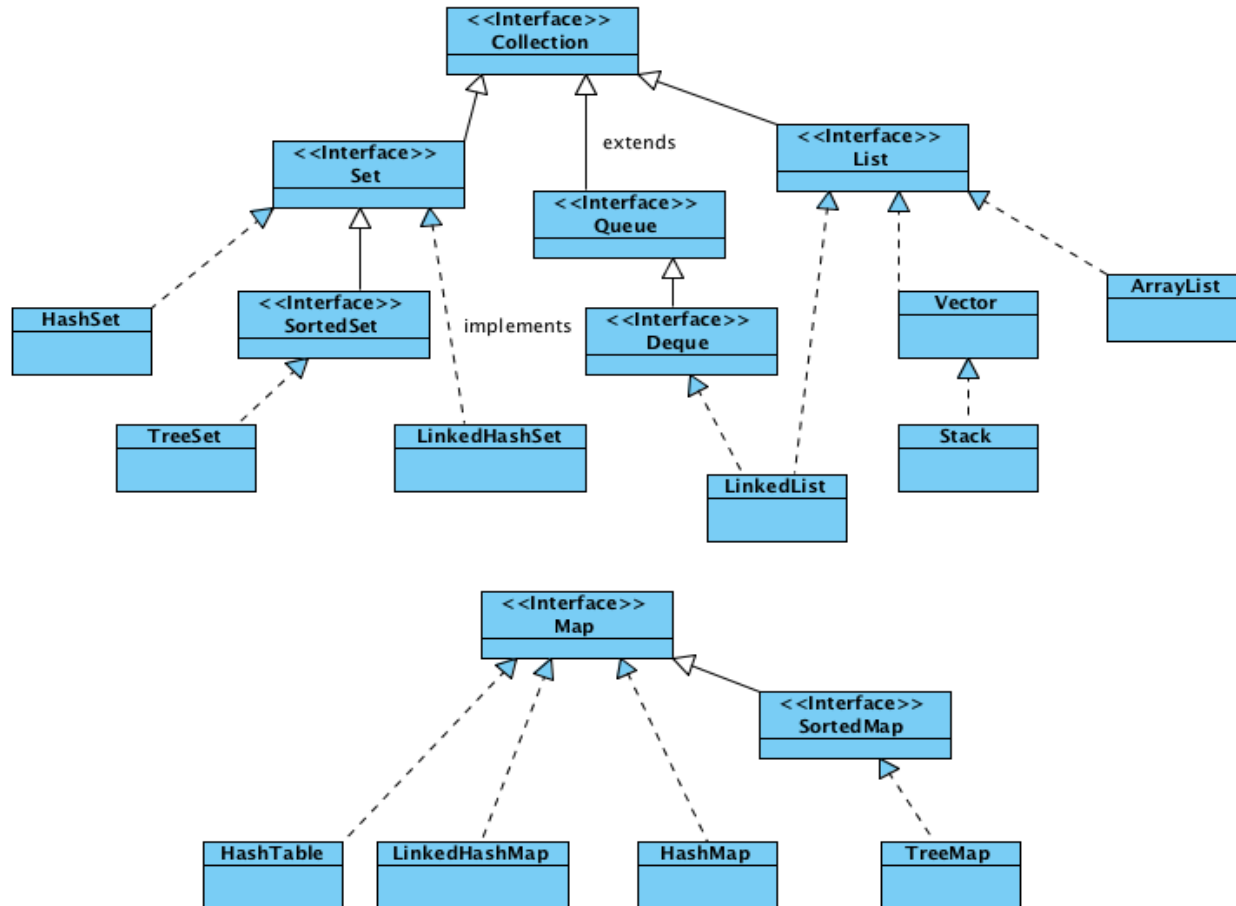


Collections framework

1. Collections Framework
2. Иерархия коллекций
3. Интерфейс List
4. Интерфейс Set
5. Интерфейс Map

Collections Framework

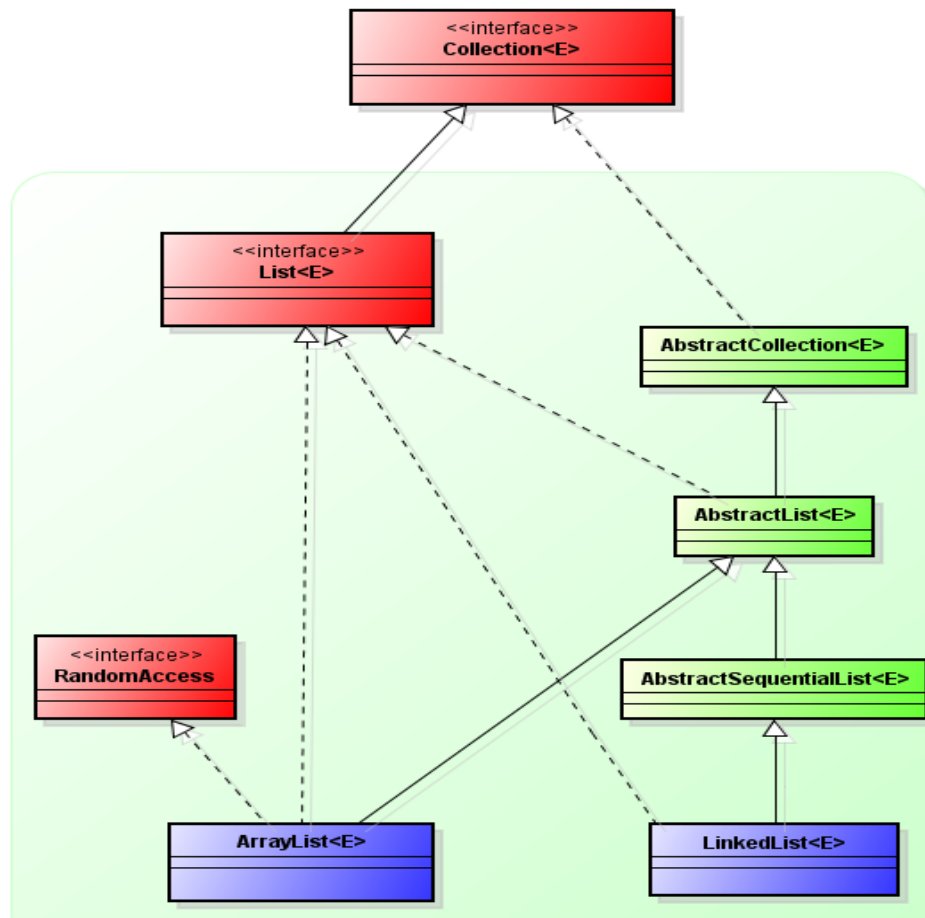


Интерфейс Collection расширяют интерфейсы **List**, **Set** и **Queue**

Интерфейс **Map** не расширяет интерфейс Collection, но считается частью Collection Framework.

Интерфейс List сохраняет последовательность добавления элементов и позволяет осуществлять доступ к элементу по индексу.

Интерфейс List



Класс ArrayList реализует интерфейс List. ArrayList поддерживает динамические массивы, которые могут расти по мере необходимости.

Достоинства класса ArrayList:

- Быстрый доступ по индексу.
- Быстрая вставка и удаление элементов с конца.
- Элементы могут быть любых типов в том числе и null.

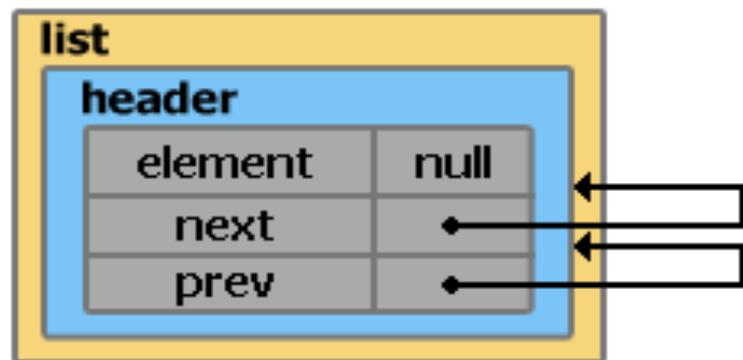
Добавление элемента на позицию с определенным индексом:

1. проверяется, достаточно ли места в массиве для вставки нового элемента;
`ensureCapacity(size+1);`
2. подготавливается место для нового элемента;
`System.arraycopy(elementData, index, elementData, index + 1, size - index);`
3. перезаписывается значение у элемента с указанным индексом.
`elementData[index] = element;`
`size++;`

Если места в массиве не достаточно, новая емкость рассчитывается по формуле $(oldCapacity * 3) / 2 + 1$.

LinkedList расширяет класс `AbstractSequentialList<E>` и реализует интерфейсы `List<E>`, `Deque<E>`. Он предоставляет структуру данных связного списка.

Класс LinkedList



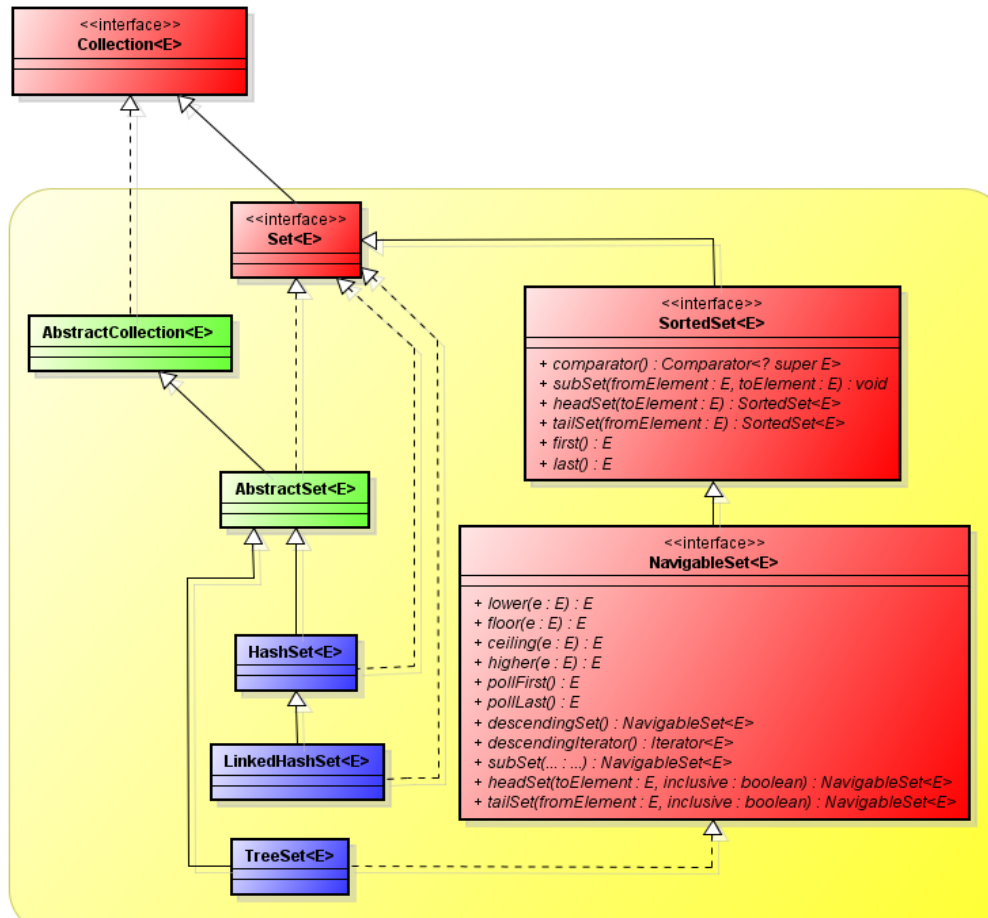
ListIterator расширяет интерфейс Iterator для двустороннего обхода списка и видоизменения его элементов.

```
ListIterator<String> itr = list.listIterator();
```

```
while ( itr.hasNext() )
```

```
    System.out.println(itr.next());
```

