrays,
where the elements of vector object can accessed by an index
- Implements a dynamic array and is **Synchronized which is thread safe.**

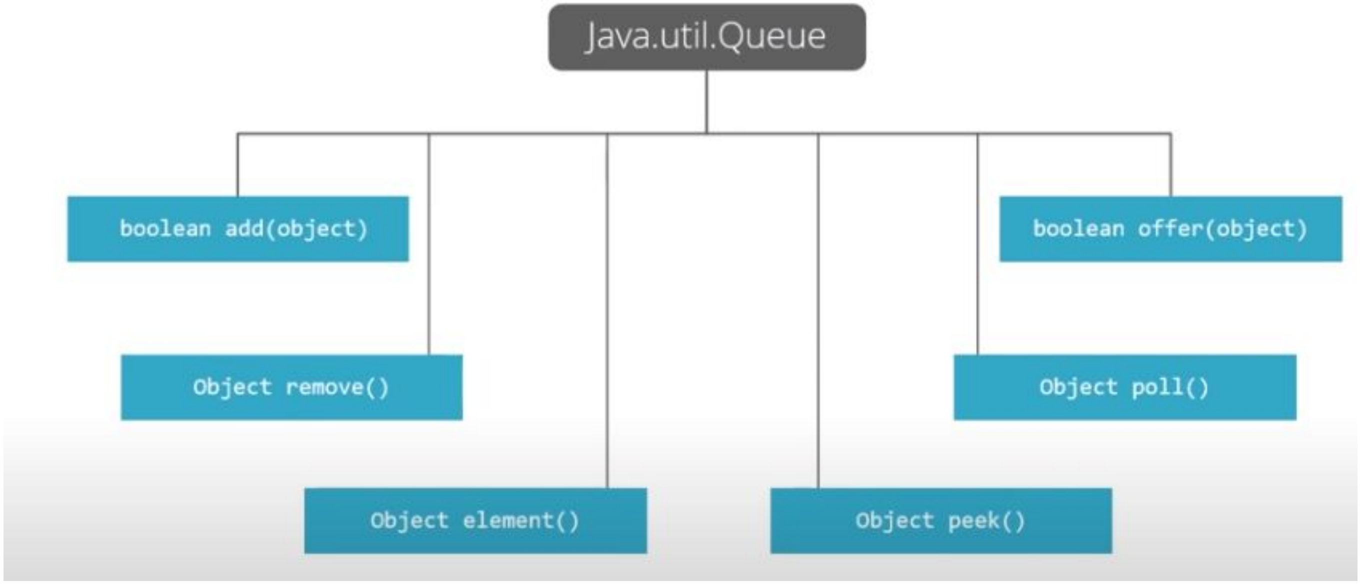
Vector object = new Vector (size, increment)

Java Queue

- Queue in Java follows a FIFO approach i.e . it orders the elements in First in First Out manner
- The first element is removed first and last element is removed in the end
- Arranged in Order Processing. A to-do list for example

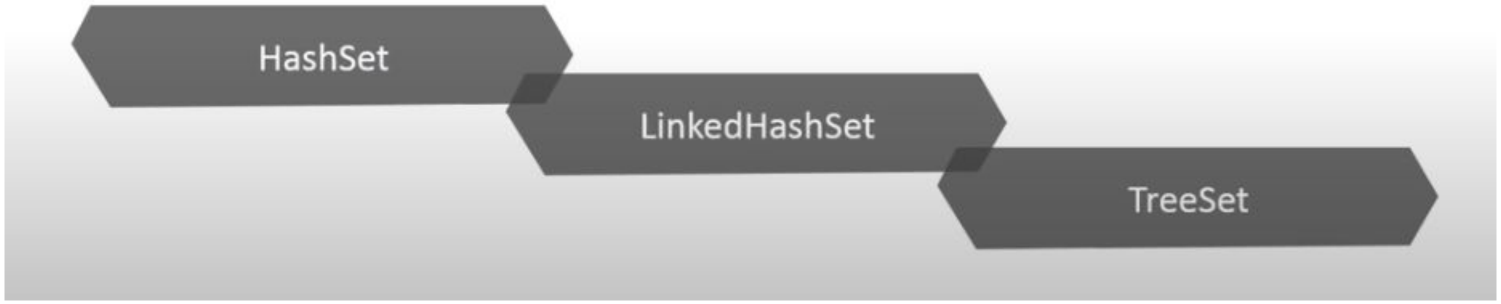
Queue <**Integer**> q = new **Queue<Integer> ()**;



