	Основная функциональность	Примеры типичного использования
Map	Имеет базовые методы для работы с данными вида «ключ — значение»	**HashMap** — не синхронизирована и позволяет использовать null как в качестве ключа, так и значения. Порядок хранения элементов зависит от хэш-функции **LinkedHashMap** — это упорядоченная реализация хэш-таблицы. Здесь, в отличии от HashMap, порядок итерирования равен порядку добавления элементов. Данная особенность достигается благодаря двунаправленным связям между элементами. Но это преимущество имеет также и недостаток — увеличение памяти, которое занимет коллекция. **TreeMap** — реализация Map основанная на красно-чёрных деревьях. Как и LinkedHashMap является упорядоченной. По-умолчанию, коллекция сортируется по ключам с использованием принципа "natural ordering", но это поведение может быть настроено под конкретную задачу при помощи объекта Comparator
List	Реализации этого интерфейса представляют собой упорядоченные коллекции. Кроме того, разработчику предоставляется возможность доступа к элементам коллекции по индексу и по значению	**ArrayList** — позволяет хранить любые данные, включая null в качестве элемента. Как можно догадаться из названия, его реализация основана на обычном массиве. Данную реализацию следует применять, если в процессе работы с коллекцией предплагается частое обращение к элементам по индексу. Из-за особенностей реализации поиндексное обращение к элементам выполняется за константное время O(1). Но данную коллекцию рекомендуется избегать, если требуется частое удаление/добавление элементов в середину коллекции. **LinkedList** — позволяет хранить любые данные, включая null. Особенностью реализации данной коллекции является то, что в её основе лежит двунаправленный связный список. Благодаря этому, добавление и удаление из середины, доступ по индексу, значению происходит за линейное время O(n), а из начала и конца за константное O(1).

	Основная функциональность	Примеры типичного использования
Queue	Этот интерфейс описывает коллекции с предопределённым способом вставки и извлечения элементов, а именно — очереди FIFO. Помимо методов, определённых в интерфейсе Collection, определяет дополнительные методы для извлечения и добавления элементов в очередь.	**PriorityQueue** — является единственной прямой реализацией интерфейса Queue, не считая класса LinkedList, который так же реализует этот интерфейс, но был реализован намного раньше. Особенностью данной очереди является возможность управления порядком элементов. **ArrayDeque** — реализация интерфейса Deque, который расширяет интерфейс Queue методами, позволяющими реализовать конструкцию вида LIFO. Эта коллекция представляет собой реализацию с использованием массивов, подобно ArrayList, но не позволяет обращаться к элементам по индексу и хранение null.
Set	Представляет собой неупорядоченную коллекцию, которая не может содержать дублирующиеся данные.	**HashSet** — внутри использует объект HashMap для хранения данных. В качестве ключа используется добавляемый элемент, а в качестве значения — объект-пустышка. Из-за особенностей реализации порядок элементов не гарантируется при добавлении. **LinkedHashSet** — в основе лежит LinkedHashMap. Благодаря этому отличию порядок элементов при обходе коллекции является идентичным порядку добавления элементов. **TreeSet** — в основе лежит NavigableMap, что и обуславливает его поведение. Предоставляет возможность управлять порядком элементов в коллекции при помощи объекта Comparator, либо сохраняет элементы с использованием "natural ordering".