Интерфейс Set



В API Java Collections есть следующие реализации интерфейса Set:

- `java.util.EnumSet;`
- `java.util.HashSet;`
- `java.util.LinkedHashSet;`
- `java.util.TreeSet.`

Интерфейс SortedSet расширяет интерфейс Set и определяет поведение множеств, отсортированных в порядке возрастания.

```
interface SortedSet<E>
```

NavigableSet расширяет интерфейс SortedSet и определяет поведение коллекции, извлечение элементов из которой осуществляется на основании наиболее точного совпадения с заданным значением или несколькими значениями.

interface NavigableSet<E>

Класс HashSet расширяет класс AbstractSet и реализует интерфейс Set. Он служит для создания коллекции, для хранения элементов которой используется хеш-таблица.

```
class HashSet<E>
```

Хеш-таблица представляет такую структуру данных, в которой все объекты имеют уникальный ключ или хеш-код. Данный ключ позволяет уникально идентифицировать объект в таблице.

Преимущество хеширования заключается в том, что оно обеспечивает постоянство времени выполнения методов `add()`, `contains()`, `remove()` и `size()` - даже для крупных множеств.

Работа с HashSet:

```
Set<String> daysOfWeek = new HashSet<>();
```

```
daysOfWeek.add("Monday");  
daysOfWeek.add("Tuesday");  
daysOfWeek.add("Wednesday");  
daysOfWeek.add("Thursday");  
daysOfWeek.add("Friday");  
daysOfWeek.add("Saturday");  
daysOfWeek.add("Sunday");
```

```
// Adding duplicate elements will be ignored  
daysOfWeek.add("Monday");
```

```
System.out.println(daysOfWeek);
```

Результат работы:

[Monday, Thursday, Friday, Sunday, Wednesday, Tuesday, Saturday]

HashSet не гарантирует порядок элементов.

Класс LinkedHashSet расширяет класс HashSet

```
class LinkedHashSet<E>
```

LinkedHashSet поддерживается связный список элементов хеш-множества в том порядке, в каком они введены в него

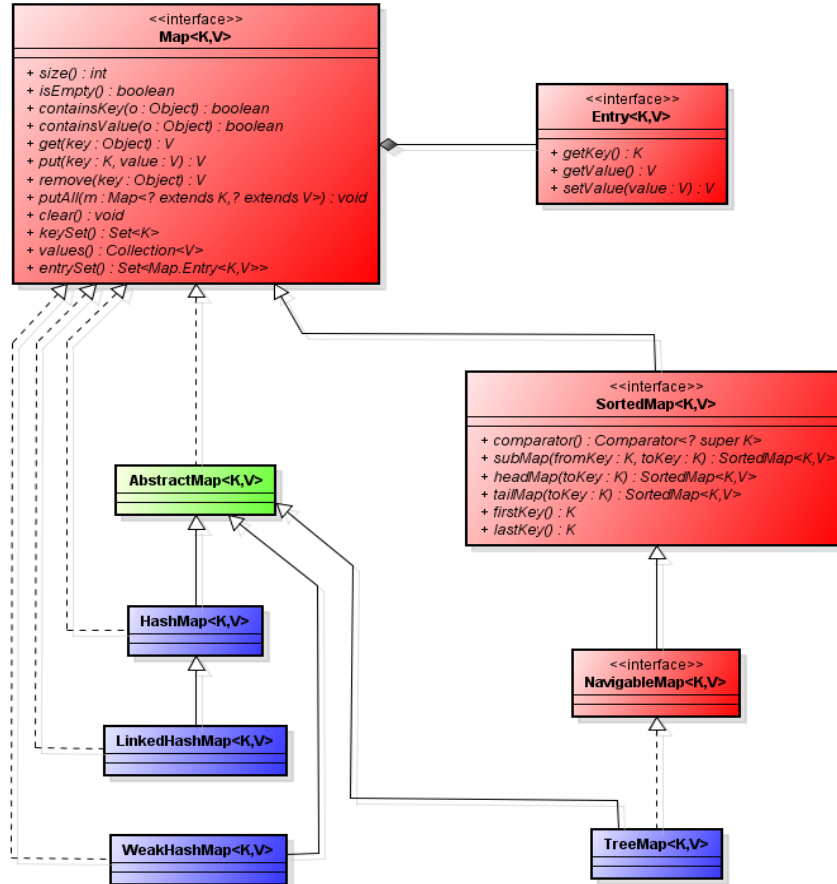
Класс `LinkedHashSet` расширяет класс `HashSet`

