JAVA: COLLECTIONS FRAMEWORK

Anastasiya Solodkaya

24 сентября 2016 г.

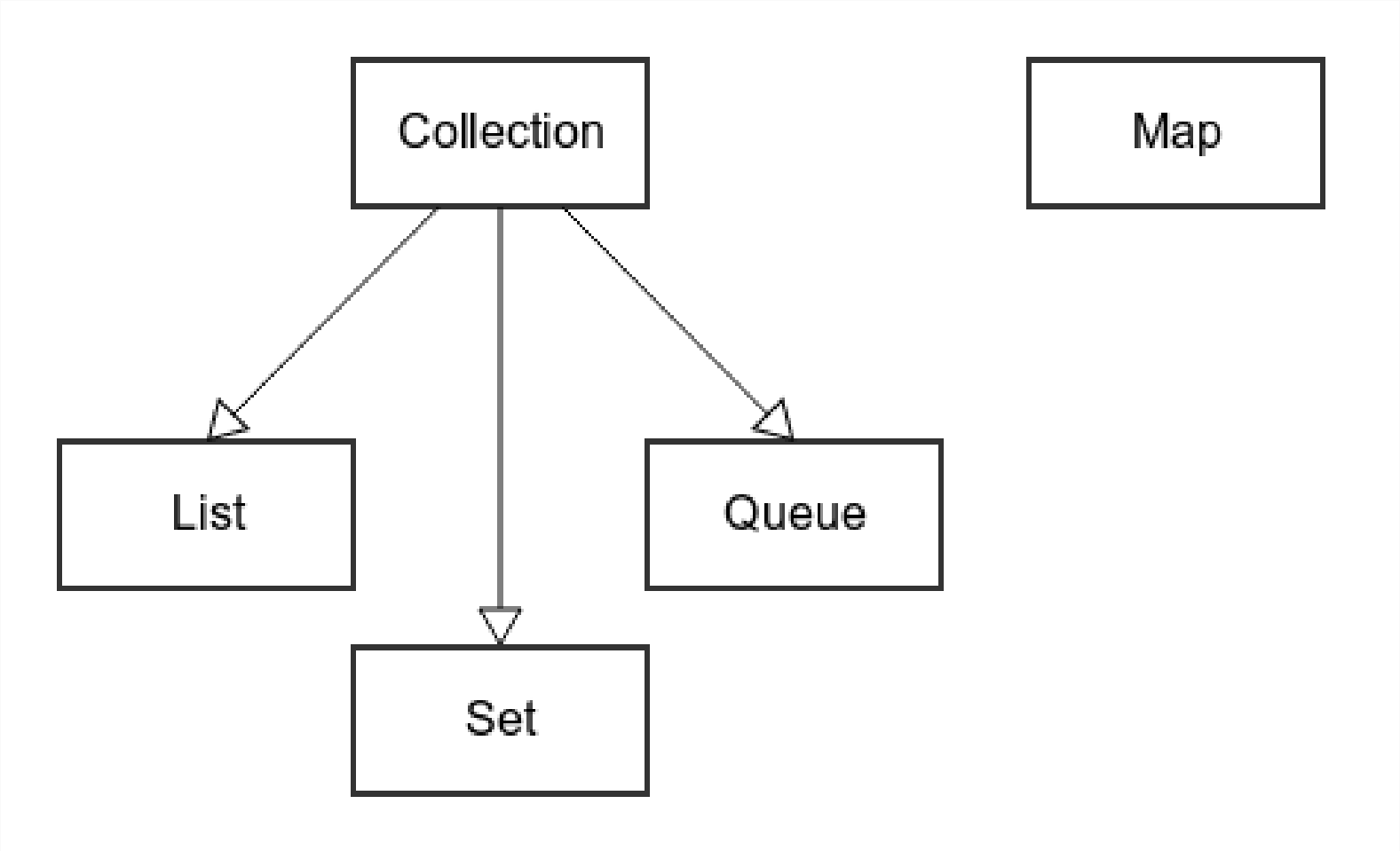
LevelUP

COLLECTION FRAMEWORK

COLLECTION FRAMEWORK

- Коллекция – набор объектов, заключенных в одном объекте.
- java.util и java.util.concurrent

ОСНОВНЫЕ ИНТЕРФЕЙСЫ

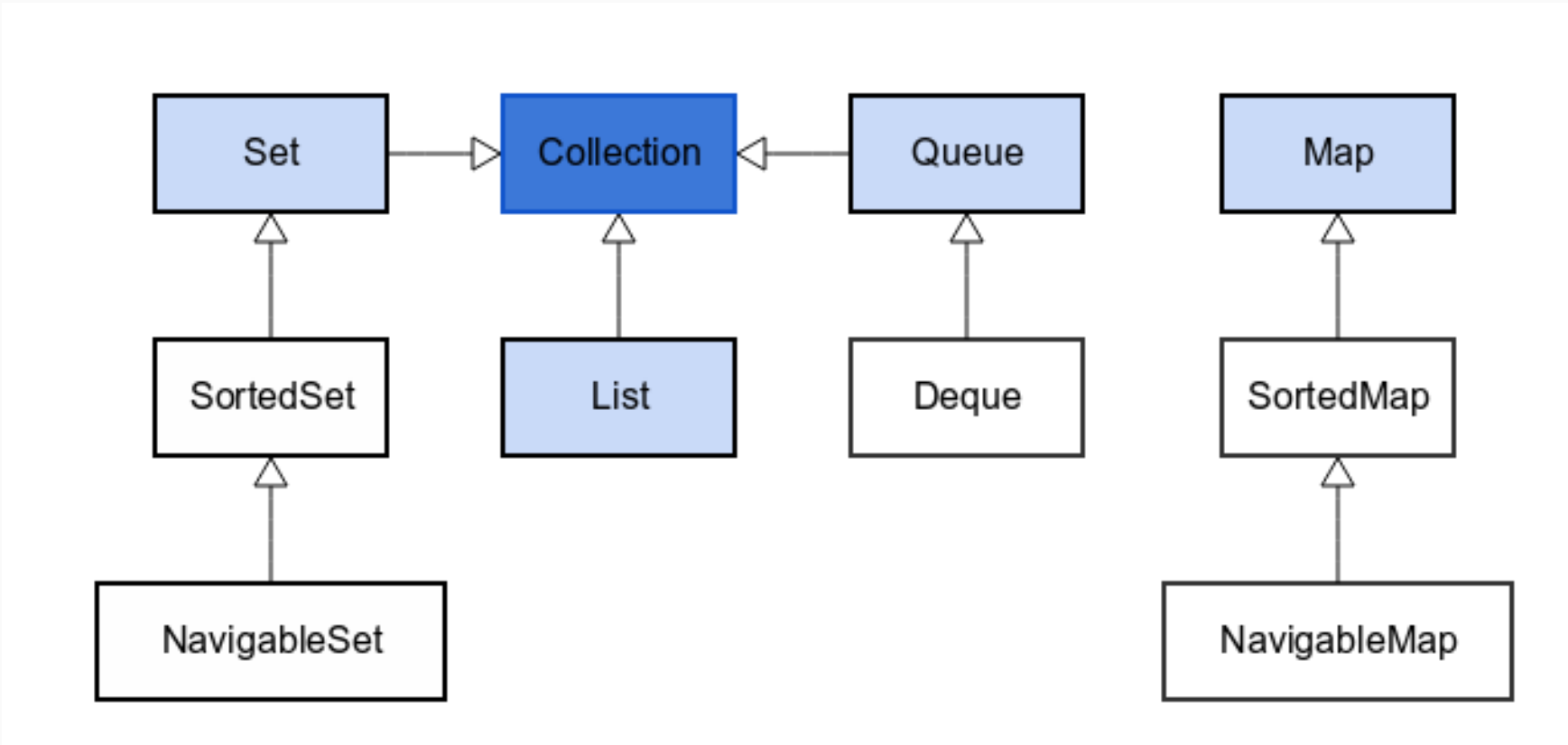


ОСНОВНЫЕ ИНТЕРФЕЙСЫ

- List
  - упорядочен
  - доступ по индексу
  - разрешены дубликаты
- Set
  - просто набор элементов
  - дубликаты не разрешены
- Queue- упорядоченные элементы, как правило порядок либо FIFO, либо LIFO
- Map - набор пар ключ-значение

