	Основная функциональность	Примеры типичного использования
Set	Множество, содержащее только уникальные значения, все объекты должны определять методы equals() hashCode(). add(E e) — добавляем элемент в коллекцию, если такого там ещё нет. Возвращает true, если элемент добавлен addAll(Collection c) — добавляет все элементы коллекции c (если их ещё нет) clear() — удаляет все элементы коллекции contains(Object o) — возвращает true, если элемент есть в коллекции containsAll(Collection c) — возвращает true, если все элементы содержатся в коллекции equals(Object o) — проверяет, одинаковы ли коллекции hashCode() — возвращает hashCode isEmpty() — возвращает true если в коллекции нет ни одного элемента iterator() — возвращает итератор по коллекции remove(Object o) — удаляет элемент removeAll(Collection c) — удаляет элементы, принадлежащие переданной коллекции retainAll(Collection c) — удаляет элементы, не принадлежащие переданной коллекции size() — количество элементов коллекции toArray() — возвращает массив, содержащий элементы коллекции toArray(T[] a) — также возвращает массив, но (в отличии от предыдущего метода, который возвращает массив объектов типа Object) возвращает массив объектов типа, переданного в параметре.	Чаще всего используется для тестирования принадлежности объекта к множеству. Реализация в классах: **HashSet** – неотсортированная и неупорядоченная коллекция, для вставки элемента используются методы hashCode() и equals(…). Чем эффективней реализован метод hashCode(), тем эффективней работает коллекция. HashSet используется в случае, когда порядок элементов не важен, но важно чтобы в коллекции все элементы были уникальны. **TreeSet** - реализует интерфейс NavigableSet, который поддерживает элементы в отсортированном по возрастанию порядке. Для хранения объектов использует бинарное дерево. При добавлении объекта в дерево он сразу же размещается в необходимую позицию с учетом сортировки. Сортировка происходит благодаря тому, что все добавляемые элементы реализуют интерфейсы Comparator и Comparable. Обработка операций удаления и вставки объектов происходит медленнее, чем в хэш- множествах, но быстрее, чем в списках. Используется в том случае, если необходимо использовать операции, определенные в интерфейсе SortedSet, NavigableSet или итерацию в определенном порядке. **LinkedHashSet** - поддерживает связный список элементов набора в том порядке, в котором они вставлялись. Это позволяет организовать упорядоченную итерацию вставки в набор. То есть, когда идет перебор объекта класса LinkedHashSet с применением итератора, элементы извлекаются в том порядке, в каком они были добавлены.
List	Представляет собой неупорядоченную коллекцию, в которой допустимы дублирующие значения. Иногда их называют последовательностями (sequence). Элементы такой коллекции пронумерованы, начиная от нуля, к ним можно обратиться по индексу. get(int index) возвращает объект, находящийся в позиции index; set(int index, E element) заменяет элемент, находящийся в позиции index объектом element; boolean add(E element) добавляет элемент в список void add(int index, E element) вставляет элемент element в позицию index,при этом список раздвигается remove(int index) удаляет элемент, находящийся на позиции index boolean addAll(int index, Collection<? extends E> c) добавляет все элементы коллекции c в список, начиная с позиции index int indexOf(Object o) возвращает индекс первого появления элемента o в списке; int lastIndexOf(Object o) возвращает индекс последнего появления элемента o в списке;	**ArrayList** инкапсулирует в себе обычный массив, длина которого автоматически увеличивается при добавлении новых элементов. Так как ArrayList использует массив, то время доступа к элементу по индексу минимально (В отличии от LinkedList). При удалении произвольного элемента из списка, все элементы находящиеся «правее» смещаются на одну ячейку влево, при этом реальный размер массива (его емкость, capacity) не изменяется. Если при добавлении элемента, оказывается, что массив полностью заполнен, будет создан новый массив размером (n * 3) / 2 + 1, в него будут помещены все элементы из старого массива + новый, добавляемый элемент. **LinkedList** - Двусвязный список. Это структура данных, состоящая из узлов, каждый из которых содержит как собственно данные, так и две ссылки («связки») на следующий и предыдущий узел списка. Доступ к произвольному элементу осуществляется за линейное время (но доступ к первому и последнему элементу списка всегда осуществляется за константное время — ссылки постоянно хранятся на первый и

	ListIterator<E> listIterator() возвращает итератор на список ListIterator<E> listIterator(int index) возвращает итератор на список, установленный на элемент с индексом index List<E> subList(int from, int to) возвращает новый список, представляющий собой часть данного (начиная с позиции from до позиции to-1 включительно).	последний, так что добавление элемента в конец списка вовсе не значит, что придется перебирать весь список в поисках последнего элемента). В целом же, LinkedList в абсолютных величинах проигрывает ArrayList и по потребляемой памяти и по скорости выполнения операций.
