Java I

Урок VIII-IX

Java Collections

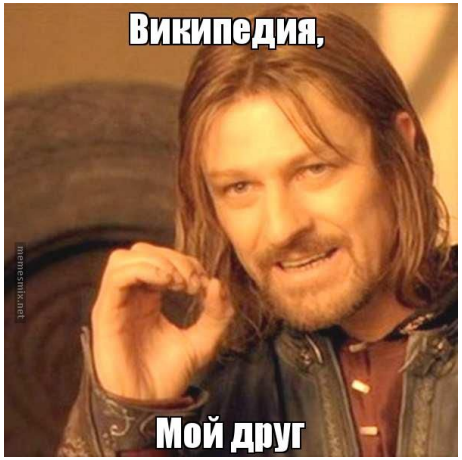
Структуры данных

- Программы = алгоритмы + структуры данных (Николаус Вирт)

Что такое коллекции?

Коллекция в программировании — программный объект, содержащий в себе, тем или иным образом, набор значений одного или различных типов, и позволяющий обращаться к этим значениям.

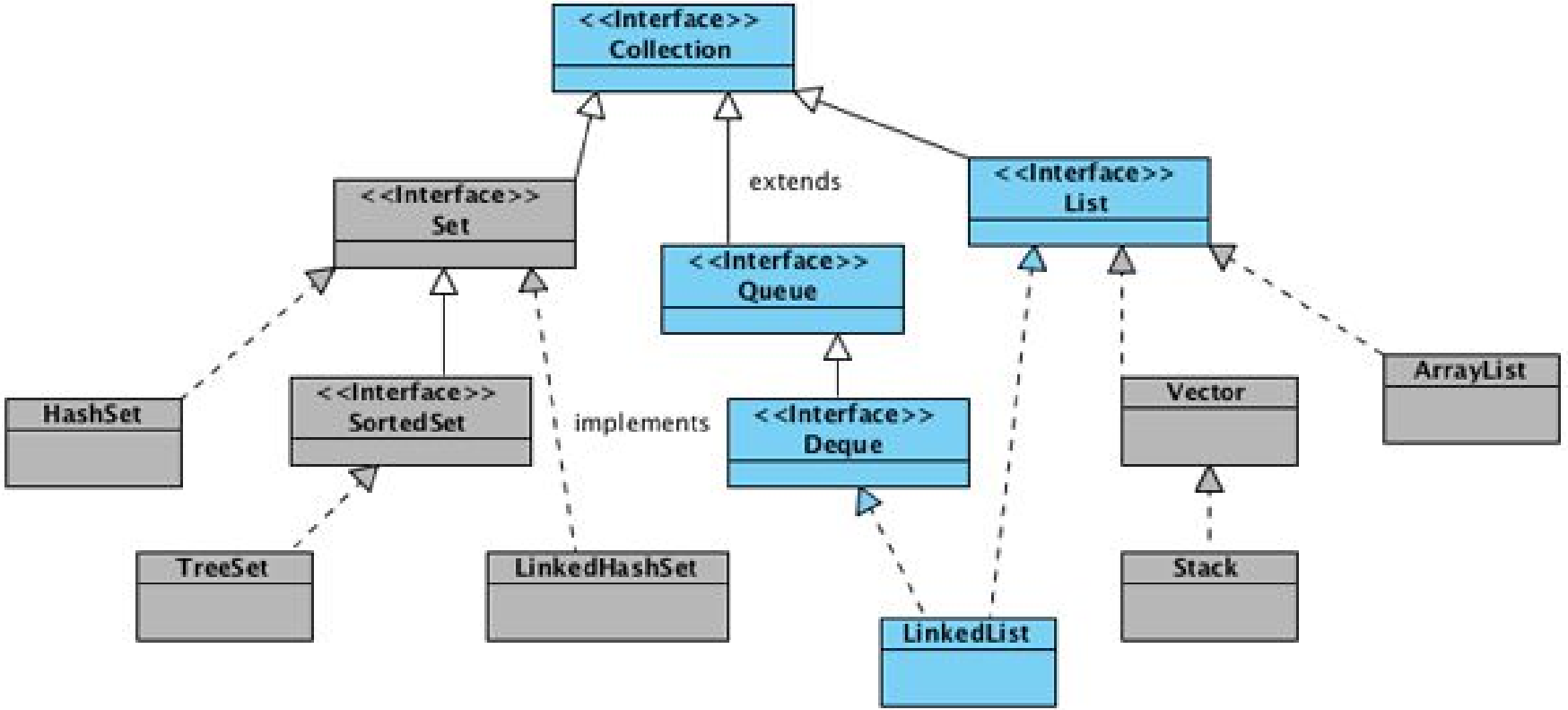
Назначение коллекции — служить хранилищем объектов и обеспечивать доступ к ним.



