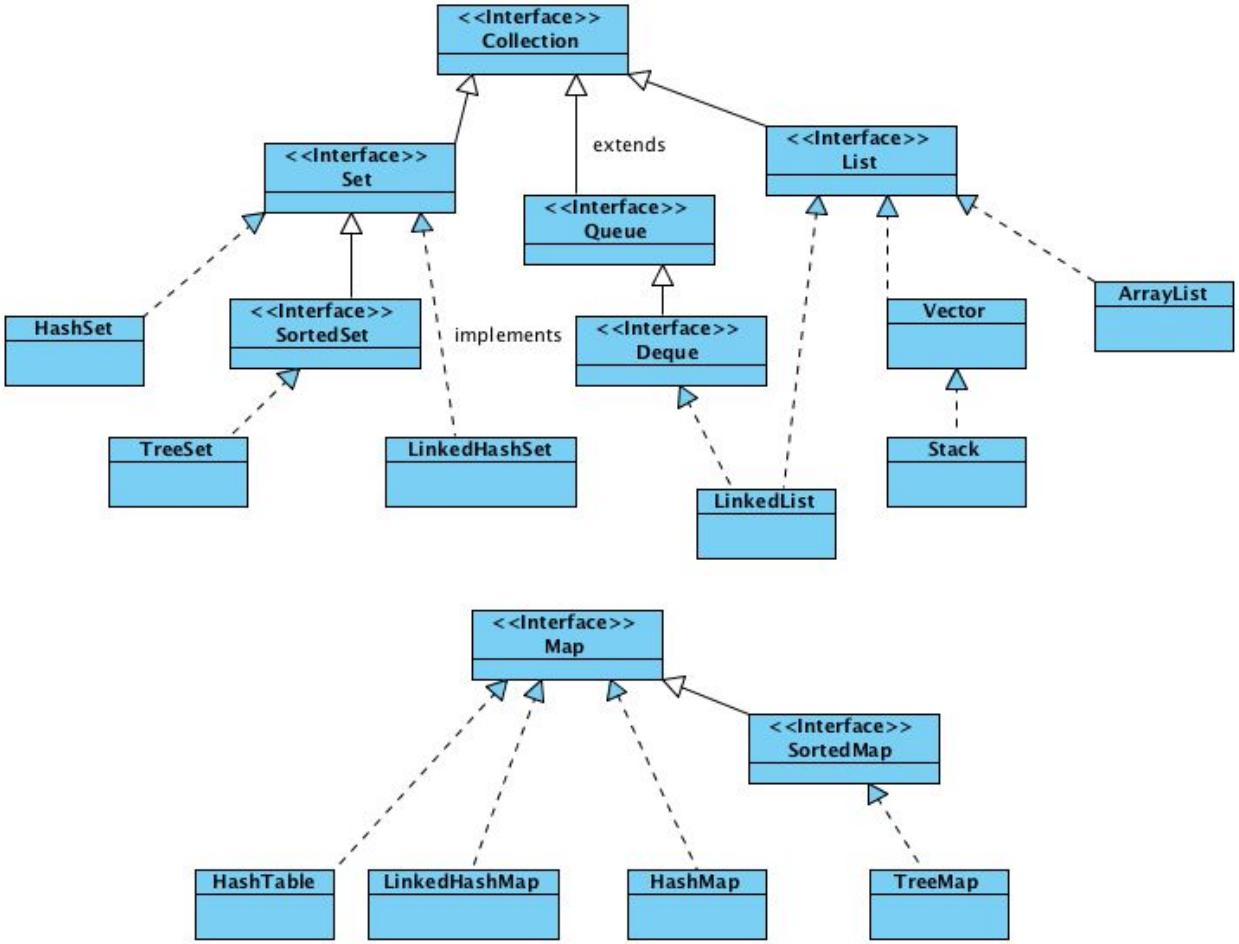
Java Collection Framework

Java Collection Framework

Иерархия интерфейсов и их реализаций, которая является частью JDK и позволяет разработчику пользоваться большим количеством структур данных из «коробки».

