**Отличие synchronized коллекций от concurrency в том, что первые используют блокировку всей коллекции для доступа, тогда как concurrency использует умное блокирование, например, ConcurrentHashMap делит всю коллекцию на блоки и блокирует только блоки, а CopyOnWriteArrayList разрешает чтение записей без синхронизации, а когда происходит запись копируется весь массив и меняется с предыдущим.**

**Arrays – содержит методы для сортировки и поиска элементов в массиве. (static methods for:**

**Лексикографического сравнения(compare()), бинарного поиска(binarySearch()),**

**copyOf(original, length) – усекает или заполняет 0, copyOfRange(original, start, stop) – ошибка если индексы за пределами),**

**deepEquals(Object[] a, Object[] b), deepHashCode(Object[] a) - ,**

**deepToString(Object[] a),**

**equals() – сравнение one by one,**

**fill(original, from, to, value) – Заполняет массив, hashCode(),**

**mismatch(a, [from, to], b, [from, to]) – находит первый различный элемент, либо -1,**

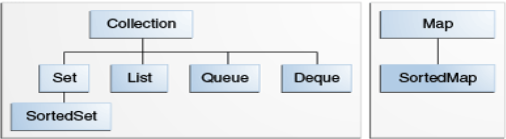
**sort()(для объектов вторым аргументом может идти компаратор) – quicksort**

**stream()**

**Arrays.asList(int[])**

**toString()**

Comparator<Object > реализует один метод compare(object a, object b).



**Интерфейсы:**

**List<> - упорядоченная последовательность,**

**Методы:**

* **get(index),**
* **add([index], element),**
* **Element remove(index),**
* **Boolean remove(Object),**
* **indexOf(Object),**
* **List.of(Elements) – возвращает unmodifiable Лист – любые операции вставки, удаления или реплейса будут возвращать ошибку,**
* **set(index, element),**
* **sort(Comparator),**
* **contains(Element)**

**Queue<> -**

**Методы:**

* **add(e) – throw [если не хватает емкости]**
* **remove() – throw [return and remove]**
* **element() – throw [return but not remove]**
* **offer(e) – false [если не хватает емкости]**
* **pool() – null [return and remove]**
* **peek() – null [return but not remove]**

**Dequeue<> -**

**Методы Queue + pop() – removeFirst()**

**Set<> - разрешен null**

* **clear() –**
* **contains() –**
* **add() –**
* **addAll(Collection<>) –**
* **isEmpty() -**
* **remove(Element) -**
* **size() -**

**Map<> -**

* **containsKey() –**
* **containsValue() –**
* **entrySet() -**
* **get(Key) –**
* **keySet() –**
* **put() –**
* **remove() –**
* **replace(key, value) –**
* **size() –**
* **values() –**
* **clear() -**

**ArrayList – коллекция строящаяся на массиве.**

**Add: amortized constant time. Значит, что для добавления n элементов O(n)**

**constant factor меньше чем у LinkedList**

**growth politics – не специфицирована, но такая, чтобы добавление n элементов происходило за O(n)**

**Capacity – размер массива для хранения элементов, можно увеличивать перед добавлением**

**Итераторы от ArrayList fail-fast, т.е вернут ошибку если List будет модифицирован как либо, кроме iterator add and remove.**

**LinkedList - double-linked list, добавлены операции peekFirst e. t. c**

**Операции добавления O (1).**

**Поиска O(n).**

**Удаления O(1) + O(n) на поиск.**

**HashMap – load factor – 0.75**

**Bucket – Отдельный слот в hashmap.**

**This implementation provides constant-time performance for the basic operations (get and put), полагая, что хэш-функция распределяет элементы равномерно**

**Итерация по коллекции требует времени, пропорционального "capacity" экземпляра HashMap (количеству сегментов) плюс его размеру (количеству сопоставлений ключ-значение). Таким образом, очень важно не устанавливать слишком высокую начальную емкость (или слишком низкий коэффициент загрузки), если важна производительность итерации.**

