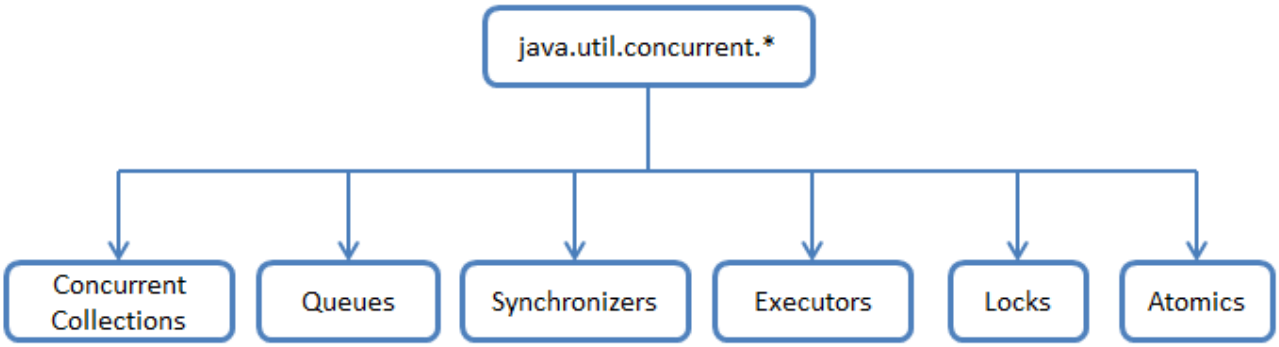
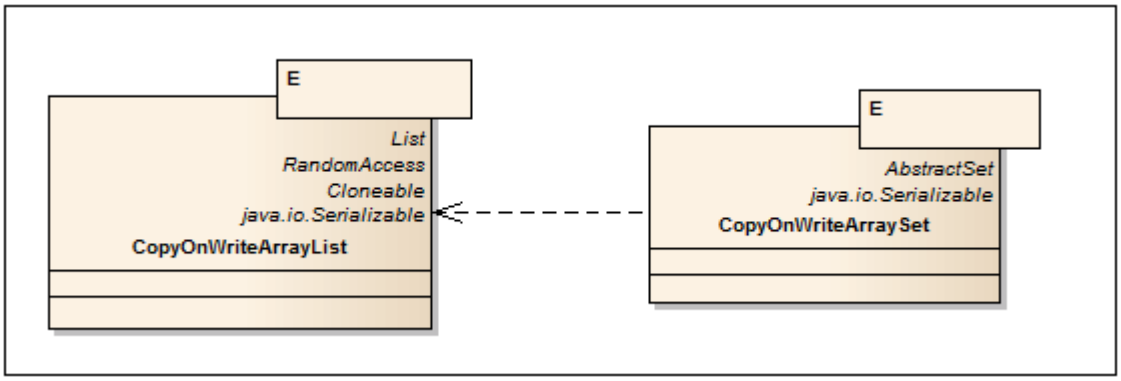



**Concurrent Collections** — набор коллекций, более эффективно работающие в многопоточной среде нежели стандартные универсальные коллекции из java.util пакета. Вместо базового враппера Collections.synchronizedList с блокированием доступа ко всей коллекции используются блокировки по сегментам данных или же оптимизируется работа для параллельного чтения данных по wait-free алгоритмам.

**CopyOnWrite коллекции**




Название говорит само за себя. Все операции по изменению коллекции (add, set, remove) приводят к созданию новой копии внутреннего массива. Тем самым гарантируется, что при проходе итератором по коллекции не кинется ConcurrentModificationException. Следует помнить, что при копировании массива копируются только референсы (ссылки) на объекты (shallow copy), т.ч. доступ к полям элементов не thread-safe. CopyOnWrite коллекции удобно использовать, когда write операции довольно редки, например при реализации механизма подписки listeners и прохода по ним.

