	Основная функциональность	Примеры типичного использования
Set	**Set** - неупорядоченный набор неповторяющихся элементов. Расширяет интерфейс Collection. Если производится попытка добавить в набор элемент, который уже в нем содержится, она будет проигнорирована.	**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	**List** - служит для работы с упорядоченными коллекциями. К каждому элементы такой коллекции можно обратиться по индексу. Расширяет интерфейс Collection.	**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	**Queue** - очередь. Представляет собой структуру данных, работающую по принципу FIFO. Однако, существуют и реализации на принципе LIFO.	**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	**Map** - предназначен для работы с коллекциями-словарями, в которых содержатся ключи и соответствующие им значения(каждому ключу соответствует только одно значение). Словарь может содержать произвольное число элементов.	**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.