**capacity - количество сегментов в hash table.**

**initial capacity – емкость на момент создания.**

**load factor – показатель того насколько будет заполнена коллекция, прежде чем ее capacity будет увеличена.**

**После перестроения коллекция увеличивается примерно вдвое.**

**Как правило, начальный Load factor равен 0.75 – компромисс между затратами временными и по памяти. Более высокое значение уменьшает накладные расходы, но увеличивает время поиска. Ожидаемое количество записей и load factor должны учитываться при установке начальной емкости чтобы уменьшить количество операций rehash – пересчитывание хэша каждого элемента и сохранение в новом массиве даблсайз(при перестроении коллекции).**

**Если initial capacity > maximum number of entries / load factor, То операции rehash не будут выполняться**

**Порядок вставки элемента:**

1. **K is converted into a small integer (called its hash code) using a hash function.**
2. **The hash code is used to find an index (hashCode % arrSize) and the entire linked list at that index(Separate chaining) is first searched for the presence of the K already.**
3. **If found, it’s value is updated and if not, the K-V pair is stored as a new node in the list.**

**Iterator is fail-fast.**

**Разрешение коллизий:**

1. **Separate Chaining  - цепочка LinkedList**

**Плюсы**

**1) Простота реализации.**

**2) Хэш-таблица никогда не заполняется, мы всегда можем добавить больше элементов в цепочку.**

**3) Менее чувствительны к хеш-функции или load factor.**

**4) Он в основном используется, когда неизвестно, сколько и как часто ключи могут быть вставлены или удалены.**

**Минусы**

**1) Производительность кэша цепочки не очень хороша, так как ключи хранятся с использованием связанного списка. Открытая адресация обеспечивает лучшую производительность кэша, поскольку все данные хранятся в одной таблице.**

**2) Потеря пространства (Некоторые части хэш-таблицы никогда не используются)**

**3) Если цепочка становится длинной, то время поиска может стать O(n) в худшем случае.**

**4) Использует дополнительное пространство для ссылок.**

1. **Open Addressing - Все элементы хранятся в таблице**

**Линейное зондирование – установка в следующий пустой слот**

**Одной из проблем линейного зондирования является Первичная кластеризация, многие последовательные элементы образуют группы, и для поиска свободного слота или поиска элемента требуется время.**

**Двойное хэширование.**

**HashSet**

**Не гарантирует порядок итерации, допускает хранение null.**

**(add, remove, contains и size) – constant time.**

**Итерация по этому набору требует времени, пропорционального сумме размера экземпляра HashSet (количество элементов) плюс "емкость" резервного экземпляра HashMap (количество сегментов). Таким образом, очень важно не устанавливать слишком высокую начальную емкость (или слишком низкий коэффициент загрузки), если важна производительность итерации.**

**Hashtable<K, V> - любой ненулевой (в HashMap можно использовать нулевые элементы) элемент может использоваться как ключ. Если происходит hash коллизия, то сингл buckets хранит несколько элементов, поиск по которым выполняется последовательно.Методы Hashtable synchronized – поэтому производительность падает, но зато можно использовать в multithreading. При это в случае итерирования по коллекции требуется ручная синхронизация, т.е итератор -fail-fast.**

**LinkedHasMap<K, V> – Реализация Hashtable и LinkedList с предсказуемым порядком адресации.**

**Отличается от HasMap тем, что поддерживает связный список, проходящий через все записи.**

**Этот список определяет порядок итерации. Обычно, это порядок, в котором ключи были вставлены в Map.Повторная вставка ключа не меняет порядок итерации. Эта реализация избавляет от хаотичного упорядочивания, которое предоставляется HashMap и Hashtable без увеличения затрат связанных с TreeMap.**