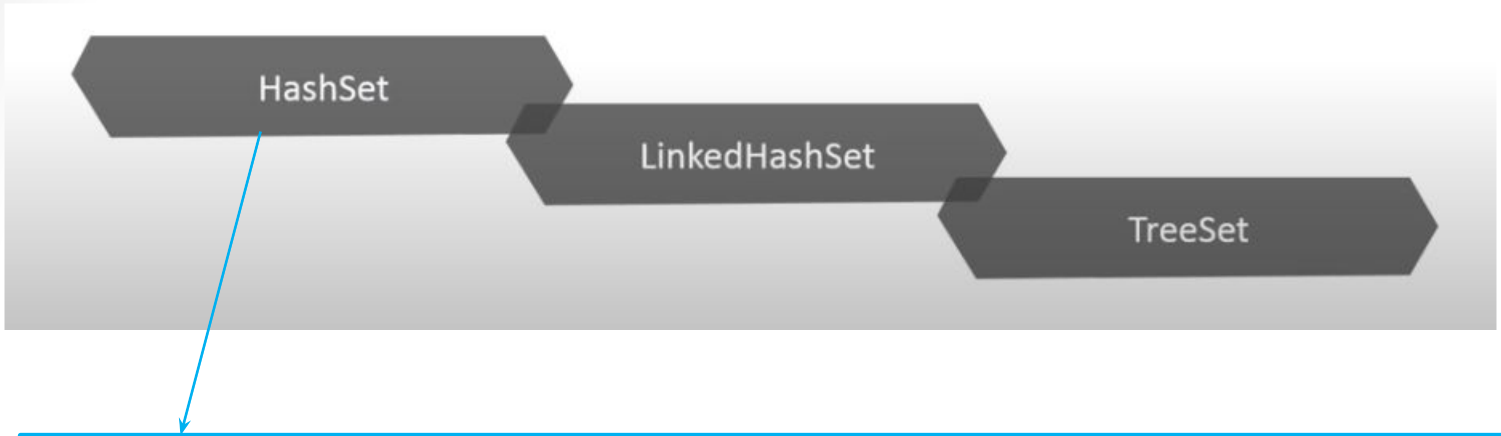

Java.util.Queue
boolean add(object)
boolean offer(object)
Object remove()
Object poll()
Object element()
Object peek()

Java Sets

- A Set refers to a collection that cannot contain duplicate elements.
- **If obj1 equals obj2 only one object will be in the set**
- It is mainly used to model mathematical set abstraction
- Set has its implementation in various classes