МЕТОДЫ ИНТЕРФЕЙСОВ

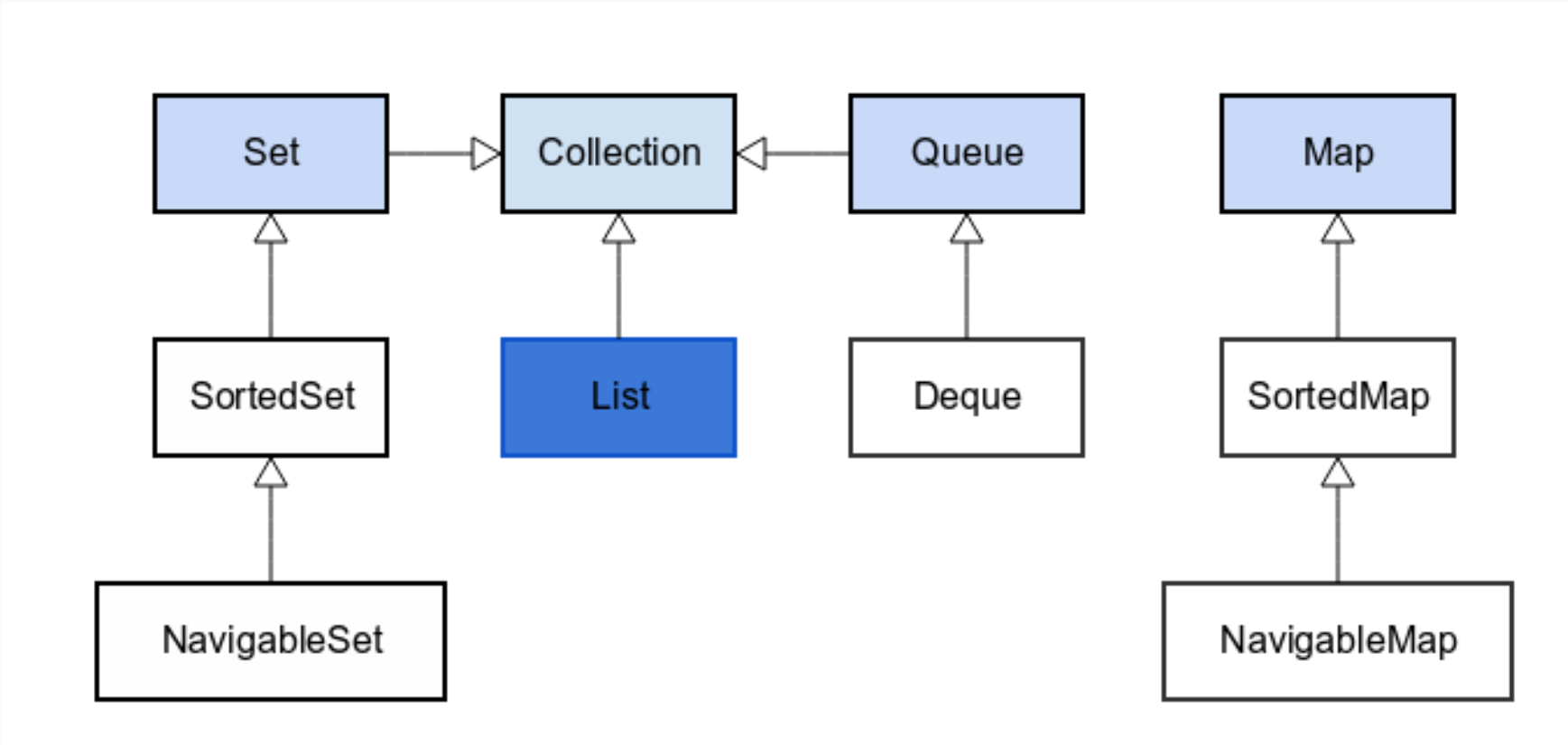
ОСНОВНЫЕ ИНТЕРФЕЙСЫ



COLLECTION INTERFACE

```
interface Collection<E> extends Iterable<E> {
    int size();
    boolean isEmpty();
    boolean contains(Object);
    boolean add(E);
    boolean remove(Object);
    void clear();
}
```

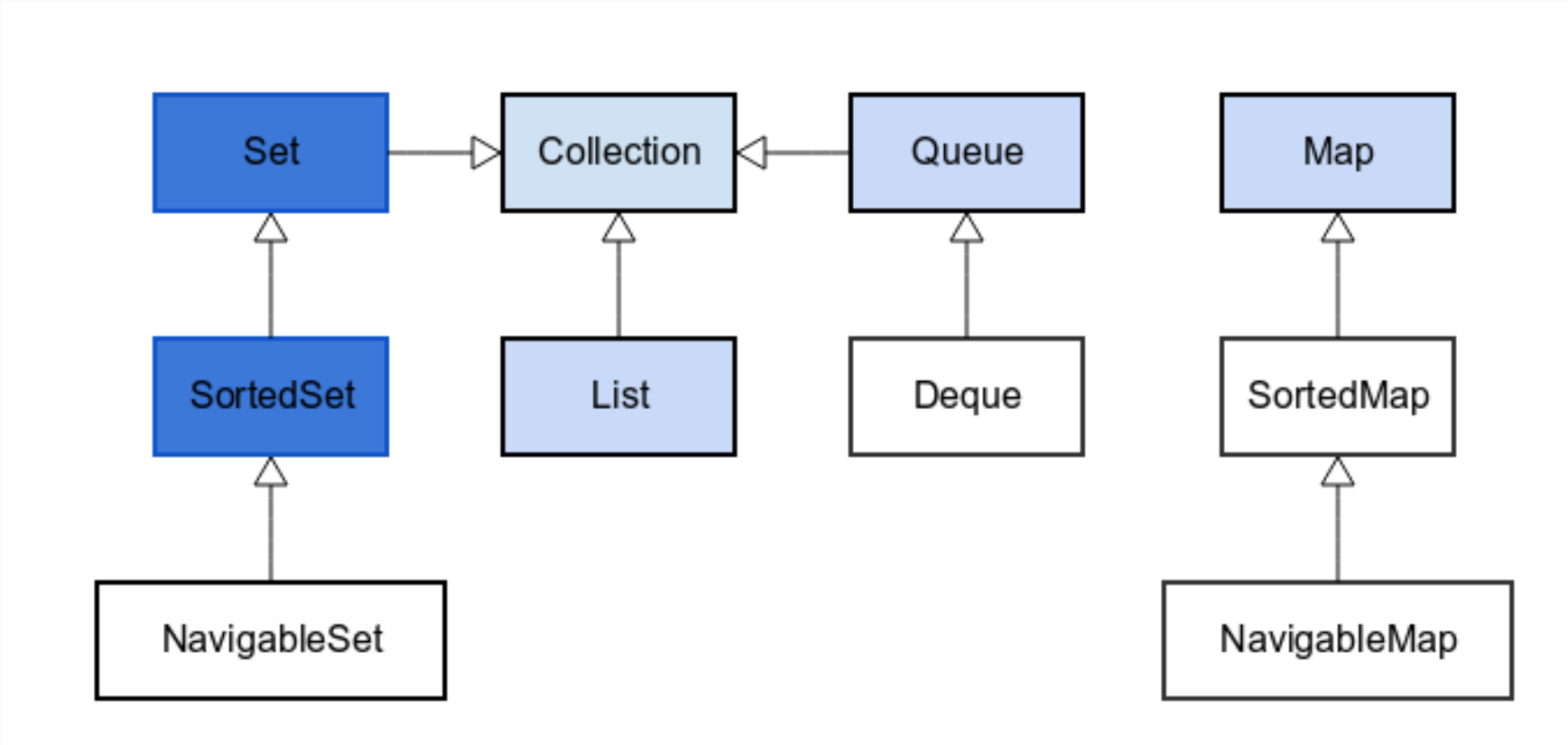
ОСНОВНЫЕ ИНТЕРФЕЙСЫ



LIST INTERFACE

```
interface List<E> extends Collection<E> {
    void add(int, E);
    void set(int, E);
    E get(int);
    int indexOf(E);
    int lastIndexOf(E)
    void remove(int)
}
```