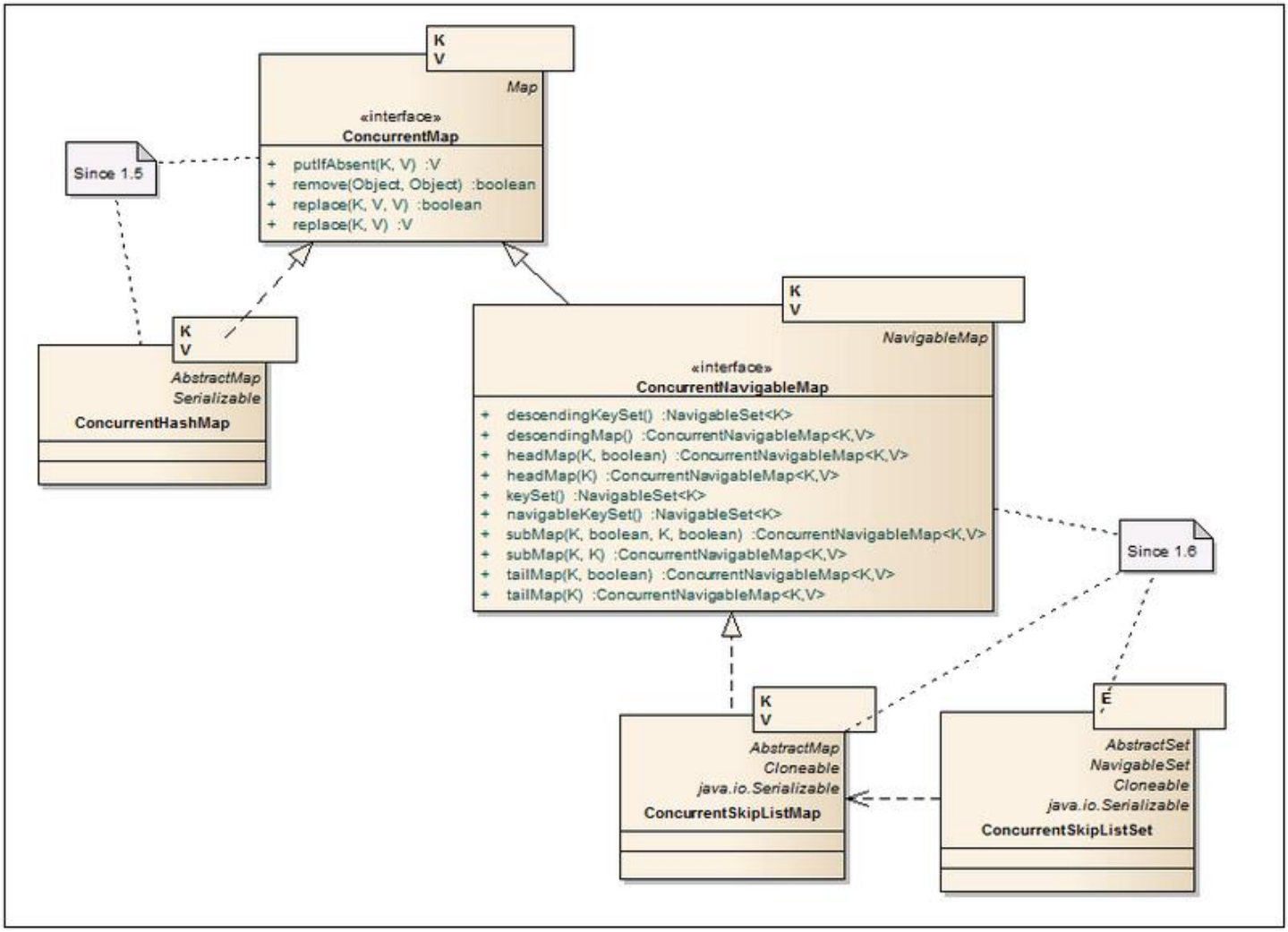
**CopyOnWriteArrayList<E>** Class 1.5 — Потокобезопасный аналог ArrayList, реализованный с CopyOnWrite алгоритмом.