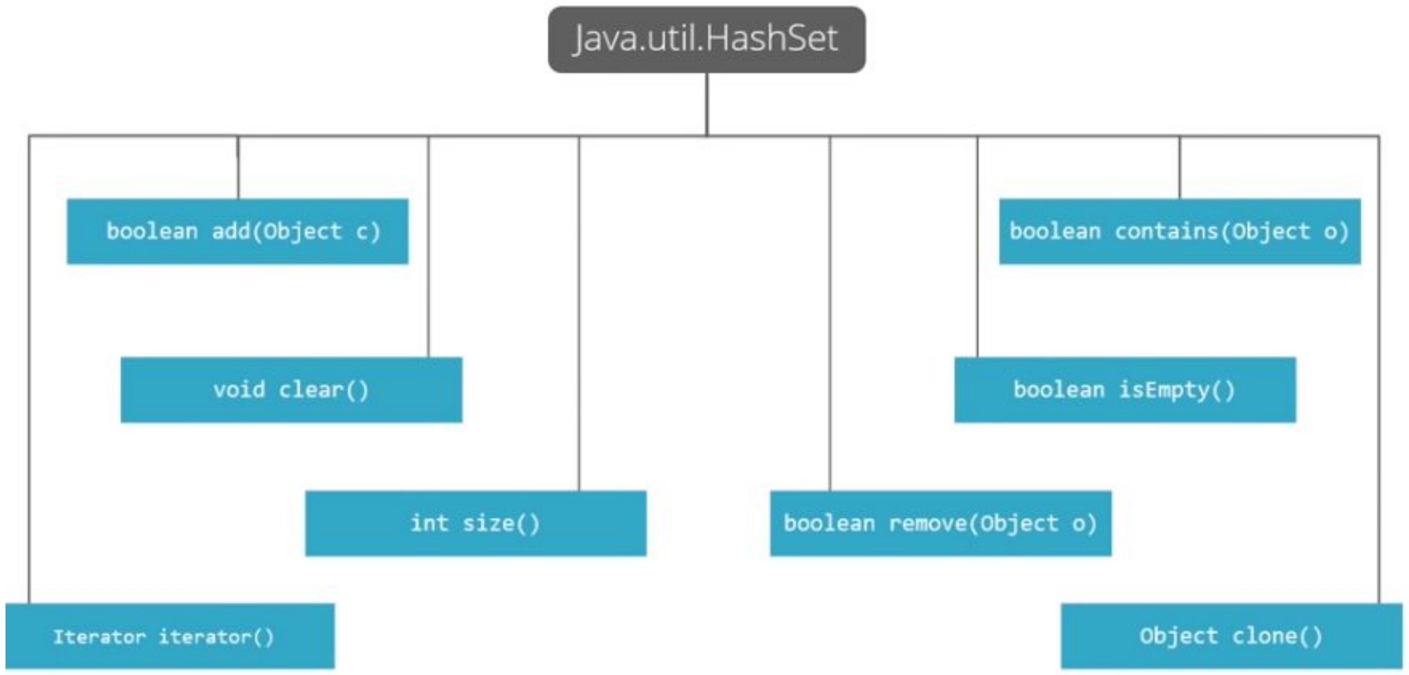


Java Sets



- Does not guarantee order of elements during insertion
- Java HashSet class creates a collection that use a hash table for storage
- Hashset only contain unique elements and it inherits the AbstractSet class and implements Set interface
- It uses a mechanism hashing to store the elements

```
HashSet <String> a1 = new HashSet();
```


Java.util.HashSet
boolean add(Object c)
boolean contains(Object o)
void clear()
boolean isEmpty()
int size()
boolean remove(Object o)
Iterator iterator()
Object clone()

Java Sets

HashSet
LinkedHashSet
TreeSet

- LinkedHashSet class is a Hash table and Lined list implementation of the set interface
- Contains only unique elements
- Provides all optional set operations and **maintains insertion order**