**Можно создать копию Map, зафиксировав порядок.**

**Порядок итерации от самого последнего вставленного до самого первого.**

**Постоянное время для основных операций. Производительность немного ниже, чем у HashMap из-за дополнительных затрат на поддержание списка. Исключение: итерация по коллекции требует время пропорционального размеру Map независимо от capacity, Итерация по HashMap требует время пропорционального ее capacity.**

**initial capacity and load factor – определяются так же как у HashMap. LinkedHashMap(Map<>) -конструктор для фиксирования порядка.**

**TreeMap<K, V> - Red-Black tree имплементация NavigableMap. Map сортируется в соответствии с натруальным порядком либо по Comapator<E>.**

**Предоставляет гарантированное log(n) время для операций containsKey, get, put, remove. Имплементация не sуnchronized, для synchronized имплементации использовать** [**Collections.synchronizedSortedMap**](https://docs.oracle.com/javase/8/docs/api/java/util/Collections.html#synchronizedSortedMap-java.util.SortedMap-)**.**

**Iterator is fail-fast. Все Map.Entry пары представляют из себя snapshot map времени, когда он был сделан, и не поддерживают Entry.setValue метод.**

**LinkedHashSet<> - Итерация пропорциональная количеству элементов, тогда как итерация по HashSet пропорциональна capacity. Сохраняет порядок вставки. Обеспечивает константное время для операций add, contains, remove (полагая, что хэш функция распределяет элементы равномерно по сету). Iterator is fail-fast: if the set is modified at any time after the iterator is created, in any way except through the iterator's own remove method, the iterator will throw a** [**ConcurrentModificationException**](https://docs.oracle.com/javase/8/docs/api/java/util/ConcurrentModificationException.html).

**LinkedList<E> - double-linked list, может содержать null. Implementation is not synchronized, for synchronized Collections.synchronizedList(new LinkedList(...)). Все операции производятся в соответствии с double-linked list.**

SortedMap – Map которая обеспечивает порядок в своих ключах в соответсвии с ествественным порядком либо через Comparator.

Порядок отображается при entryKey, keySet, values.