Queue	Очередь, предназначенная для размещения элемента перед его обработкой. Расширяет коллекцию методами для вставки, выборки и просмотра элементов Очередь – хранилище элементов, предназначенных для обработки. Кроме базовых методов Collection очередь(Queue) предоставляет дополнительные методы по добавлению, извлечению и проверке элементов. Чаще всего порядок выдачи элементов соответствует FIFO (first-in, first-out), но в общем случае определяется конкретной реализацией. Очереди не могут хранить null. У очереди может быть ограничен размер. element() возвращает, но не удаляет головной элемент очереди boolean offer(E o) добавляет в конец очереди новый элемент и возвращает true, если вставка удалась. E peek() возвращает первый элемент очереди, не удаляя его. E poll() возвращает первый элемент и удаляет его из очереди E remove() возвращает и удаляет головной элемент очереди	**PriorityQueue** – это класс очереди с приоритетами. По умолчанию очередь с приоритетами размещает элементы согласно естественному порядку сортировки используя Comparable. Элементу с наименьшим значением присваивается наибольший приоритет. Если несколько элементов имеют одинаковый наивысший элемент – связь определяется произвольно. Также можно указать специальный порядок размещения, используя Comparator. **ArrayDequ**. Этот класс представляют обобщенную двунаправленную очередь, наследуя функционал от класса AbstractCollection и применяя интерфейс Deque.
Map	Интерфейс Map соотносит уникальные ключи со значениями. Ключ — это объект, который вы используете для последующего извлечения данных. Задавая ключ и значение, вы можете помещать значения в объект карты. После того как это значение сохранено, можно получить его по ключу. V put(K key, V value) запись V get(Object key) получение значение V remove(Object key) удаление boolean containsKey(Object key) наличие ключа boolean containsValue(Object value) наличие значения int size() размер отображения boolean isEmpty() проверка на пустоту void putAll(Map m) добавление всех пар void clear() полная очистка Set keySet() множество ключей Collection values() коллекция значений Set<Map.Entry> entrySet() множество пар	**HashMap** — основан на хэш-таблицах, реализует интерфейс Map (что подразумевает хранение данных в виде пар ключ/значение). Ключи и значения могут быть любых типов, в том числе и null. Данная реализация не дает гарантий относительно порядка элементов с течением времени. **LinkedHashMap** - расширяет класс HashMap. Он создает связный список элементов в карте, расположенных в том порядке, в котором они вставлялись. Это позволяет организовать перебор карты в порядке вставки. То есть, когда происходит итерация по коллекционному представлению объекта класса LinkedHashMap, элементы будут возвращаться в том порядке, в котором они вставлялись. Вы также можете создать объект класса LinkedHashMap, возвращающий свои элементы в том порядке, в котором к ним в последний раз осуществлялся доступ. **TreeMap** - расширяет класс AbstractMap и реализует интерфейс NavigatebleMap. Он создает коллекцию, которая для хранения элементов применяет дерево. Объекты сохраняются в отсортированном порядке по возрастанию. Время доступа и извлечения элементов достаточно мало, что делает класс TreeMap блестящим выбором для хранения больших объемов отсортированной информации, которая должна быть быстро найдена.