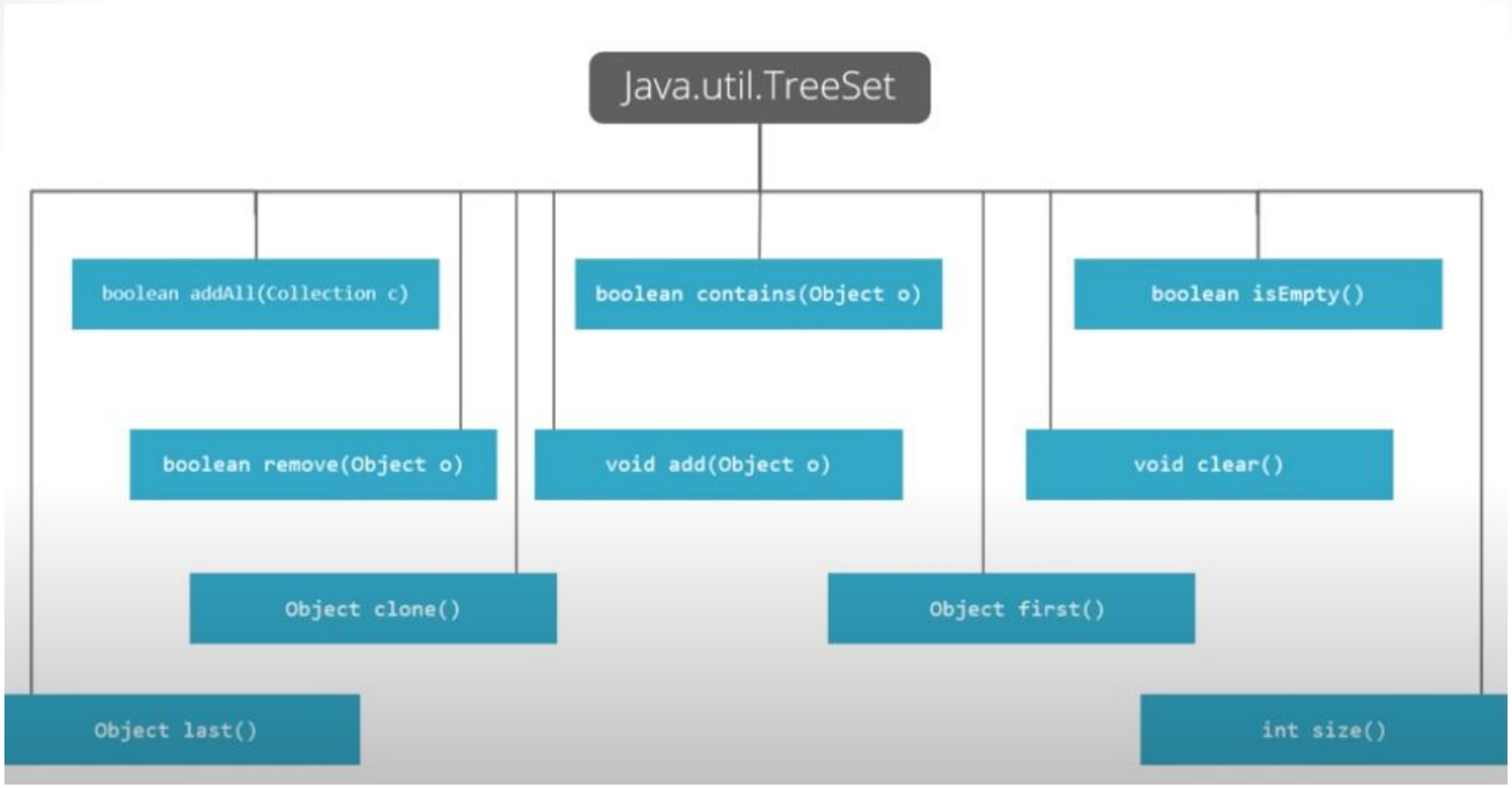
```
LinkedHashSet <String> a1 = new LinkedHashSet();
```

Java Sets

HashSet
LinkedHashSet
TreeSet

- TreeSet class implements the Set interface that uses a tree for storage
- The objects of this class are unique and **are stored in the ascending order**
- It inherits AbstractSet class and implements NavigableSet interface