class `LinkedHashSet<E>`

`LinkedHashSet` поддерживается связный список элементов хеш-множества в том порядке, в каком они введены в него

Интерфейс $\text{Map}\langle K, V \rangle$ представляет отображение или иначе говоря словарь, где каждый элемент представляет пару "ключ-значение". При этом все ключи уникальные в рамках объекта Map.

Интерфейс Map



Конструкторы класса HashMap:

- `HashMap()` - значение начальной емкости (16) и коэффициентом загрузки (0.75)
- `HashMap(int initialCapacity)` - коэффициент загрузки (0.75) и начальная емкостью `initialCapacity`
- `HashMap(int initialCapacity, float loadFactor)` – начальная емкости `initialCapacity` и коэффициентом загрузки `loadFactor`
- `HashMap(Map<? extends K,? extends V> m)` - Конструктор с определением структуры согласно объекту-параметра.

Хэширование - это процесс преобразования объекта в целочисленную форму, выполняется с помощью метода `hashCode()`.

Хеширование

```
class Key {  
    String key;  
    Key(String key) {  
        this.key = key;  
    }  
  
    @Override  
    public int hashCode() {  
        return (int)key.charAt(0);  
    }  
  
    @Override  
    public boolean equals(Object obj) {  
        return key.equals((String)obj);  
    }  
}
```

Внутреннее устройство HashMap

Bucket - это единственный элемент массива HashMap. Он используется для хранения узлов (Nodes).

Отношение между bucket и capacity:

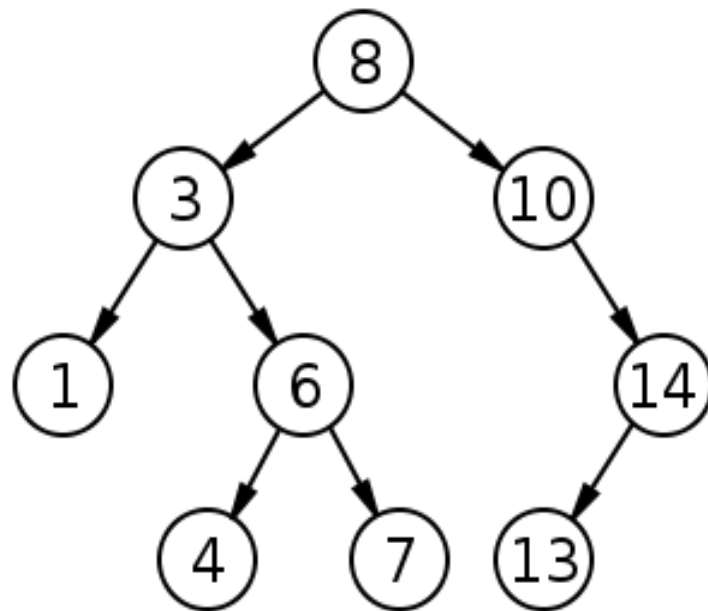
$\text{capacity} = \text{number of buckets} * \text{load factor}$

Изначально HashMap пустой и размер его равен 16:

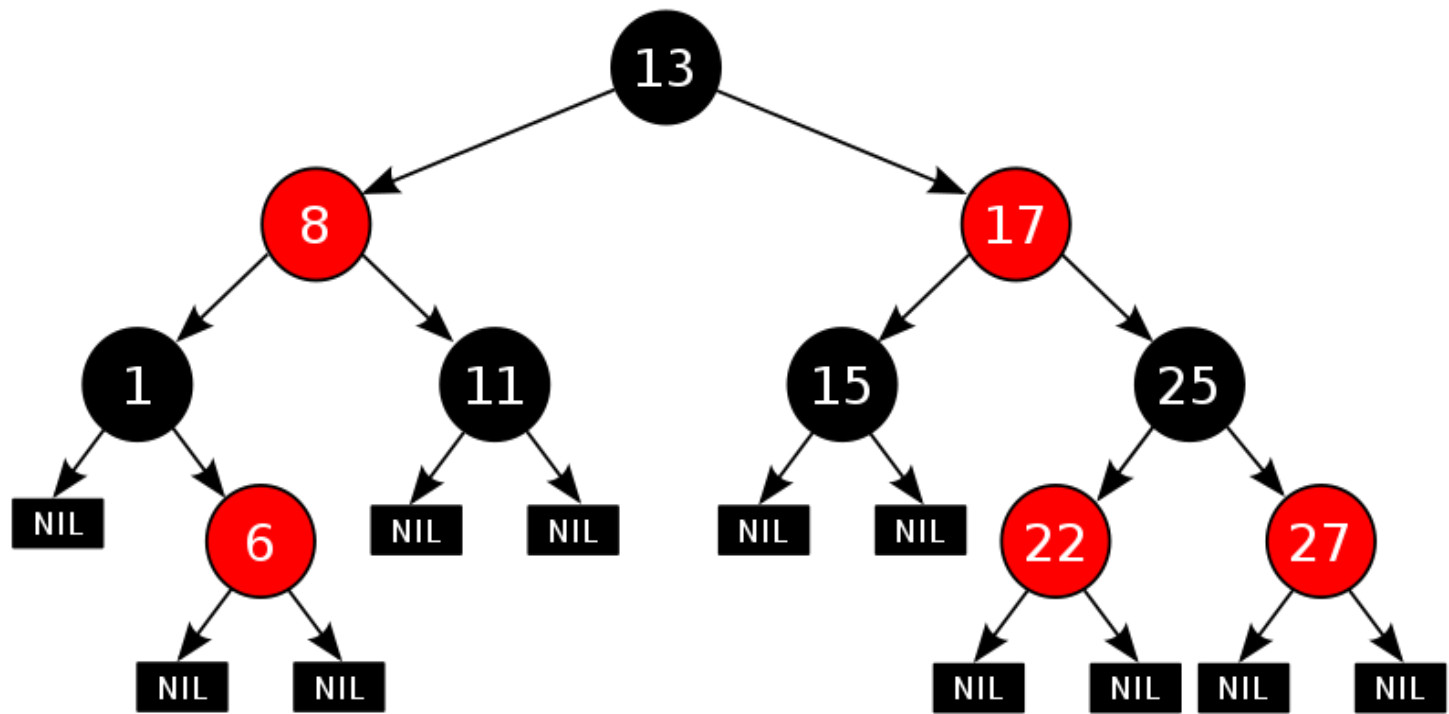
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
---	---	---	---	---	---	---	---	---	---	----	----	----	----	----	----

Класс TreeMap расширяет класс AbstractMap и реализует интерфейс NavigableMap. Он создает коллекцию, которая для хранения элементов применяет дерево. Объекты сохраняются в отсортированном порядке по возрастанию.

Двоичное дерево поиска



Красно-черное дерево



Сложность алгоритмов

	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	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$

Вопросы для самоконтроля

1. В чем отличие Collection от массивов?
2. Я добавляю элемент в середину большого списка. На какой коллекции это будет быстрее- ArrayList или LinkedList?
3. Как производится балансировка в красно-черном дереве?
4. Какова алгоритмическая сложность поиска в хеш-коллекции?
5. Как возможно потерять элемент в hashMap?