SET INTERFACES

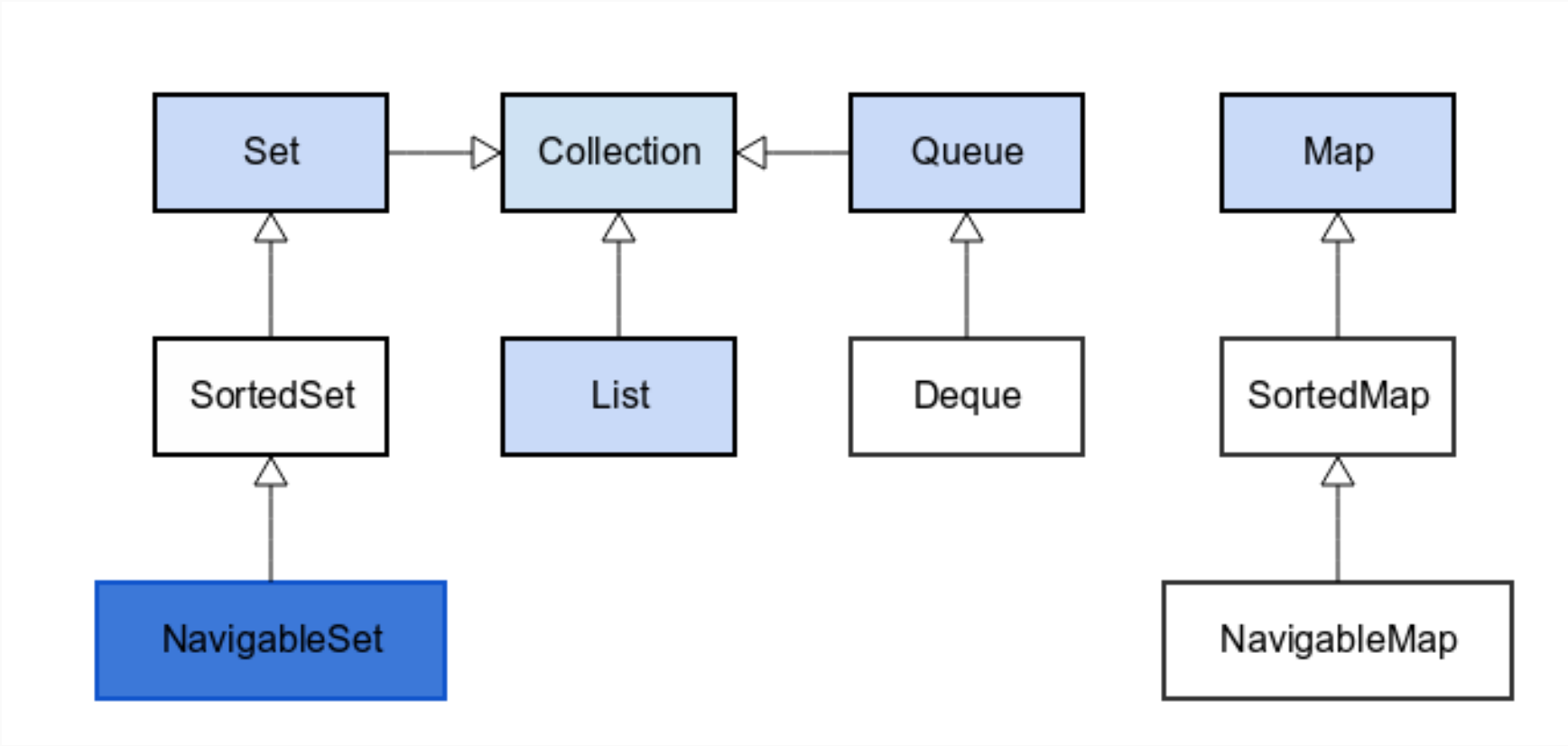


SET INTERFACES

```
interface Set<E> extends Collection<E> {
    ... // Nothing interesting here
}

interface SortedSet<E> extends Set<E> {
    SortedSet<E> subSet(E from, E to);
    SortedSet<E> headSet(E to);
    SortedSet<E> tailSet(E from);
    E first(); // lowest
    E last(); // highest
}
```

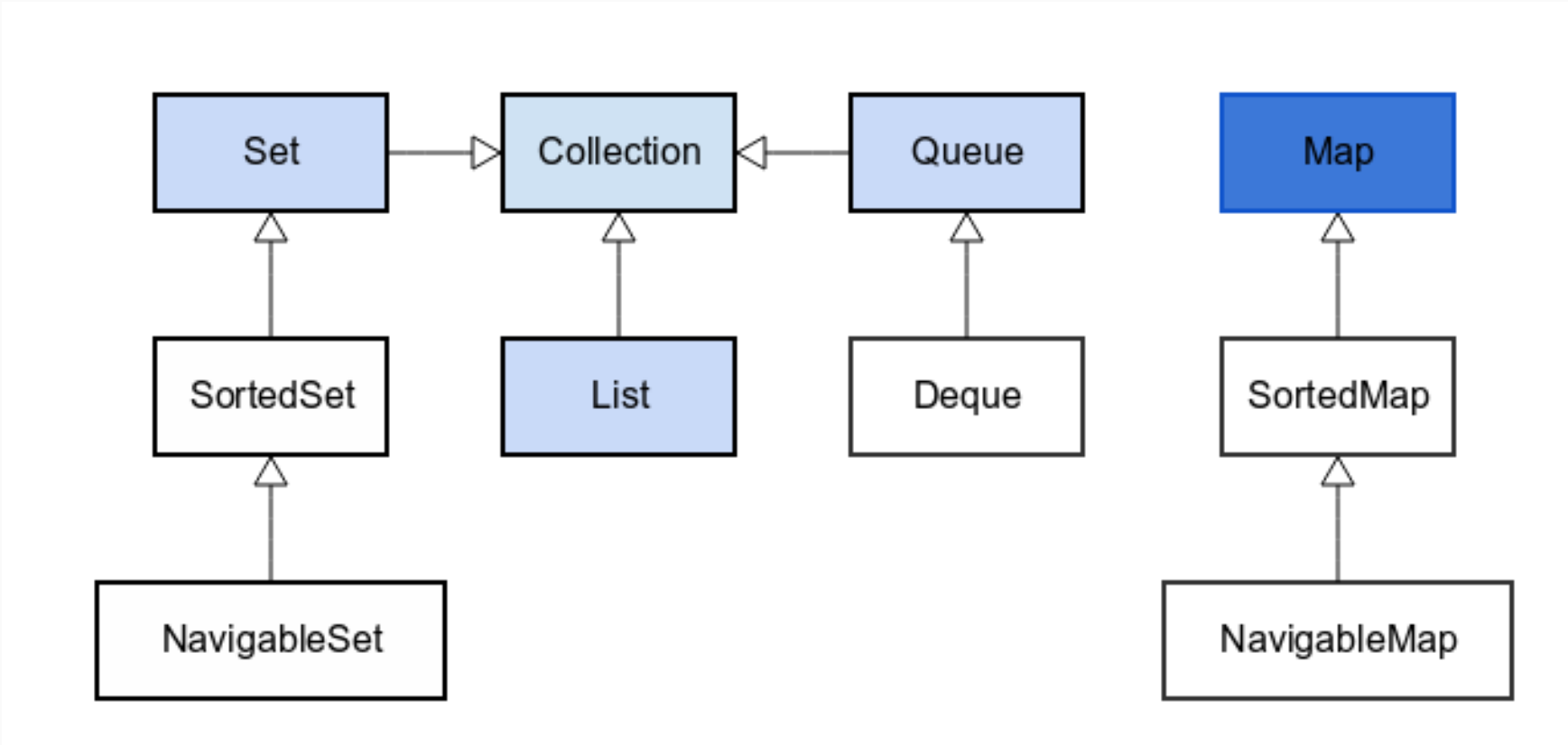
SET INTERFACES



SET INTERFACES

```
interface NavigableSet<E> extends
    SortedSet<E> {
     E lower(E e); // highest(t): t < e
     E floor(E e); // highest(t): t <= e
     E ceiling(E e); // lower(t): t >= e
     E higher(E e); // lower(t): t > e
     E pollFirst(); // lowest
     E pollLast(); // highest
     NavigableSet<E> subSet(E, boolean, E
         boolean);
}
```