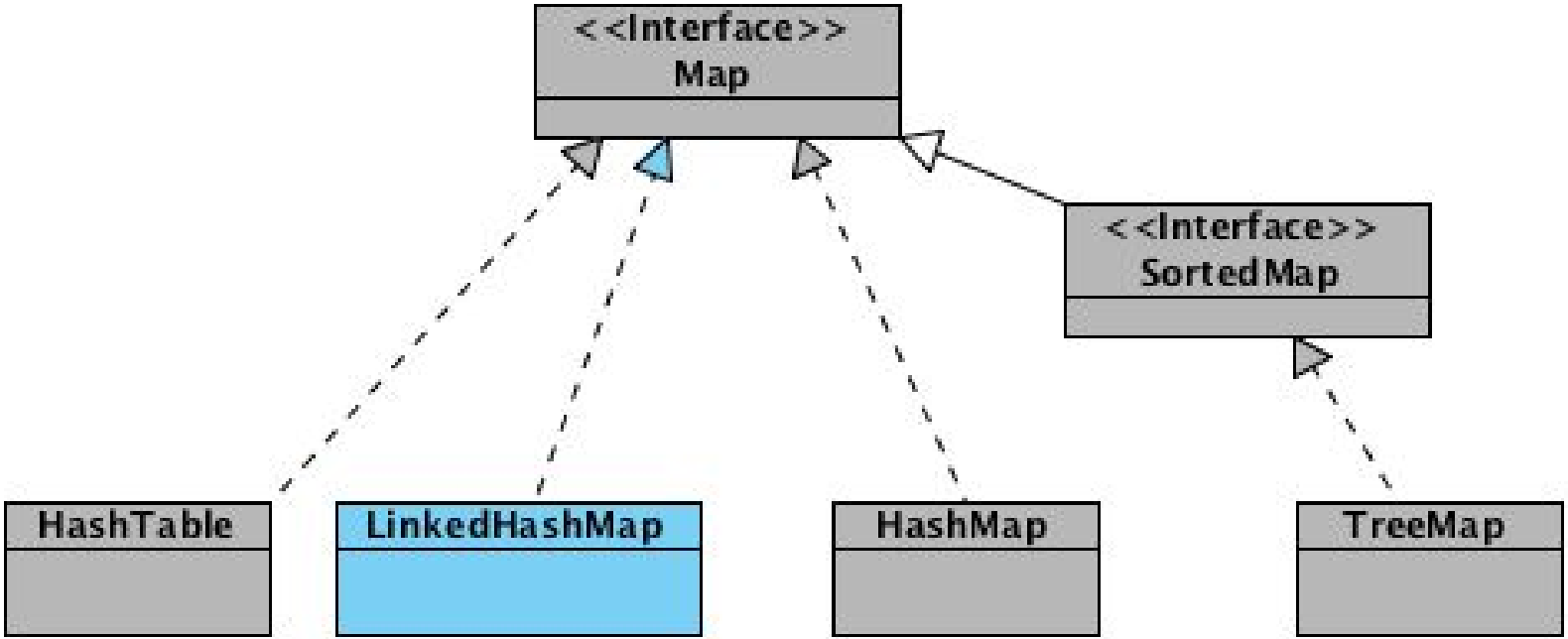
Java Collections API

- Набор интерфейсов и их реализаций, которая является частью JDK и позволяет разработчику пользоваться большим количеством структур данных из «коробки».
- Объект коллекции объединяет множество элементов в единую структуру
- API содержит интерфейсы которые задают абстрактное поведение для каждой коллекции.
- API содержит реализации этих интерфейсов. Каждая конкретная реализация использует различный алгоритм и имеют различное поведение.

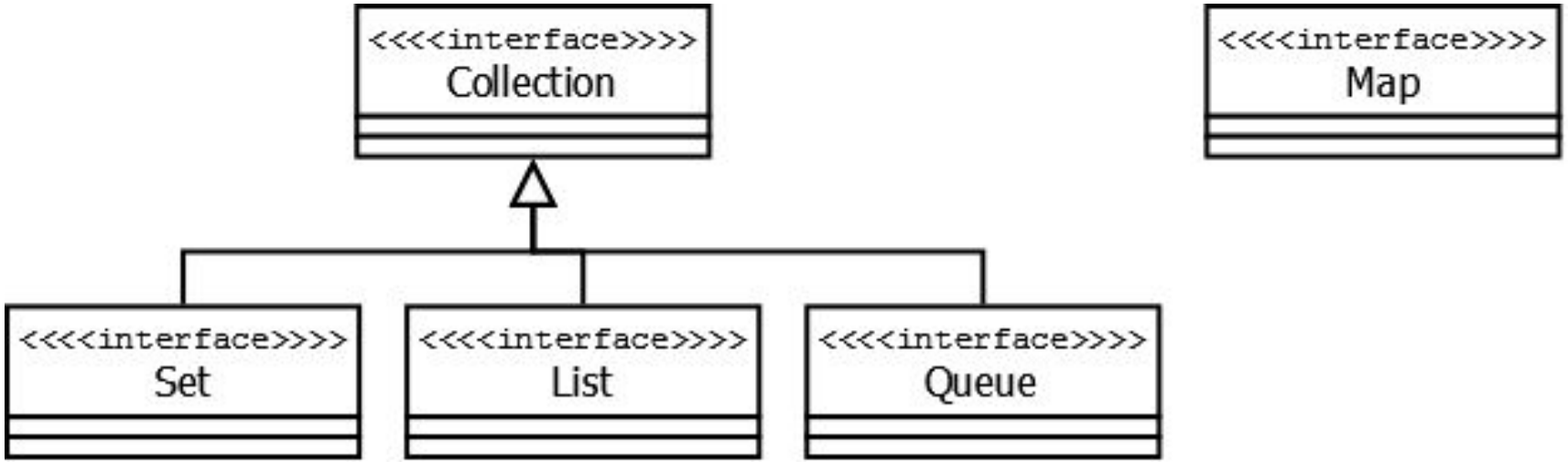
Иерархия Java Collections API



Иерархия Java Collections API



Основные интерфейсы



Map

Map Interface

Интерфейс Map из пакета java.util описывает своеобразную коллекцию, состоящую не из элементов, а из пар "ключ — значение". У каждого ключа может быть только одно значение, что соответствует математическому понятию однозначной функции, или отображения (map).