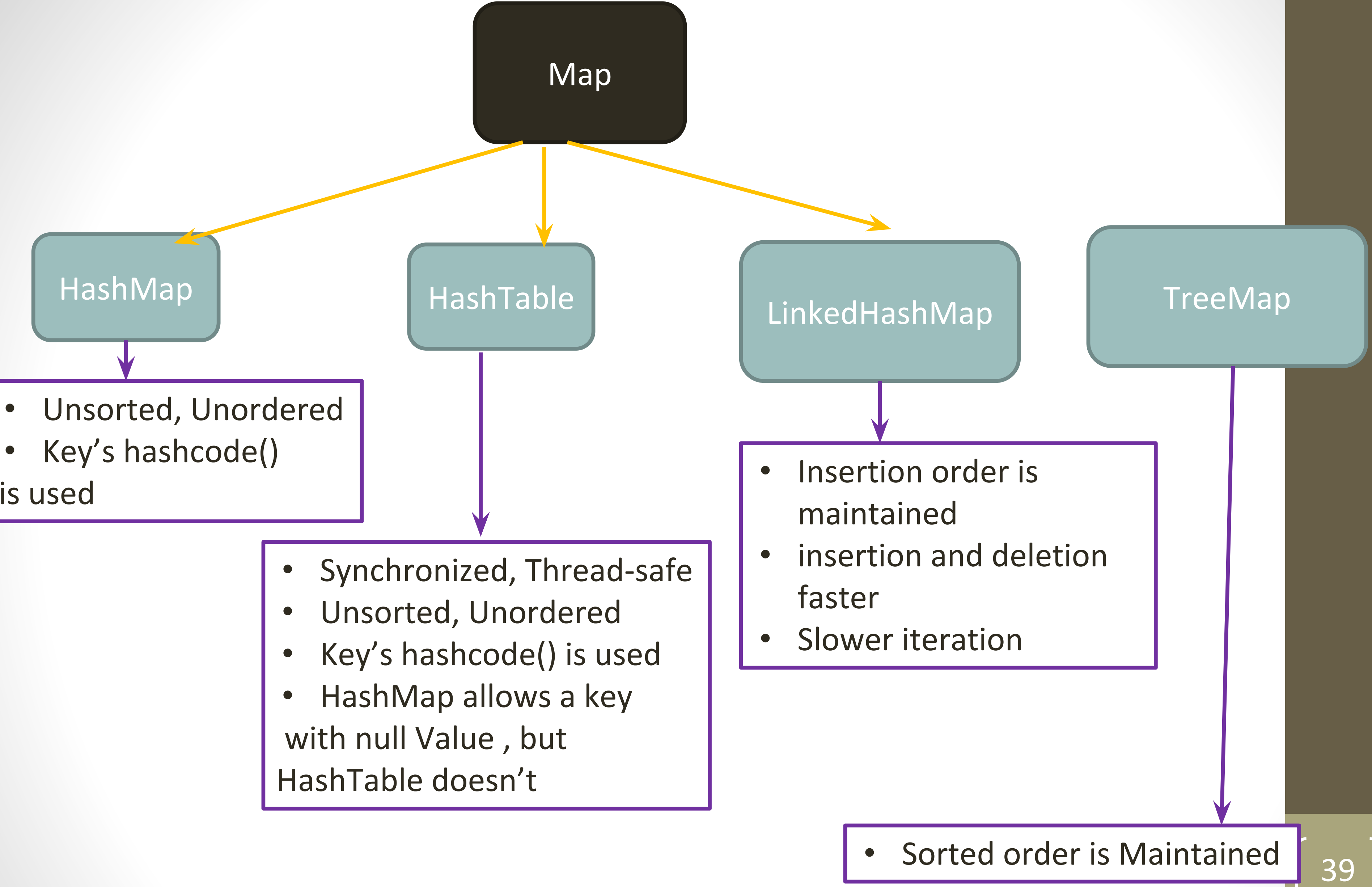
```
TreeSet <String> a1 = new TreeSet<String> ();
```


Java.util.TreeSet
boolean addAll(Collection c)
boolean contains(Object o)
boolean isEmpty()
boolean remove(Object o)
void add(Object o)
void clear()
Object clone()
Object first()
Object last()
int size()

Map

- Another interface is Map
- Although Map is interface in Collection Hierarchy **it doesn’t extends Collection interface**
- Map is used to store the Key-Value pair

⍰ A, C, A, C, E, C, M, D, H, A:: {(A, 3), (C, 3)}



- Unsorted, Unordered
- Key’s hashcode()
is used

- Synchronized, Thread-safe
- Unsorted, Unordered
- Key’s hashcode() is used
- HashMap allows a key
with null Value , but
HashTable doesn’t

- Insertion order is maintained
- insertion and deletion faster
- Slower iteration

- Sorted order is Maintained

KEEP PRACTICING