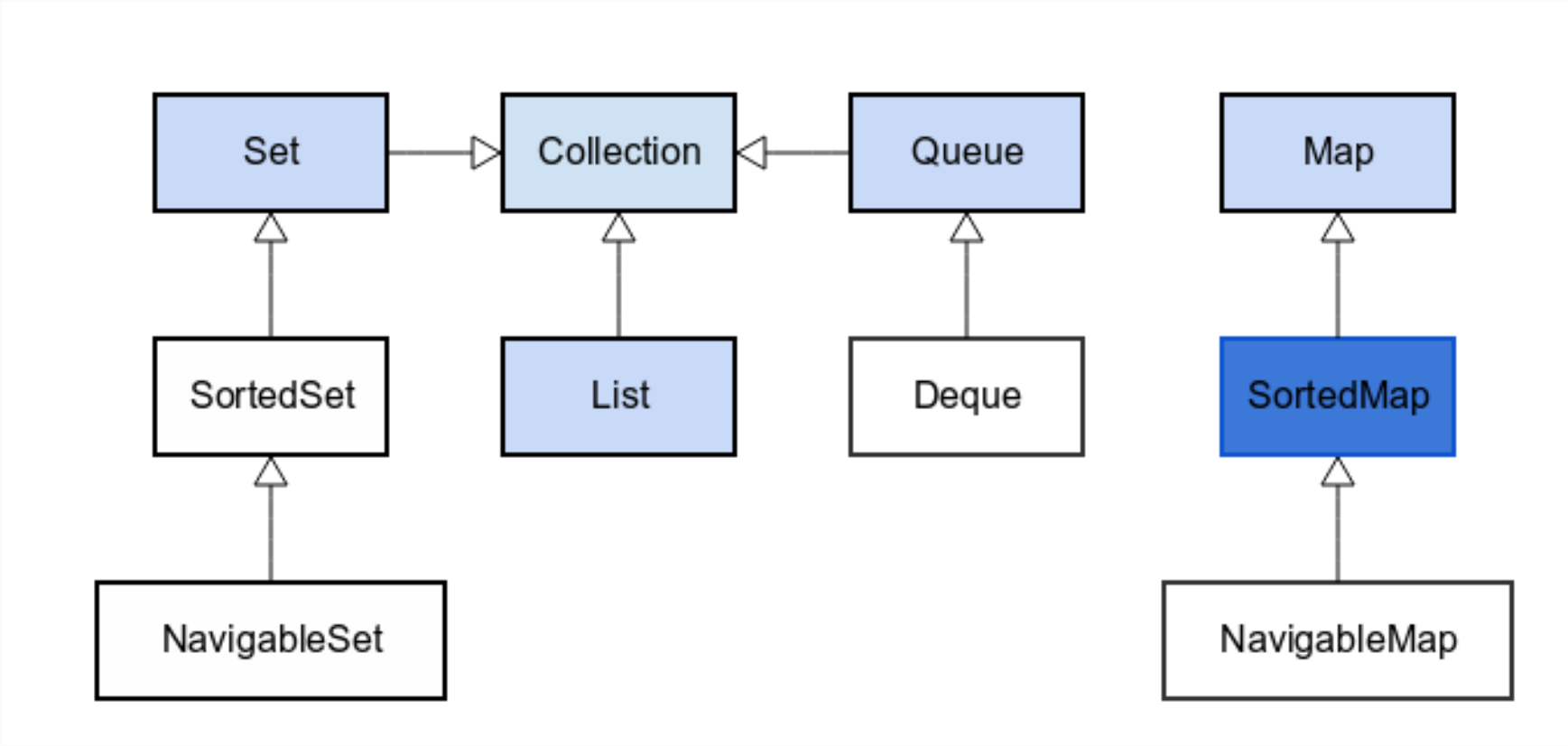
MAP INTERFACES



MAP INTERFACES

```
interface Map<K, V> {
    boolean containsKey(Object);
    boolean containsValue(Object);
    Set<K> keySet();
    Collection<V> values();
    V get(Object);
    V put(K, V);
    V remove(Object key);
    V getOrDefault(Object, V);
    V putIfAbsent(K, V);
    boolean replace(K, V, V);
    boolean replace(K, V);
}
```

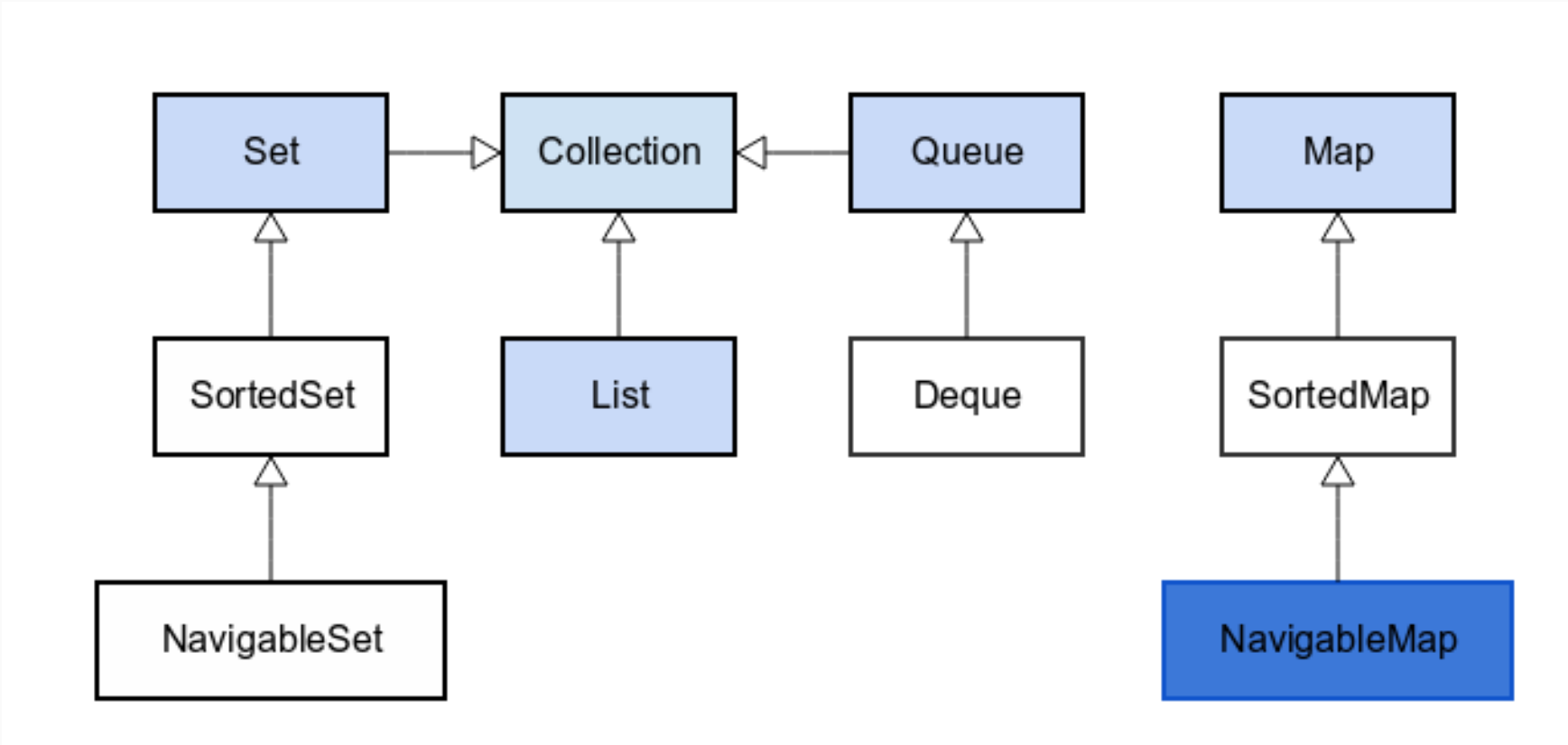
MAP INTERFACES



MAP INTERFACES

```
interface SortedMap<K, V> extends Map<K, V> {
    // same as for SortedSet
    K firstKey();
    K lastKey();
    SortedMap<K,V> subMap(K from, K to);
}
```