Java Collection Framework



java.util.Collection

Основной интерфейс, который описывает базовые методы, которыми должна обладать любая коллекция. Т.е. если какой-то класс претендует на звание «коллекция» — он должен реализовать те методы, которые описаны в этом интерфейсе.

java.util.Collection

Основные методы:

- boolean add(E e);
- boolean remove(Object o);
- int size();
- boolean isEmpty();
- boolean contains(Object o);

java.util.Collection

Есть также **методы, позволяющие преобразовать коллекцию в массив**:

Object[] toArray()

<T> T[] toArray(T[] a)

Рекомендуется использовать второй метод, так как он позволяет преобразовать в массив определённого типа.

String[] array = collection.toArray(new String[0]);.

java.util.Collection

Итератор

```
Iterator<Integer> iterator = intCollection.iterator();
while (iterator.hasNext()){
   if (iterator.next() > 2){
       iterator.remove(); позволяет удалять элементы при прохождении по
коллекции
   }
}
System.out.println(intCollection.toString());
```

java.util.List

Интерфейс для операций с коллекцией, которая является списком.
Список обладает следующими важными признаками:

- Список может включать одинаковые элементы
- Элементы в списке хранятся в том порядке, в котором они туда помещались. Самопроизвольных перемещений элементов в нем не происходит — только с вашего ведома. Например, вы можете добавить элемент на какую-то позицию и тогда произойдет сдвиг других элементов.
- Можно получить доступ к любому элементу по его порядковому номеру/индексу внутри списка

java.util.List

Основные методы:

- void add(int index, E obj)
- boolean addAll(int index, Collection<? extends E> col) Если в результате добавления список был изменен, то возвращается true, иначе возвращается false
- E get(int index)
- int indexOf(Object obj): возвращает индекс первого вхождения объекта obj в список. Если объект не найден, то возвращается -1
- int lastIndexOf(Object obj)

java.util.List

Основные методы:

- E remove(int index): удаляет объект из списка по индексу index, возвращая при этом удаленный объект
- E set(int index, E obj): присваивает значение объекта obj элементу, который находится по индексу index
- List<E> subList(int start, int end): получает набор элементов, которые находятся в списке между индексами start и end

java.util.Set

Интерфейс для хранения множества. В отличии от java.util.List этот интерфейс как раз не может иметь одинаковые элементы и порядок хранения элементов в множестве может меняться при добавлении/удалении/изменении элемента. Может возникнуть вопрос, зачем такая коллекция нужна — это удобно в случае, когда вы создаете набор уникальных элементов из какой-то группы элементов.

java.util.Set

Основные методы:

- boolean add(Object o) Возвращает true, если элемент добавлен.
- addAll(Collection c) Добавление элементов коллекции, если отсутствуют.
- clear() Очистка коллекции.
- contains(Object o) Проверка присутствия элемента в наборе.
- containsAll(Collection c) Проверка присутствия коллекции в наборе.
- remove(Object o)Удаление элемента из набора.
- removeAll(Collection c) Удаление из набора всех элементов переданной коллекции.
- retainAll(Collection c) Удаление элементов, не принадлежащих переданной коллекции.

java.util.ArrayList

реализует интерфейс List. Как известно, в Java массивы имеют фиксированную длину, и после того как массив создан, он не может расти или уменьшаться.
ArrayList может менять свой размер во время исполнения программы, при этом не обязательно указывать размерность при создании объекта.
Элементы ArrayList могут быть абсолютно любых типов в том числе и null.

java.util.ArrayList

Создание объекта
ArrayList<String> list = new ArrayList<String>();

Методы ensureCapacity() и trimToSize()

Если заранее известно, сколько элементов следует хранить, то перед заполнением массива вызовите метод ensureCapacity():
list.ensureCapacity(50);