Такую коллекцию часто называют еще словарем (dictionary) или ассоциативным массивом (associative array).

https://docs.oracle.com/javase/8/docs/api/java/util/Map.html

Map Interface

Интерфейс Map содержит методы, работающие с ключами и значениями:

- **`boolean containsKey(Object key)`** — проверяет наличие ключа key;
- **`boolean containsValue(Object value)`** — проверяет наличие значения value;
- **`Set entrySet()`** — представляет коллекцию в виде множества с элементами в виде пар из данного отображения, с которыми можно работать методами вложенного интерфейса Map.Entry;
- **`Object get(Object key)`** — возвращает значение, отвечающее ключу key;

Map Interface

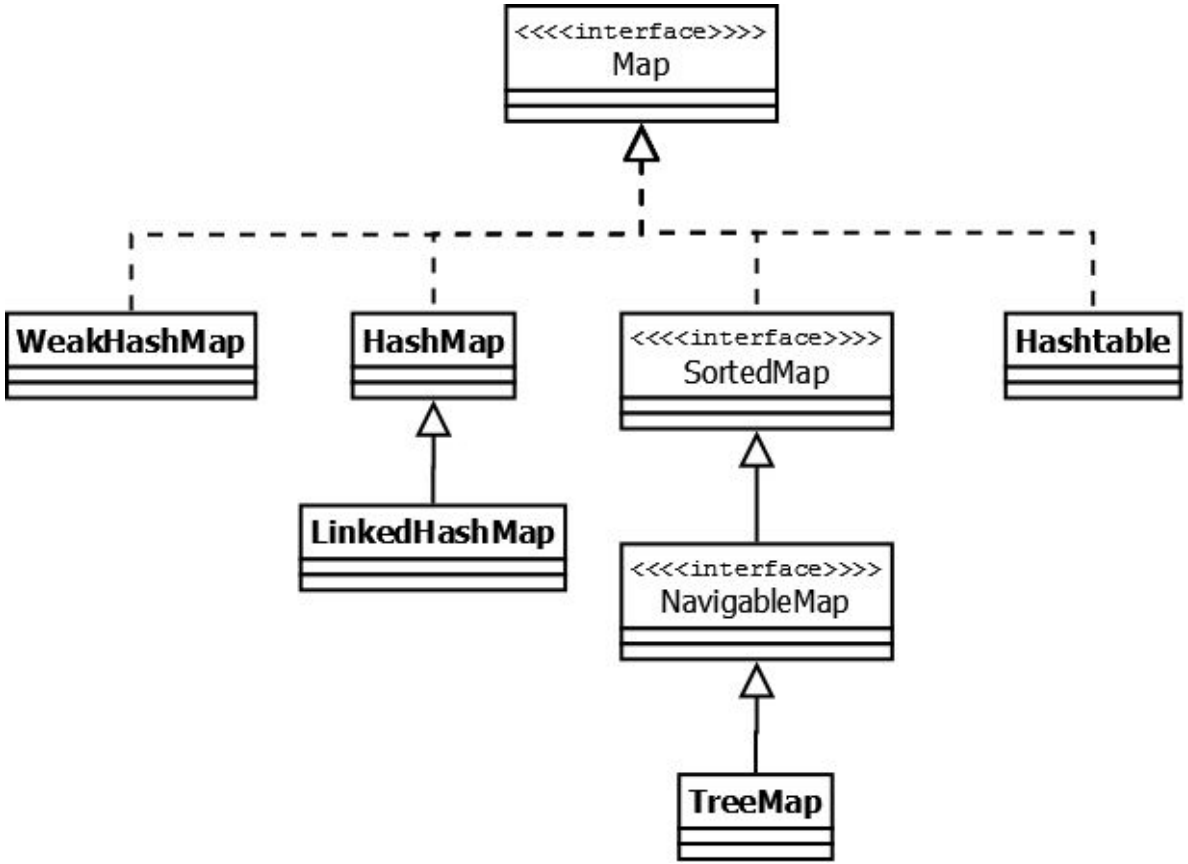
- **`Set keySet()`** — представляет ключи коллекции в виде множества;
- **`Object put(Object key, Object value)`** — добавляет пару "key — value", если такой пары не было, и заменяет значение ключа key, если такой ключ уже есть в коллекции;
- **`void putAll(Map m)`** — добавляет к коллекции все пары из отображения m;
- **`Collection values()`** — представляет все значения в виде коллекции.

Map.Entry Interface

В интерфейс Map вложен интерфейс Map.Entry, содержащий методы работы с отдельной парой отображения.

- методы getKey() и getValue() позволяют получить ключ и значение пары;
- метод setValue(Object value) меняет значение в данной паре.

Map Interface



Map Interface - Hashtable

Hashtable — реализация такой структуры данных, как хэш-таблица. Она не позволяет использовать null в качестве значения или ключа. Эта коллекция была реализована раньше, чем Java Collection Framework, но в последствии была включена в его состав. Как и другие коллекции из Java 1.0, Hashtable является синхронизированной (почти все методы помечены как synchronized). Из-за этой особенности у неё имеются существенные проблемы с производительностью и, начиная с Java 1.2, в большинстве случаев рекомендуется использовать другие реализации интерфейса Map ввиду отсутствия у них синхронизации.

Map Interface - HashMap

HashMap — коллекция является альтернативой Hashtable. Двумя основными отличиями от Hashtable являются то, что HashMap не синхронизирована и HashMap позволяет использовать null как в качестве ключа, так и значения. Так же как и Hashtable, данная коллекция не является упорядоченной: порядок хранения элементов зависит от хэш-функции. Добавление элемента выполняется за константное время O(1), но время удаления, получения зависит от распределения хэш-функции. В идеале является константным, но может быть и линейным O(n).

Map Interface - LinkedHashMap

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 (аналогично LinkedList). Но это преимущество имеет также и недостаток — увеличение памяти, которое занимает коллекция.

Map Interface - TreeMap

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, которые указывается в качестве параметра при создании объекта TreeMap.

Map Interface - WeakHashMap

WeakHashMap — реализация хэш-таблицы, которая организована с использованием weak references. Другими словами, Garbage Collector автоматически удалит элемент из коллекции при следующей сборке мусора, если на ключ этого элеметна нет жёстких ссылок.

