	Основная функциональность	Примеры типичного использования
Set	**:** - неупорядоченный набор повторяющихся элементов. сширяет интерфейс Collection. ли производится попытка бавить в набор элемент, который е в нем содержится, она будет игнорирована.	**HashSet** хранит элементы в хеш-таблице, из-за чего имеет наиболее высокую производительность, но не гарантирует порядок элементов; **TreeSet** хранит элементы в отсортированном порядке, из-за чего работает существенно медленнее, чем HashSet; **LinkedHashSet** отличается от HashSet тем, что хранит элементы в порядке их вставки в коллекцию. Эта коллекция лишь немного медленнее HashSet.
List	**t** - служит для работы с орядоченными коллекциями. К ждому элементы такой плекции можно обратиться по дексу. Расширяет интерфейс llection.	**ArrayList** - наиболее широко используемая реализация List. ArrayList обладает наибольшей производительностью в плане доступа к случайному элементу в массиве. **LinkedList** - еще одна реализация интерфейса List. В отличие от ArrayList, LinkedList обладает большей скоростью вставки элемента в произвольное место в списке, однако доступ к элементу прямо пропорциональный его позиции относительно начала последовательности.
Queue	**eue** - очередь. Представляет бой структуру данных, ботающую по принципу FIFO. днако, существуют и реализации принципе LIFO и унаправленные очереди	**PriorityQueue** – (FIFO) прямая реализация интерфейса. Поддерживает возможность управления порядком элементов с помощью компаратора. **LinkedQueue** - (FIFO). Элементы очереди имеют ссылки друг на друга. **ArrayDeque** - (LIFO) Двунаправленная очередь, реализующая интерфейс Deque, который расширяет интерфейс Queue.
Map	**ıp** - предназначен для работы с плекциями-словарями, в которых держатся ключи и ответствующие им ачения(каждому ключу ответствует только одно ачением). Словарь может держать произвольное число ементов.	**HashMap** хранит ключи в хеш-таблице, из-за чего имеет наиболее высокую производительность, но не гарантирует порядок элементов. Может содержать как null-ключи, так и null-значения; **TreeMap** хранит ключи в отсортированном порядке, из-за чего работает существенно медленнее, чем HashMap. Не может содержать null-ключи, но может содержать null-значения. Сортироваться элементы будут либо в зависимости от реализации интерфейса Comparable, либо используя объект Comparator, который необходимо передать в конструктор TreeMap; **LinkedHashMap** отличается от HashMap тем, что хранит ключи в порядке их вставки в Map. Эта реализаци Map лишь немного медленнее HashMap. Может содержать как null-ключи, так и null-значения.