Если вы уверены, что списочный массив будет иметь постоянный размер, используйте метод trimToSize().

list.trimToSize(50);

java.util.ArrayList

Добавление элементов

list.add("0");

Удалять элементы можно двумя способами:

- по индексу remove(index)
- по значению remove(value)

list.remove(5);

java.util.ArrayList

ArrayList из массива

```
String[] strArr = {"String 1", "String 2", "String 3", "String 4"};
ArrayList<String> listFromArr = new ArrayList<>(Arrays.asList(strArr));
for (String str: listFromArr){
    System.out.println(str);
}
```

java.util.ArrayList

Итоги

- Быстрый доступ к элементам по индексу;
- Медленный доступ, когда вставляются и удаляются элементы из «середины» списка (лучше использовать LinkedList);
- Позволяет хранить любые значения в том числе и null;

java.util.LinkedList

класс, реализующий два интерфейса - List и Deque. Это обеспечивает возможность создания двунаправленной очереди из любых (в том числе и null) элементов. Каждый объект, помещенный в связанный список, является узлом. Каждый узел содержит элемент, ссылку на предыдущий и следующий узел. Фактически связанный список состоит из последовательности узлов

java.util.LinkedList

Создание объекта:

```
LinkedList<Integer> nums = new LinkedList<>();
```

Добавление объекта в конец связанного списка

```
nums.add(23);
nums.add(12);
```

Добавление объекта в середину связанного списка

```
nums.add(1, 13);


for (Integer i: nums){
    System.out.println(i);
}
```

java.util.LinkedList и java.util.ArrayList в документации

Посмотрите доступные **конструкторы LinkedList** и его **методы**:
https://docs.oracle.com/javase/9/docs/api/java/util/LinkedList.html

Доступные **конструкторы ArrayList** и его **методы**:
https://docs.oracle.com/javase/9/docs/api/java/util/ArrayList.html

java.util.HashSet

Коллекция, не позволяющая хранить одинаковые объекты. HashSet инкапсулирует в себе объект HashMap.

Выгода от хеширования состоит в том, что оно обеспечивает константное время выполнения методов add(), contains(), remove() и size() , даже для больших наборов.

Это наиболее эффективная реализация,
но она не гарантирует сохранение порядка элементов при обходе.

java.util.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.

Эта реализация избавляет от хаотичного порядка элементов и лишь незначительно хуже по эффективности HashSet-а.

java.util.SortedSet

Представляет собой множество, в котором элементы хранятся в порядке возрастания.

SortedSet предоставляет следующие методы:

- E first(): возвращает первый элемент набора
- E last(): возвращает последний элемент набора
- SortedSet<E> headSet(E end): возвращает объект SortedSet, который содержит все элементы первичного набора до элемента end
- SortedSet<E> subSet(E start, E end): возвращает объект SortedSet, который содержит все элементы первичного набора между элементами start и end
- SortedSet<E> tailSet(E start): возвращает объект SortedSet, который содержит все элементы первичного набора, начиная с элемента tail

java.util.TreeSet

Коллекция, которая хранит свои элементы в виде упорядоченного по значениям дерева.
TreeSet инкапсулирует в себе TreeMap, который в свою очередь использует сбалансированное бинарное красно-черное дерево для хранения элементов.
TreeSet хорош тем, что для операций add, remove и contains потребуется гарантированное время log(n).

java.util.HashSet, java.util.LinkedHashSet и java.util.TreeSet в документации

Конструкторы HashSet и его **методы**:
https://docs.oracle.com/javase/9/docs/api/java/util/HashSet.html

Конструкторы LinkedHashSet и его **методы**:

https://docs.oracle.com/javase/9/docs/api/java/util/LinkedHashSet.html

Конструкторы TreeSet и его **методы**:
https://docs.oracle.com/javase/9/docs/api/java/util/TreeSet.html