Map Example

Создание

```
Map<String, Person> map = new HashMap<>();
```

Добавление

```
map.put("test0", new Person("Test0", "Test0"));
```

Итерация по ключам

```
for (String s : map.keySet()) {
  System.out.println(s);
}
```

Итерация по значениям

```
for (Person p : map.values()) {

  System.out.println(p.getName() + " " + p.getSurname());

}
```

Итерация по ключам и значениям

```
for (Map.Entry<String, Person> stringPersonEntry : map.entrySet()) {

  String key = stringPersonEntry.getKey();

  Person value = stringPersonEntry.getValue();

  System.out.println(key + ": " + value.getName() + " " +  value.getSurname());

}
```

Размер

```
System.out.println("Size: " + map.size());
```

Получение элемента

```
Person p = map.get("test0");

System.out.println(p.getName() + " " + p.getSurname());
```

Удаление элемента

```
map.remove("test0");
```

Collection

Collection Interface

Интерфейс Collection из пакета java.util описывает общие свойства коллекций List, Set и Queue. Он содержит методы добавления и удаления элементов, проверки и преобразования элементов.

https://docs.oracle.com/javase/8/docs/api/java/util/Collection.html

Collection Interface

- **`boolean add(Object obj)`** — добавляет элемент obj в конец коллекции; возвращает false, если такой элемент в коллекции уже есть, а коллекция не допускает повторяющиеся элементы; возвращает true, если добавление прошло удачно;
- **`boolean addAll(Collection col)`** — добавляет все элементы коллекции **`col`** в конец данной коллекции;
- **`void clear()`** — удаляет все элементы коллекции;
- **`boolean contains(Object obj)`** — проверяет наличие элемента obj в коллекции;

Collection Interface

- **`boolean containsAll(Collection col)`** — проверяет наличие всех элементов коллекции **`col`** в данной коллекции;
- **`boolean isEmpty()`** — проверяет, пуста ли коллекция;
- **`Iterator iterator()`** — возвращает итератор данной коллекции;
- **`boolean remove(Object obj)`** — удаляет указанный элемент из коллекции; возвращает false, если элемент не найден, true, если удаление прошло успешно;
- **`boolean removeAll(Collection coll)`** — удаляет элементы указанной коллекции, лежащие в данной коллекции;

Collection Interface

- **`boolean retainAll(Collection coll)`** — удаляет все элементы данной коллекции, кроме элементов коллекции **`coll`**;
- **`int size()`** — возвращает количество элементов в коллекции;
- **`Object[] toArray()`** — возвращает все элементы коллекции в виде массива;
- **`Object[] toArray(Object[] a)`** — записывает все элементы коллекции в массив **`a`**, если в нем достаточно места.

List

List Interface

Интерфейс List из пакета java.util, расширяющий интерфейс Collection, описывает методы работы с упорядоченными коллекциями. Иногда их называют последовательностями (sequence). Элементы такой коллекции пронумерованы, начиная от нуля, к ним можно обратиться по индексу. В отличие от коллекции Set элементы коллекции List могут повторяться.

https://docs.oracle.com/javase/8/docs/api/java/util/List.html

List Interface

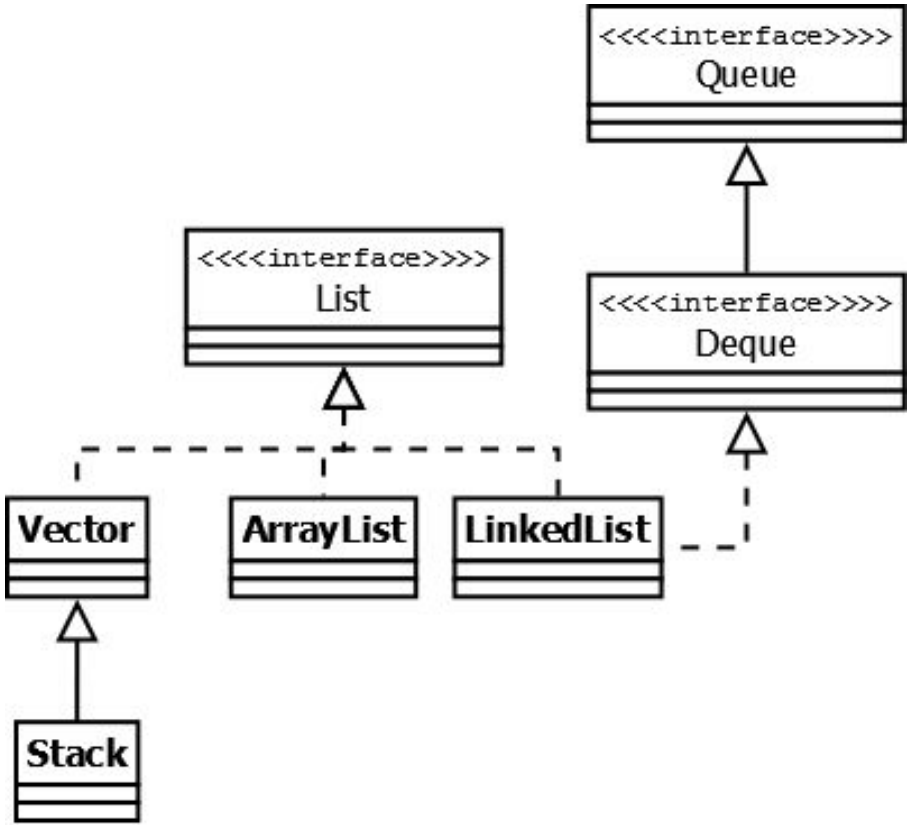
Интерфейс List добавляет к методам интерфейса Collection методы, использующие индекс index элемента:

- **`void add(int index, Object obj)`** — вставляет элемент obj в позицию index; старые элементы, начиная с позиции index, сдвигаются, их индексы увеличиваются на единицу;
- **`boolean addAll(int index, Collection coll)`** — вставляет все элементы коллекции coll;
- **`Object get(int index)`** — возвращает элемент, находящийся в позиции index;

List Interface

- **`int indexOf(Object obj)`** — возвращает индекс первого появления элемента obj в коллекции;
- **`int lastIndexOf(Object obj)`** — возвращает индекс последнего появления элемента obj в коллекции;
- **`ListIterator listIterator()`** — возвращает итератор коллекции;
- **`ListIterator listIterator(int index)`** — возвращает итератор конца коллекции от позиции index;
- **`Object set(int index, Object obj)`** — заменяет элемент, находящийся в позиции index, элементом obj;
- **`List subList(int from, int to)`** — возвращает часть коллекции от позиции from включительно до позиции to исключительно.