Все ключи должны быть сравнимы друг с другом

Все реализации должны содержать 4 коснтруктора с параметрами  
Comparator

Map

SortedMap

Void

**I Comparator<T> - compare(T o1, T o2)**

**Math – все методы статичные, Math.min(a, b), Math.max(a, b), e.t.c**

**Java.util.Collections –**

**Collections.revers(List ) – изменяет порядок элементов в List**

**Collections.min/max – возвращает минимальный максимальный элемент.**

**Collections.singleton[List, Map](T o) – возвращает immutable Set[List, Map], содержащий только один элемент o**

**Collections.sort(List, [Comparator]) – сортирует коллекцию**

**Collections.binarySearch (List, Key) – бинарный поиск**

**Collections.synchronizedCollection[List e.t.c]**

**Collections.addAll(Collection, T … element)**

**Objects**

**compare(a,b,Comaprator) – 0 если идентичны comparator.compare иначе**

|  |
| --- |
| [**deepEquals**](https://docs.oracle.com/javase/8/docs/api/java/util/Objects.html#deepEquals-java.lang.Object-java.lang.Object-)**(**[**Object**](https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html)**a,** [**Object**](https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html)**b)**  **Returns true if the arguments are deeply equal to each other and false otherwise.** |
| [**equals**](https://docs.oracle.com/javase/8/docs/api/java/util/Objects.html#equals-java.lang.Object-java.lang.Object-)  **Returns true if the arguments are equal to each other and false otherwise.** |
| [**hash**](https://docs.oracle.com/javase/8/docs/api/java/util/Objects.html#hash-java.lang.Object...-)**(**[**Object**](https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html)**... values)**  **Generates a hash code for a sequence of input values.** |
| [**hashCode**](https://docs.oracle.com/javase/8/docs/api/java/util/Objects.html#hashCode-java.lang.Object-)**(**[**Object**](https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html)**o)**  **Returns the hash code of a non-null argument and 0 for a null argument.** |
| [**isNull**](https://docs.oracle.com/javase/8/docs/api/java/util/Objects.html#isNull-java.lang.Object-)**(**[**Object**](https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html)**obj)**  **Returns true if the provided reference is null otherwise returns false.** |
| [**nonNull**](https://docs.oracle.com/javase/8/docs/api/java/util/Objects.html#nonNull-java.lang.Object-)**(**[**Object**](https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html)**obj)**  **Returns true if the provided reference is non-null otherwise returns false.** |
| [**requireNonNull**](https://docs.oracle.com/javase/8/docs/api/java/util/Objects.html#requireNonNull-T-)**(T obj)**  **Checks that the specified object reference is not null.** |
| [**requireNonNull**](https://docs.oracle.com/javase/8/docs/api/java/util/Objects.html#requireNonNull-T-java.lang.String-)**(T obj,** [**String**](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html)**message)**  **Checks that the specified object reference is not null and throws a customized** [**NullPointerException**](https://docs.oracle.com/javase/8/docs/api/java/lang/NullPointerException.html) **if it is.** |
| [**requireNonNull**](https://docs.oracle.com/javase/8/docs/api/java/util/Objects.html#requireNonNull-T-java.util.function.Supplier-)**(T obj,** [**Supplier**](https://docs.oracle.com/javase/8/docs/api/java/util/function/Supplier.html)**<**[**String**](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html)**> messageSupplier)**  **Checks that the specified object reference is not null and throws a customized** [**NullPointerException**](https://docs.oracle.com/javase/8/docs/api/java/lang/NullPointerException.html) **if it is.** |
| [**toString**](https://docs.oracle.com/javase/8/docs/api/java/util/Objects.html#toString-java.lang.Object-)**(**[**Object**](https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html)**o)**  **Returns the result of calling toString for a non-null argument and "null" for a null argument.** |
| [**toString**](https://docs.oracle.com/javase/8/docs/api/java/util/Objects.html#toString-java.lang.Object-java.lang.String-)**(**[**Object**](https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html)**o,** [**String**](https://docs.oracle.com/javase/8/docs/api/java/lang/String.html)**nullDefault)**  **Returns the result of calling toString on the first argument if the first argument is not null and returns the second argument otherwise.** |

**Java.nio**

**Buffers – контейнер для данных**

**Charsets – decoders and encoders переводящие символы из байтов в Unicode символы и charset’ы**

**Channels – различных типов, представляющие соединение с сущностями, способными выполнять операции ввода/вывода**

**Selector’ы и selection key – вместе с каналами осуществляют неблокирующий ввод/вывод**

**Buffer – контейнер для фиксированного объема данных примитивного типа.**

**Содержит**

**position – индекс элемента, который должен быть прочитан или записан**

**limit – индекс элемента, который не должен быть записан.**

**mark – индекс элемента, на который будет сброшена Position при reset.**

**ByteBuffer, CharBuffer e.t.c Get/put**

**MappedByteBuffer – маппирует регион файла в буфер.**

**Java.lang.Thread –**

**Потоки выполняются пока:**

* **не был вызван метод exit из класса Runtime.**
* **Все потоки не демоны, умерли, вернулись из run, или выбросили исключение.**

**Daemon thread – низкоприоритетный поток, который поставляет некоторый сервис для User thread’ам, например: garbage collector – daemon thread.**

**Join() – ждет пока поток умрет**

**Sleep() – усыпляет поток**

**Start() –**

**Yield() – освобождает процессор. Планировщик запускает следующий поток, если таких нет, то поток который предпоследним сделал yield.**

**Interrupt() – прерывает поток.**

**Java.util.Concurrency –**

**Executor**

**Java.io -**

**Metaspace – (stored in native heap) Область в памяти предназначенная для хранения загруженных классов, интернированных строк и других метаданных. Сборка мусора вызывается, когда достигнуто определенное значение, заданное при запуске jvm.**