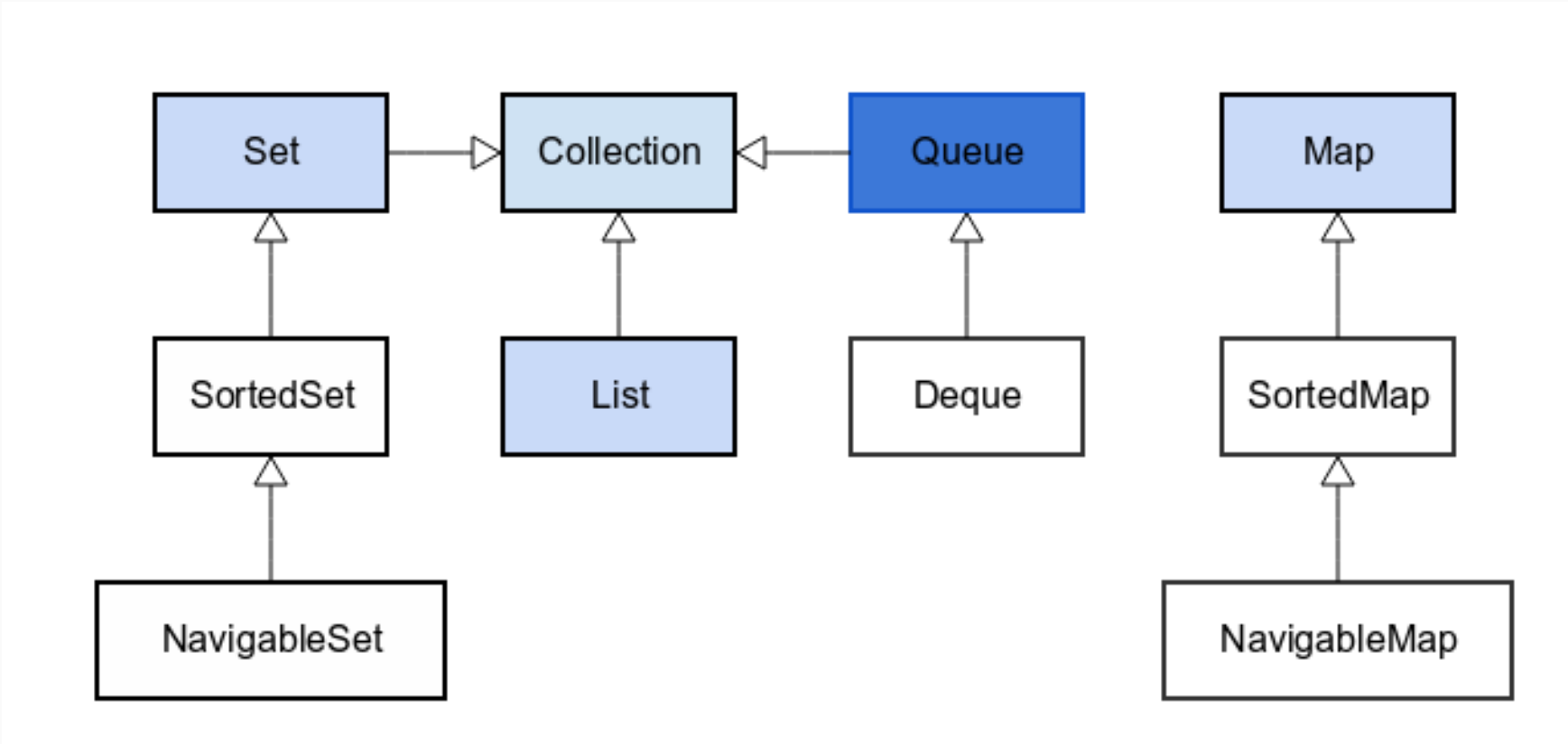
MAP INTERFACES



MAP INTERFACES

```
interface NavigableMap<K, V> extends
    SortedMap<K, V> {
     // same as for SortedSet
     K lowerKey(K key);
     Map.Entry<K,V> lowerEntry(K key);
     Map.Entry<K,V> firstEntry();
     Map.Entry<K,V> pollFirstEntry();
}
```

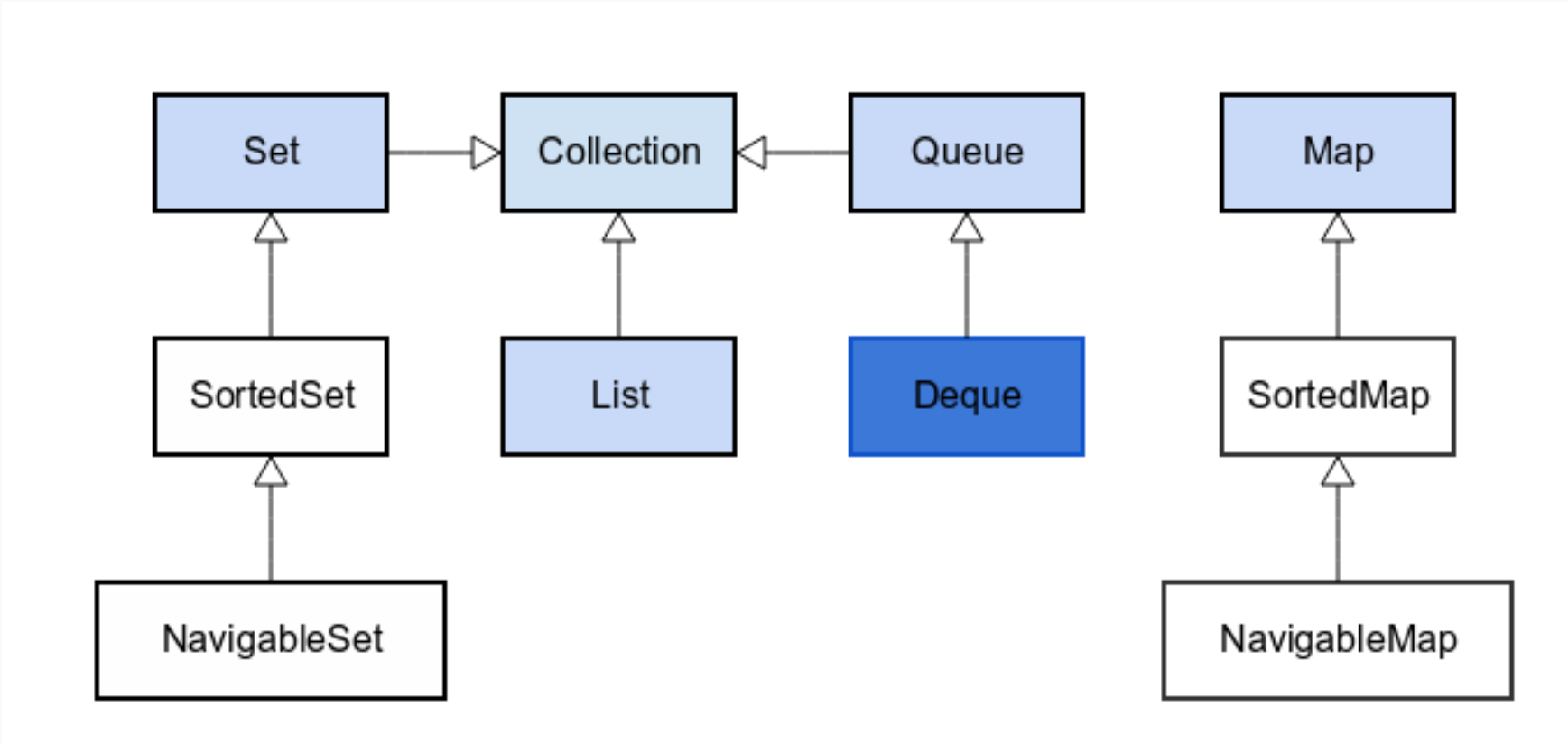
QUEUE INTERFACES



QUEUE INTERFACES

```
interface Queue<E> extends Collection<E> {
    boolean add(E);
    boolean offer(E);
    E remove();
    E poll();
    E element();
    E peek();
}
```

DEQUE INTERFACES

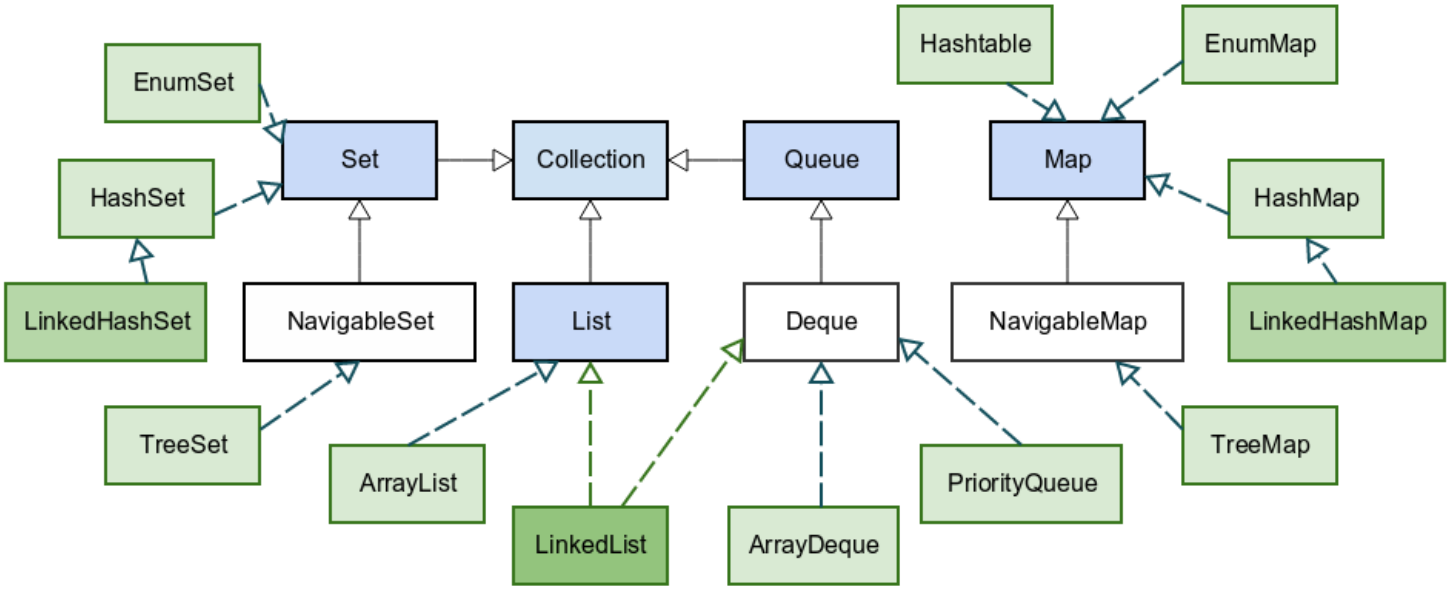


DEQUE INTERFACES

```
interface Deque<E> extends Queue<E> {
    // *Last(...) is equivalent to *(...)
    boolean addFirst(E);
    boolean addLast(E);
    boolean offerFirst(E);
    boolean offerLast(E);
    ...
    void push(E);
    E pop();
    E element();
    E peek();
}
```