List Interface



List Interface - Vector

Vector — реализация динамического массива объектов. Позволяет хранить любые данные, включая null в качестве элемента. Vector появился в JDK версии Java 1.0, но как и Hashtable, эту коллекцию не рекомендуется использовать, если не требуется достижения потокобезопасности. Потому как в Vector, в отличии от других реализаций List, все операции с данными являются синхронизированными. В качестве альтернативы часто применяется аналог — ArrayList.

List Interface - Stack

Stack — данная коллекция является расширением коллекции Vector. Была добавлена в Java 1.0 как реализация стека LIFO (last-in-first-out). Является частично синхронизированной коллекцией (кроме метода добавления push()). После добавления в Java 1.6 интерфейса Deque, рекомендуется использовать именно реализации этого интерфейса, например ArrayDeque.

List Interface - ArrayList

ArrayList — как и Vector является реализацией динамического массива объектов. 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 предполагается 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.

List Interface - LinkedList

LinkedList — ещё одна реализация List. Позволяет хранить любые данные, включая null. Особенностью реализации данной коллекции является то, что в её основе лежит двунаправленный связный список (каждый элемент имеет ссылку на предыдущий и следующий). 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). Также, ввиду реализации, данную коллекцию можно использовать как стек или очередь. Для этого в ней реализованы соответствующие методы.

Примеры List

Создание

```
List<Person> list = new ArrayList<>();
```

Добавление

```
list.add(new Person("Test0", "Test0"));
```

Итерация

```
for (Person p : list) {

  System.out.println(p.getName() + " " + p.getSurname());

}
```

Итерация 2

```
for (Iterator<Person> iterator = list.iterator(); iterator.hasNext(); ) {

  Person p = iterator.next();

  System.out.println(p.getName() + " " + p.getSurname());

}
```

Размер

```
System.out.println("Size: " + list.size());
```

Получение элемента

```
Person p = list.get(0);

System.out.println(p.getName() + " " + p.getSurname());
```

Удаление элемента

```
list.remove(p);
```

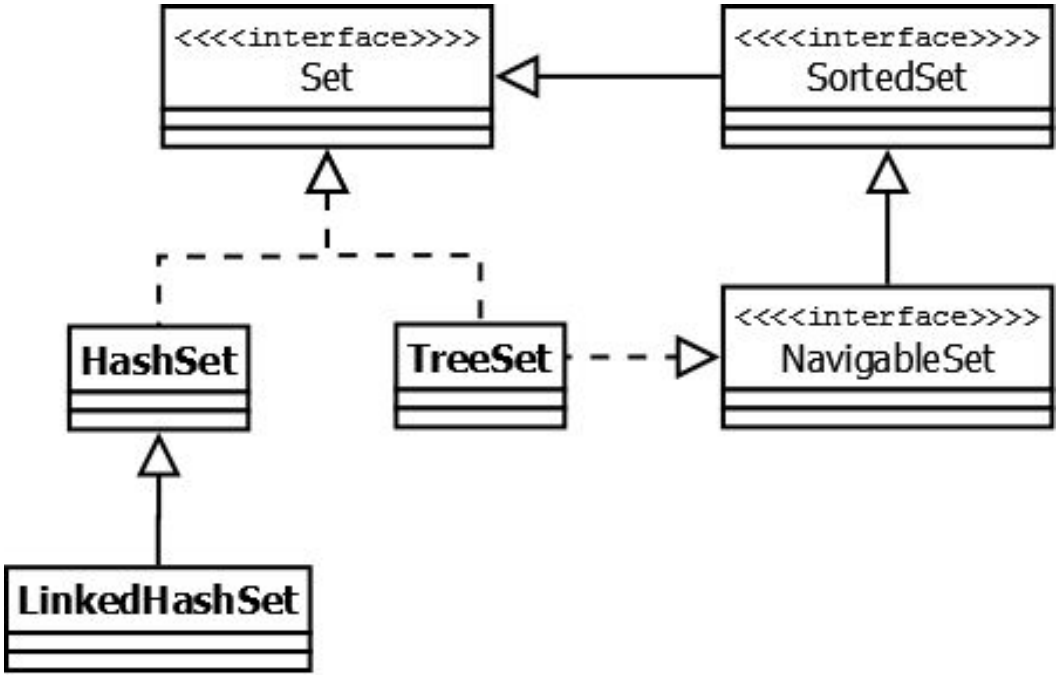
Set

Set Interface

Интерфейс Set из пакета java.util, расширяющий интерфейс Collection, описывает неупорядоченную коллекцию, не содержащую повторяющихся элементов. Это соответствует математическому понятию множества (set). Такие коллекции удобны для проверки наличия или отсутствия у элемента свойства, определяющего множество. Новые методы в интерфейс Set не добавлены, просто метод add() не станет добавлять еще одну копию элемента, если такой элемент уже есть в множестве.

https://docs.oracle.com/javase/8/docs/api/java/util/Set.html

Set Interface



Set Interface - HashSet

HashSet — реализация интерфейса Set, базирующаяся на HashMap. Внутри использует объект HashMap для хранения данных. В качестве ключа используется добавляемый элемент, а в качестве значения — объект-пустышка (new Object()). Из-за особенностей реализации порядок элементов не гарантируется при добавлении.