▾ **Дополнительные методы и конструктор**

| | |
|---|---|
| CopyOnWriteArrayList(E[] toCopyIn) | Конструктор, принимающий на вход массив. |
| int indexOf(E e, int index) | Возвращает индекс первого найденного элемента, начиная поиск с заданного индекса. |
| int lastIndexOf(E e, int index) | Возвращает индекс первого найденного элемента при обратном поиске, начиная с заданного индекса. |
| boolean addIfAbsent(E e) | Добавить элемент, если его нет в коллекции. Для сравнения элементов используется метод equals. |
| int addAllAbsent(Collection<? extends E> c) | Добавить элементы, если они отсутствуют в коллекции. Возвращает количество добавленных элементов. |

**CopyOnWriteArraySet<E>** Class 1.5 — Имплементация интерфейса Set, использующая за основу CopyOnWriteArrayList. В отличии от CopyOnWriteArrayList, дополнительных методов нет.

**Scalable maps**




Улучшенные реализации HashMap, TreeMap с лучшей поддержкой многопоточности и масштабируемости.

**ConcurrentMap<K, V>** Interface 1.5 — Интерфейс, расширяющий Map несколькими дополнительными атомарными операциями.

**Дополнительные методы**

| | |
|---|---|
| V putIfAbsent(K key, V value) | Добавляет новую пару key-value только в том случае, если ключа нет в коллекции. Возвращает предыдущее значение для заданного ключа. |
| boolean remove(Object key, Object value) | Удаляет key-value пару только если заданному ключу соответствует заданное значение в Map. Возвращает true, если элемент был успешно удален. |
| boolean replace(K key, V oldValue, V newValue) | Заменяет старое значение на новое по ключу только если старое значение соответствует заданному значению в Map. Возвращает true, если значение было заменено на новое. |
| V replace(K key, V value) | Заменяет старое значение на новое по ключу только если ключ ассоциирован с любым значением. Возвращает предыдущее значение для заданного ключа. |

**ConcurrentHashMap<K, V>** Class 1.5 — В отличие от Hashtable и блоков synhronized на HashMap, данные представлены в виде сегментов, разбитых по hash'ам ключей. В результате доступ к данным лочится по сегментам, а не по одному объекту. В дополнение итераторы представляют данные на определенный срез времени и не кидают ConcurrentModificationException. Более детально ConcurrentHashMap описан в хабратопике тут.

**Дополнительный конструктор**

| | |
|---|---|
| ConcurrentHashMap(int initialCapacity, float loadFactor, int concurrencyLevel) | 3-й параметр конструктора — ожидаемое количество одновременно пишущих потоков. Значение по умолчанию 16. Влияет на размер коллекции в памяти и производительность. |

**ConcurrentNavigableMap<K,V>** Interface 1.6 — Расширяет интерфейс NavigableMap и вынуждает использовать ConcurrentNavigableMap объекты в качестве возвращаемых значений. Все итераторы декларируются как безопасные к использованию и не кидают ConcurrentModificationException.

**ConcurrentSkipListMap<K, V>** Class 1.6 — Является аналогом TreeMap с поддержкой многопоточности. Данные также сортируются по ключу и гарантируется усредненная производительность log(N) для containsKey, get, put, remove и других похожих операций. Алгоритм работы SkipList описан на Wiki и хабре.

**ConcurrentSkipListSet<E>** Class 1.6 — Имплементация Set интерфейса, выполненная на основе ConcurrentSkipListMap. Она представляет собой связный список, где вставка и удаление элементов происходит достаточно быстро. Такая структура данных также хорошо подходит для неблокирующего доступа несколькими потоками, ведь, например, для вставки достаточно заблокировать изменение двух соседних элементов в связном списке.