КАК УСТРОЕНЫ ОСНОВНЫЕ КЛАССЫ КОЛЛЕКЦИЙ

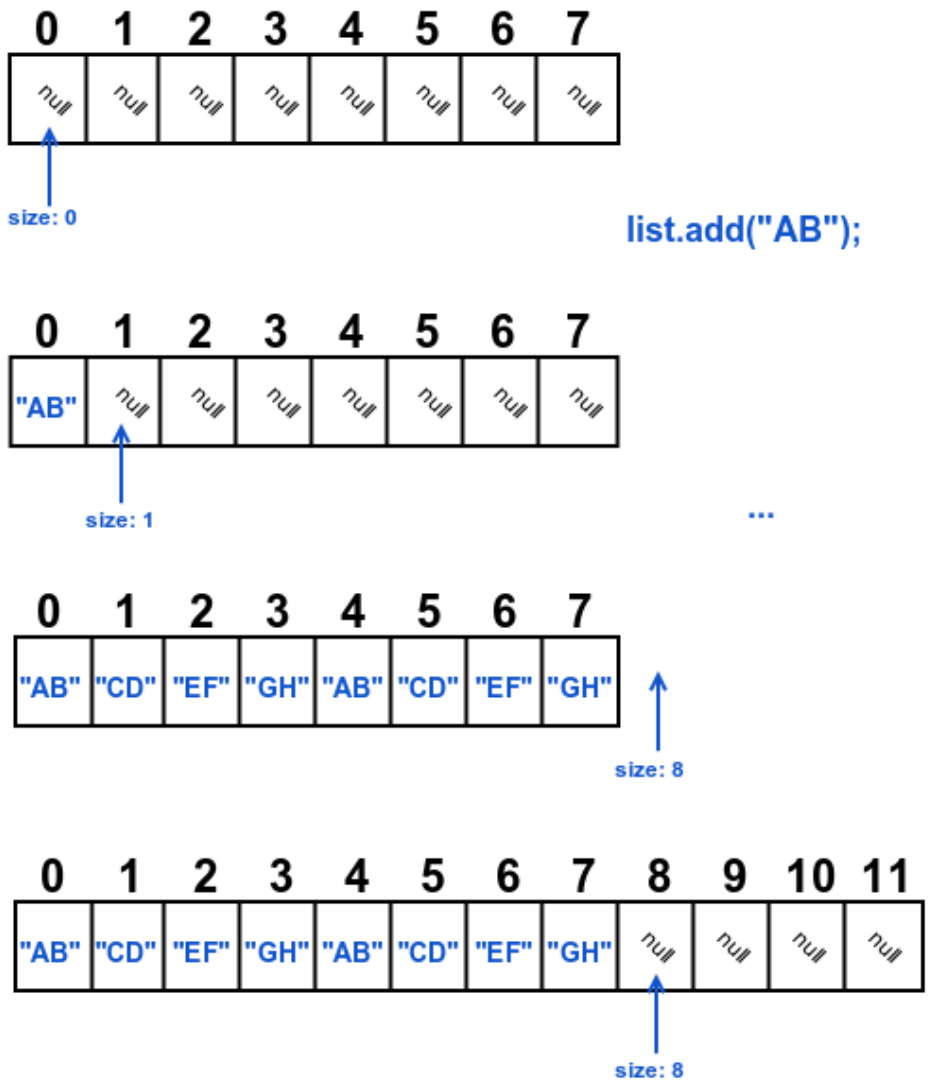
РЕАЛИЗАЦИИ


EnumSet
Set
Collection
Queue
Hashtable
EnumMap
Map
HashSet
HashMap
LinkedHashSet
NavigableSet
List
Deque
NavigableMap
LinkedHashMap
TreeSet
ArrayList
LinkedList
ArrayDeque
PriorityQueue
TreeMap

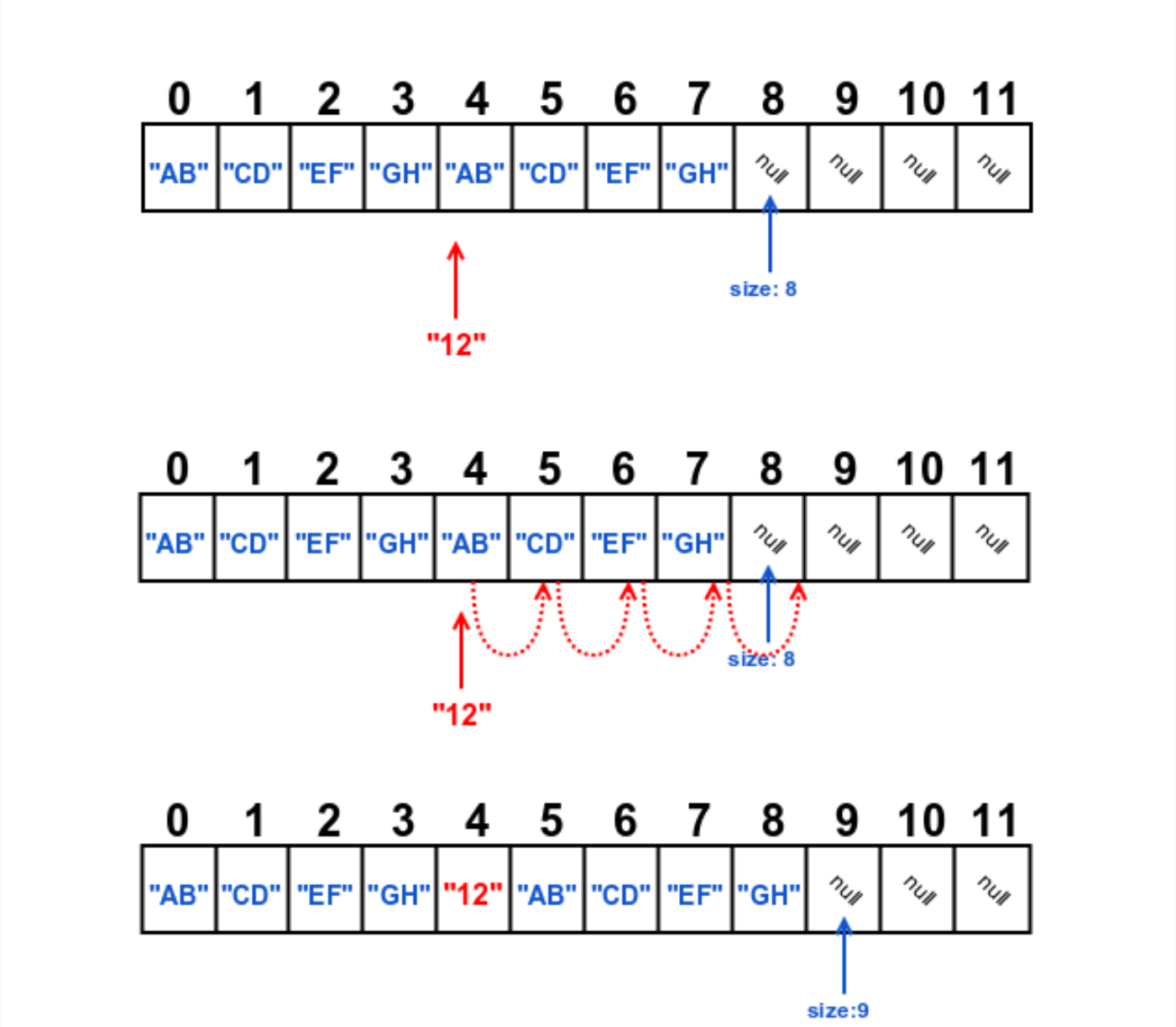
ARRAYLIST

- Реализует интерфейс **List**
- Позволяет хранить *null*
- Быстрый доступ по индексу
- Медленные операции - добавление в середину, поиск, удаление из середины
- Добавление в конец иногда очень медленное

ARRAYLIST


0 1 2 3 4 5 6 7
null
size: 0
list.add("AB");
"AB"
size: 1
...
"AB" "CD" "EF" "GH" "AB" "CD" "EF" "GH"
size: 8
0 1 2 3 4 5 6 7 8 9 10 11
size: 8