Set Interface - LinkedHashSet

LinkedHashSet — отличается от HashSet только тем, что в основе лежит LinkedHashMap вместо HashSet. Благодаря этому отличию порядок элементов при обходе коллекции является идентичным порядку добавления элементов.

Set Interface - TreeSet

TreeSet — аналогично другим классам-реализациям интерфейса Set содержит в себе объект NavigableMap, что и обуславливает его поведение. Предоставляет возможность управлять порядком элементов в коллекции при помощи объекта Comparator, либо сохраняет элементы с использованием "natural ordering".

Примеры Set

Создание

```
Set<Person> set = new HashSet<>();
```

Добавление

```
set.add(new Person("Test0", "Test0"));
```

Итерация

```
for (Person p : set) {

  System.out.println(p.getName() + " " + p.getSurname());

}
```

Итерация 2

```
for (Iterator<Person> iterator = set.iterator(); iterator.hasNext(); ) {
  Person p = iterator.next();
  System.out.println(p.getName() + " " + p.getSurname());
}
```

Размер

```
System.out.println("Size: " + set.size());
```

Получение элемента

Возможно только через цикл

Удаление элемента

Возможно только через цикл

Queue

Queue Interface

Интерфейс Queue из пакета java.util, расширяющий интерфейс Collection, описывает методы работы с очередями. Очередью называется коллекция, элементы в которую добавляются с одного конца, а удаляются с другого конца. Хороший пример такой коллекции — обычная житейская очередь в магазине или на автобусной остановке. Такой порядок обработки называется FIFO (First In — First Out, первым пришел — первым ушел).

https://docs.oracle.com/javase/8/docs/api/java/util/Queue.html

Queue Interface

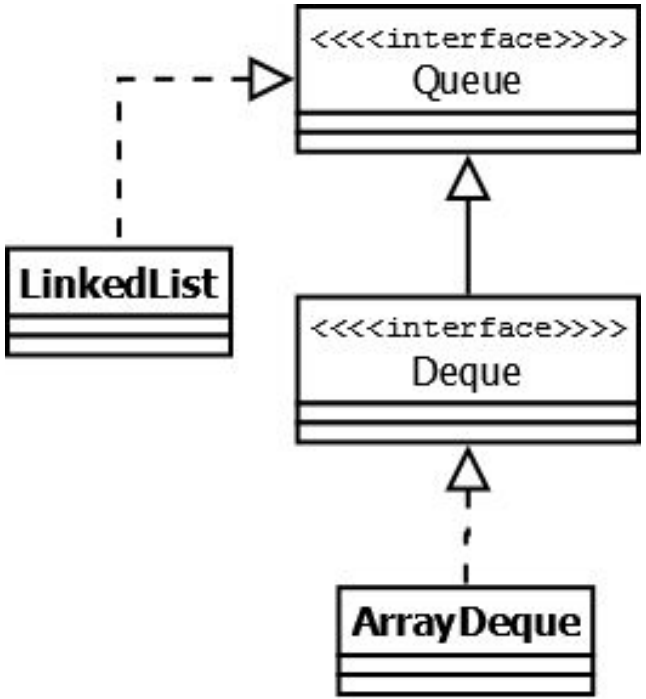
Интерфейс Queue добавляет к методам интерфейса Collection методы, характерные для очередей

- **`Object element()`** — возвращает первый элемент очереди, не удаляя его из очереди. Метод выбрасывает исключение, если очередь пуста;
- **`Object peek()`** — возвращает первый элемент очереди, не удаляя его. В отличие от метода element() не выбрасывает исключение;
- **`Object remove()`** — возвращает первый элемент очереди и удаляет его из очереди. Метод выбрасывает исключение, если очередь пуста;
- **`Object poll()`** — возвращает первый элемент очереди и удаляет его из очереди. В отличие от метода remove() не выбрасывает исключение;

Queue Interface

- **`boolean offer(Object obj)`** — вставляет элемент в конец очереди и возвращает true,если вставка удалась.

Queue Interface



Queue Interface - PriorityQueue

PriorityQueue — является единственной прямой реализацией интерфейса Queue (была добавлена, как и интерфейс Queue, в Java 1.5), не считая класса LinkedList, который так же реализует этот интерфейс, но был реализован намного раньше. Особенностью данной очереди является возможность управления порядком элементов. По-умолчанию, элементы сортируются с использованием «natural ordering», но это поведение может быть переопределено при помощи объекта Comparator, который задаётся при создании очереди. Данная коллекция не поддерживает null в качестве элементов.

Queue Interface - ArrayDeque

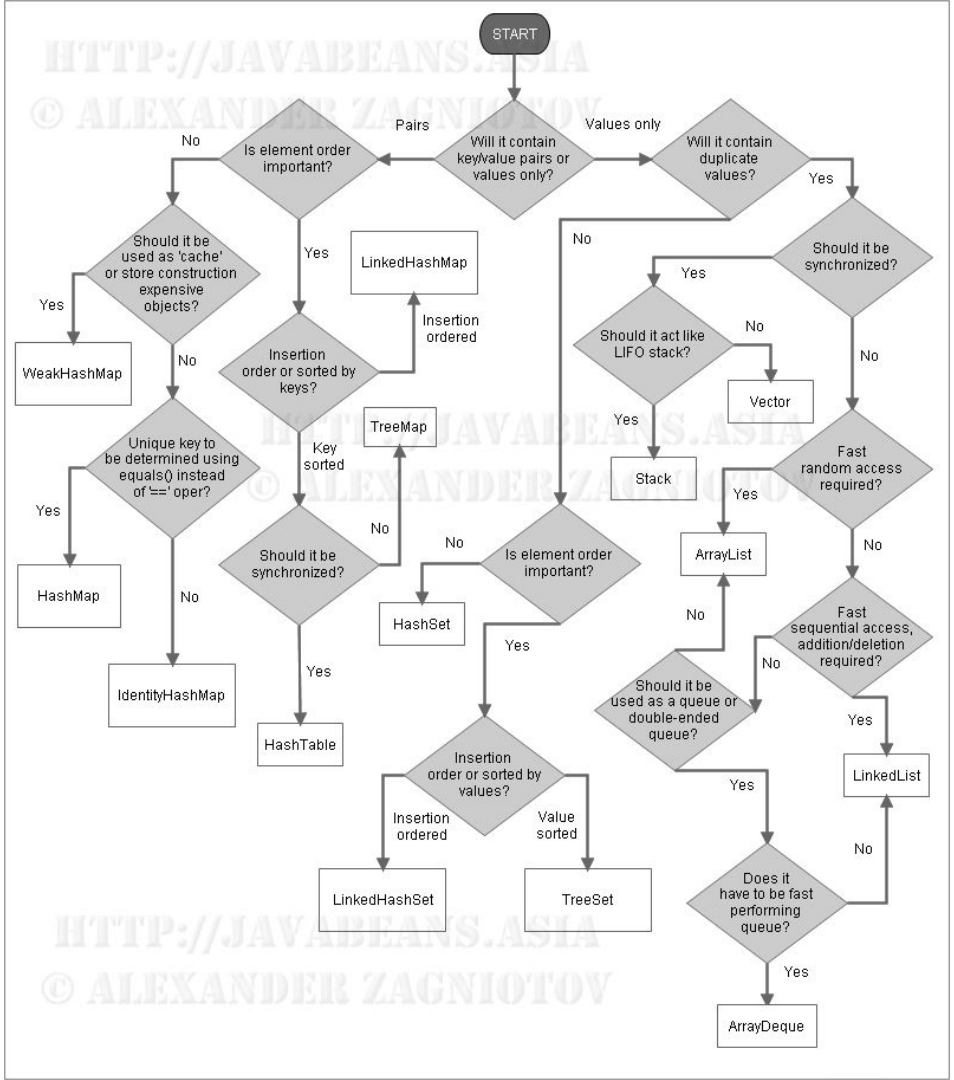
ArrayDeque — реализация интерфейса Deque, который расширяет интерфейс Queue методами, позволяющими реализовать конструкцию вида LIFO (last-in-first-out). Интерфейс Deque и реализация ArrayDeque были добавлены в Java 1.6. Эта коллекция представляет собой реализацию с использованием массивов, подобно ArrayList, но не позволяет обращаться к элементам по индексу и хранение null. Как заявлено в документации, коллекция работает быстрее чем Stack, если используется как LIFO коллекция, а также быстрее чем LinkedList, если используется как FIFO.

Выбор коллекций

Просто о сложном

- Необходимо хранить множество объектов и иметь возможность всегда достать какой-то элемент оттуда? (Например список Posts у Person)
  - ArrayList
- Необходимо хранить множество объектов, где порядок не имеет значение? (Например Tags у Post)
  - HashSet
- Необходимо хранить множество объектов в виде ключ-значение? (Например Settings)
  - HashMap

Сложно о сложном



Сложно о сложном

	Ordering	Random Access	Key-Value Pairs	Allows Duplicates	Allows Null Values	Thread Safe	Blocking Operations	Upper Bounds
Most Commonly Known Collections								
ArrayList	YES	YES	NO	YES	YES	NO	NO	NO
HashMap	NO	YES	YES	NO	YES	NO	NO	NO
Vector	YES	YES	NO	YES	YES	YES	NO	NO
Hashtable	NO	YES	YES	NO	NO	YES	NO	NO

Сложно о сложном

	Ordering	Random Access	Key-Value Pairs	Allows Duplicates	Allows Null Values	Thread Safe	Blocking Operations	Upper Bounds
Most Talked About Collections								
HashSet	NO	YES	NO	NO	YES	NO	NO	NO
TreeSet	YES	YES	NO	NO	NO	NO	NO	NO
LinkedList	YES	NO	NO	YES	YES	NO	NO	NO
ArrayDeque	YES	YES	NO	YES	NO	NO	NO	NO
Stack	YES	NO	NO	YES	YES	YES	NO	NO
TreeMap	YES	YES	YES	NO	NO	NO	NO	NO

Сложно о сложном

	Ordering	Random Access	Key-Value Pairs	Allows Duplicates	Allows Null Values	Thread Safe	Blocking Operations	Upper Bounds
Special Purpose Collections								
WeakHashMap	NO	YES	YES	NO	YES	NO	NO	NO
Arrays	YES	YES	NO	YES	YES	NO	NO	YES
Properties	NO	YES	YES	NO	NO	YES	NO	NO