ARRAYLIST


0 1 2 3 4 5 6 7 8 9 10 11
"AB" "CD" "EF" "GH" "AB" "CD" "EF" "GH" null null null null
size: 8
"12"
0 1 2 3 4 5 6 7 8 9 10 11
"AB" "CD" "EF" "GH" "AB" "CD" "EF" "GH" null null null null
size: 8
"12"
0 1 2 3 4 5 6 7 8 9 10 11
"AB" "CD" "EF" "GH" "12" "AB" "CD" "EF" "GH" null null null
size:9

ARRAYLIST - ПОДВОДНЫЕ КАМНИ

- Если не хватает места для нового элемента, увеличивает массив. Для этого используются

  ```
  native Array.newInstance(...);
  System.arraycopy(...);
  ```

  И это очень медленно.
- Если массив вырос до больших размеров, то могут быть утечки памяти после удаления элементов. Это связано с тем, что его длина при этом не сокращается. Проблему решит

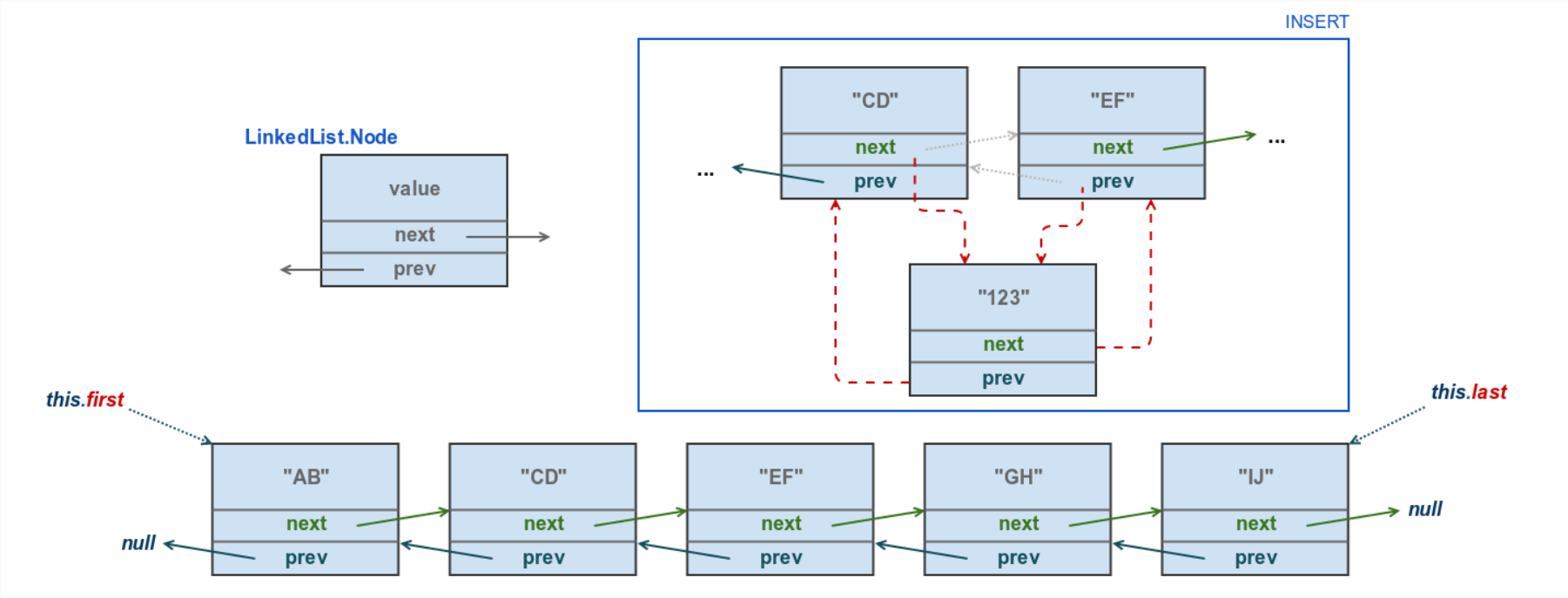
  ```
  void trimToSize();
  ```

- Доступ по индексу $O(1)$. Поиск, добавление и удаление из середины списка $O(n)$.

LINKEDLIST

- Реализует интерфейсы **List**, **Queue** и **Deque**
- Позволяет хранить *null*
- Медленный доступ по индексу и поиск
- Быстрое добавление/удаление из середины при условии, что элемент уже найден
- Добавление и удаление из конца и начала - быстрое
- Стек, очередь, двойна очередь
- Требует немало дополнительной памяти

LINKEDLIST


INSERT
LinkedList.Node
value
next
prev
"CD"
"EF"
next
prev
...
"123"
this.first
this.last
"AB"
"CD"
"EF"
"GH"
"IJ"
null
null

ARRAYLIST VS LINKEDLIST

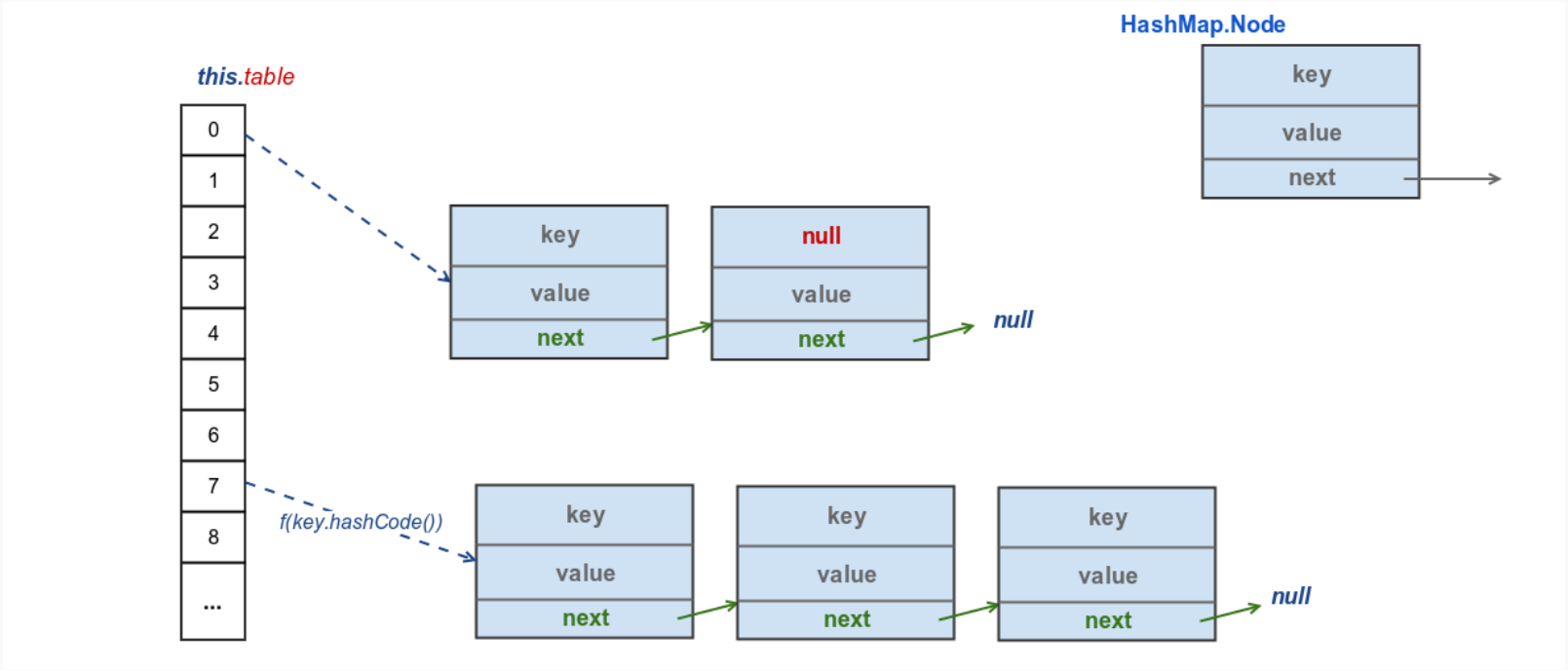
	A/R (head)	A/R (tail)	A/R (mid)	Index	Search
ArrayList	O(n)	O(1)/O(n)	O(n)	O(1)	O(n)
LinkedList	O(1)	O(1)	O(1)	O(n)	O(n)

- **LinkedList** требует больше памяти для хранения элементов.
- **ArrayList** требует ручного сокращения массива при удалении элементов.
- Оба имеют 2 метода *remove*