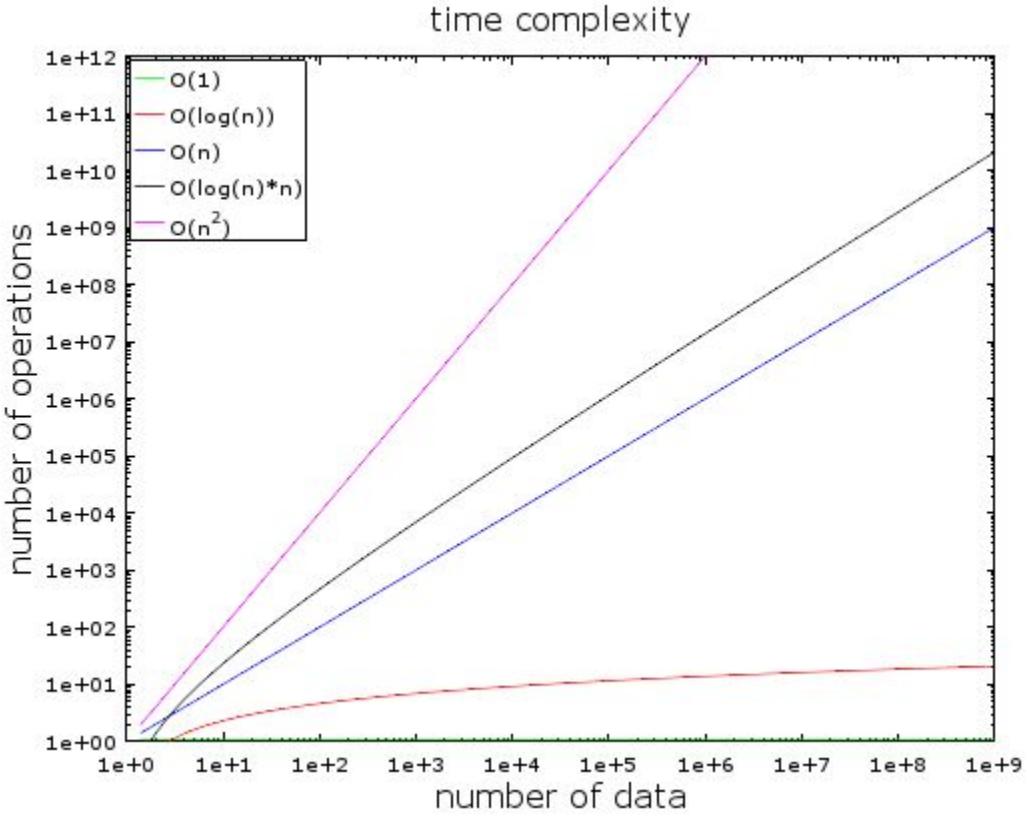
Сложно о сложном

	Ordering	Random Access	Key-Value Pairs	Allows Duplicates	Allows Null Values	Thread Safe	Blocking Operations	Upper Bounds
Thread Safe Collections								
CopyOnWriteArrayList	YES	YES	NO	YES	YES	YES	NO	NO
ConcurrentHashMap	NO	YES	YES	NO	NO	YES	NO	NO
ConcurrentSkipListMap	YES	YES	YES	NO	NO	YES	NO	NO
ConcurrentSkipListSet	YES	NO	NO	NO	NO	YES	NO	NO
CopyOnWriteArraySet	YES	YES	NO	NO	YES	YES	NO	NO
ConcurrentLinkedQueue	YES	NO	NO	YES	NO	YES	NO	NO
ConcurrentLinkedDeque	YES	NO	NO	YES	NO	YES	NO	NO

Сложно о сложном

	Ordering	Random Access	Key-Value Pairs	Allows Duplicates	Allows Null Values	Thread Safe	Blocking Operations	Upper Bounds
Blocking Collections								
ArrayBlockingQueue	YES	NO	NO	YES	NO	YES	YES	YES
LinkedBlockingQueue	YES	NO	NO	YES	NO	YES	YES	YES
LinkedTransferQueue	YES	NO	NO	YES	NO	YES	YES	YES
PriorityBlockingQueue	YES	NO	NO	YES	NO	YES	YES	NO
LinkedBlockingDeque	YES	NO	NO	YES	NO	YES	YES	YES
SynchronousQueue	YES	NO	NO	YES	NO	YES	YES	NO
DelayQueue	YES	NO	NO	YES	NO	YES	YES	NO

Временная сложность операций



Временная сложность операций

	Временная сложность							
	Среднее				*Худшее*			
	Индекс	Поиск	Вставка	Удаление	Индекс	Поиск	Вставка	Удаление
ArrayList	O(1)	O(n)	O(n)	O(n)	O(1)	O(n)	O(n)	O(n)
Vector	O(1)	O(n)	O(n)	O(n)	O(1)	O(n)	O(n)	O(n)
LinkedList	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)
Hashtable	n/a	O(1)	O(1)	O(1)	n/a	O(n)	O(n)	O(n)
HashMap	n/a	O(1)	O(1)	O(1)	n/a	O(n)	O(n)	O(n)
LinkedHashMap	n/a	O(1)	O(1)	O(1)	n/a	O(n)	O(n)	O(n)
TreeMap	n/a	O(log(n))	O(log(n))	O(log(n))	n/a	O(log(n))	O(log(n))	O(log(n))
HashSet	n/a	O(1)	O(1)	O(1)	n/a	O(n)	O(n)	O(n)
LinkedHashSet	n/a	O(1)	O(1)	O(1)	n/a	O(n)	O(n)	O(n)
TreeSet	n/a	O(log(n))	O(log(n))	O(log(n))	n/a	O(log(n))	O(log(n))	O(log(n))

Сортировка

Сортировка

```
Collections.sort(List<T extends Comparable> list);

Collections.sort(List<T> list, Comparator comparator);
```

Comparable

```
public class … implements Comparable<...> {
…
    @Override
    public int compareTo(... o) {
      // Should return 0 if this equals to o
      // Should return number < 0 if this smaller than o
      // Should return number > 0 if this bigger than o
      return ...;
    }
…
}
```

Comparator

```
public class ...Comparator implements Comparator<...> {
  @Override
  public int compare(... o1, ... o2) {
      // Should return 0 if o1 equals to o2
      // Should return number < 0 if o1 smaller than o2
      // Should return number > 0 if o1 bigger than o2
     return ...;
  }
}
```

Вложенные и анонимные классы

Вложенный класс

https://github.com/TheAvalanche/akurss-java-1/blob/master/src/lv/akurss/lesson9/lecture/comparator_inner_class/

Анонимный класс

https://github.com/TheAvalanche/akurss-java-1/tree/master/src/lv/akurss/lesson9/lecture/comparator_anonymous_class