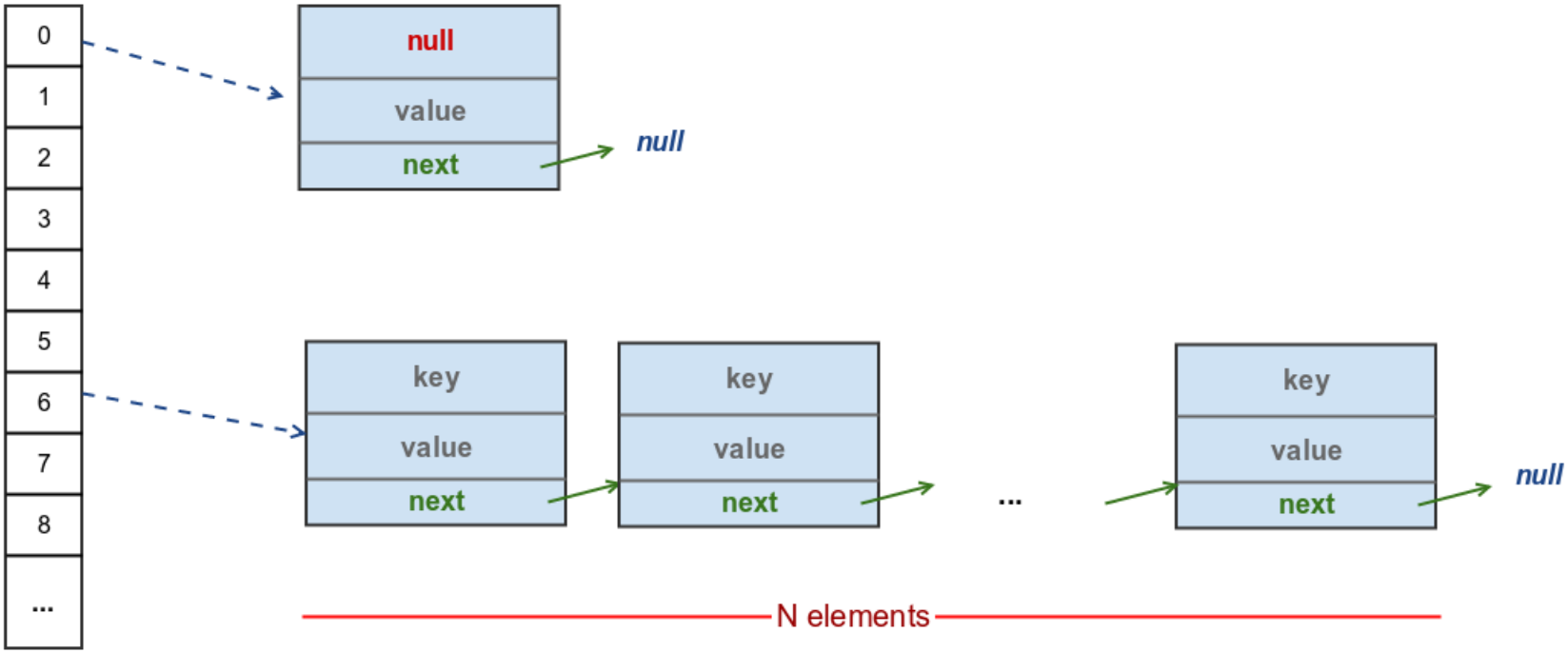
HASHMAP

- Реализует интерфейс **Map**
- Позволяет хранить *null*
- Быстрое взятие элемента по ключу (при равномерном распределении hash-функции), но в худшем случае до $O(n)$
- Быстрое добавление элемента
- Требует наличие *hashCode* и *equals* у ключа
- Наподобии *ArrayList* расширяется и перераспределяет данные при максисмальном заполнении массива (см. $threshold = (capacity * loadFactor)$)

HASHMAP


HashMap.Node
key
value
next
this.table
0
1
2
3
4
5
6
7
8
...
key
value
next
null
value
next
null
f(key.hashCode())
key
value
next
key
value
next
key
value
next
null

HASHMAP - ВЫРОЖДЕННЫЙ СЛУЧАЙ



LINKEDHASHMAP

- Реализует интерфейс **Map**
- Симбиоз hash map и linked list
- Немного уступает в скорости и использовании памяти HashMap
- Порядок сортировки ключей (при итерировании) тот же, что и при вставке

HASHMAP VS LINKEDHASHMAP VS TREEMAP

	Sorted	A/R	Get
HashMap	✗	O(1)	O(1)
TreeMap	✓	O(log N)	O(log N)
LinkedHashMap	✓	O(1)	O(1)

- **LinkedHashSet** требует больше памяти для хранения элементов.

HASHSET

- Реализует интерфейс **Set**
- Позволяет хранить *null*
- Отсутствует сортировка
- Самый быстрый - все базовые операции $O(1)$
- Сделан на основе *HashMap*

TREESET

- Реализует интерфейсы **Set**, **SortedSet** и **NavigableSet**
- Запрещены *null*
- Есть сортировка
- Медленнее, чем HashSet - все базовые операции $O(logn)$
- Сделан на основе *TreeMap* (красно-черное дерево)

LINKEDHASHSET

- Реализует интерфейс **Set**, наследник **HashSet**
- Разрешены *null*
- Есть сортировка (в порядке добавления)
- Все базовые операции $O(1)$
- Сделан на основе *HashMap* и *LinkedList*

HASHSET VS TREESET VS LINKEDHASHSET

	Sorted	A/R	Search
HashSet	✗	O(1)	O(1)
TreeSet	✓	O(log N)	O(log N)
LinkedHashSet	✓	O(1)	O(1)

- **LinkedHashSet** требует больше памяти для хранения элементов.

КОЛЛЕКЦИИ, КОТОРЫЕ НЕ ВХОДЯТ В COLLECTION FRAMEWORK

VECTOR

```
class Vector<E> implements List<E> { ... }
```

- Очень медленный
- Очень старый
- Потокобезопасный
- Аналог ArrayList

STACK

```
class Stack<E> extends Vector<E> { ... }
```

- Наследник вектора
- Такой же медленный
- Вместо него рекомендуется ArrayDeque

HASHTABLE

```
class Hashtable<K, V> implements Map<K, V> {
    ... }
```

- Потокбезопасен
- Не разрешает null
- Несколько уступает в скорости HashMap

ДОМАШНЕЕ ЗАДАНИЕ

ДОМАШНЕЕ ЗАДАНИЕ - STACK

Реализовать интерфейс IStack на основе односвязного списка.

```
public interface IStack<E> {
    void push(E element);
    E pop();
    E peek();
    int size();
    boolean isEmpty();
}
```

Добавить unit-тесты на каждый из методов.

ДОМАШНЕЕ ЗАДАНИЕ - *LULIST*

Это реальная задача с собеседования в Яндекс. Необходимо реализовать метод, который поворачивает "обратно"список без дополнительных затрат. подразумевается, что список имеет очень большое количество элементов.

```
public class LUList<E> {
    private class Entry {}
    public void add(E element);

    public void reverse(){ } // <---implement
}
```

ДОМАШНЕЕ ЗАДАНИЕ - *LULIST*

```
LUList<Integer> list = new LUList<>();
list.add(1); list.add(2);
list.add(3); list.add(4);
// prints { 4 } -> { 3 } -> { 2 } -> { 1
   } ->  null
System.out.println(list);
```

В результате вызова нового метода список должен быть "развернут"в обратную сторону.

```
list.reverse();
// ! prints { 1 } -> { 2 } -> { 3 } -> {
   4 } ->  null
System.out.println(list);
```

ВОПРОСЫ С СОБЕСЕДОВАНИЯ

ВОПРОСЫ С СОБЕСЕДОВАНИЯ - LIST

- В чем отличие ArrayList от LinkedList?
- В чем отличие ArrayList от Vector?
- Что быстрее - ArrayList от LinkedList?
- Что лучше использовать для 1 миллиона элементов- ArrayList от LinkedList?
- Предложите эффективный алгоритм для удаления нескольких элементов в середине ArrayList.
- Как преобразовать одной строчкой HashSet в ArrayList?
- В чем преимущества массива перед ArrayList?

ВОПРОСЫ С СОБЕСЕДОВАНИЯ - MAPS

- В чем отличие HashMap от Hashtable?
- Почему Map не коллекция?
- В чем проявляется сортированность SortedMap?
- Что будет, если в Map положить два значения с одним и тем же ключом?
- Какие ограничения для класса-ключа есть при работу с HashMap?

ВОПРОСЫ С СОБЕСЕДОВАНИЯ - MAPS

- Каким свойством должна обладать функция hashCode() ключа для оптимальной работы с HashMap?
- Почему нельзя использовать byte[] в качестве ключа в HashMap?
- В каком случае может быть потерян элемент HashMap?
- Как производится увеличение размера HashMap?
- Какое начальное количество корзин в HashMap?
- Что будет, если добавлять в TreeSet элементы по возрастанию?

ВОПРОСЫ С СОБЕСЕДОВАНИЯ - РАЗНОЕ

- Как перебрать значения Map, учитывая, что это не Iterable?
- Как связаны Iterable, Iterator и for-each?
- Отличия ListIterator от Iterator.