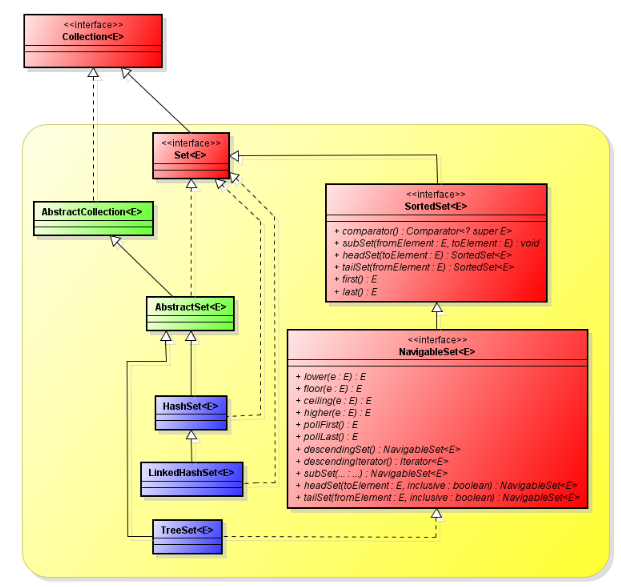
**SET**

(МНОЖЕСТВО)



Set этонеупорядоченное множество, которое **не позволяет хранить дубликаты**. Множества чаще всего используют для проверки принадлежности, принадлежит ли объект заданному множеству (на практике выбирают HashSet – которая оптимизированная для быстрого поиска).

**HashSet**

Данная реализация использует хэш-функцию для эффективного размещения объектов заносимых в коллекцию. Классы заносимые в коллекцию должны переопределить метод HashCode. Уникальность обеспечивается благодаря двум методам: equals() and hashCode()! После того как определили hashCode обьекта, HashSet прыгает в указанную корзинку и проверяет там обьект по equals.

**LinkedHashSet**

Эта реализация Set-та наследуется от HashSet и новых методов не добавляет. Элементы хранит в порядке добавления. При повторном добавлении элемента порядок не меняется. Используем когда нужен порядок и быстрый доступ.

**TreeSet**

Хранение элементов по принципу деревьев. Уникальность обеспечивается только с помощью метода compare(o1, o2)! По этому когда пишем принцип по которому будем сортировать treeSet нужно быть аккуратным:

Set<Integer> set = new TreeSet<>(new Comparator<Integer>() {  
 @Override  
 public int compare(Integer o1, Integer o2) {  
 return 0;  
 }  
});  
  
set.add(1);  
set.add(2);  
set.add(1);  
  
System.*out*.println(set); //result [1]

Благодаря тому, что TreeSet имплементирует интерфейс NavigableSet, появляются дополнительные возможности навигации по дереву отсортированному(можем получить элементы после/до числа, вернуть элементы с конца дерева).

**2)Почему хешкод умножается на 31 ? почему именно на 31 ? Могу умножить на другое число ?**

Значение 31 потому что это нечётное простое число. Если сделать его чётным и умножение было переполнено, информацию потеряли бы, так как умножение на 2 эквивалентно сдвигу.

Преимущество использования прайма менее очевидно,но оно традиционно. Хорошим свойством 31 является то что умножение может быть заменено сдвигом и вычитанием для лучшей производительности.

**3) Что будет выведено в консоль ? и объяснить почему так**

**без IDEA**

public void lalala(View view) {

Random random = new Random(30);

Set<Integer> numberSet = new HashSet<>();

for (int i = 0; i < 20; i++)

numberSet.add(random.nextInt(10));

Syste,out.println(numberSet.toString());

}

Будут выведены числа от 0 до 9 включительно.

**4) Как получить элемент по его индексу из HashSet или TreeSet ?**Реализации на основе интерфейса Set - не индексируются.

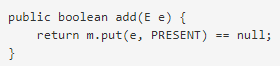
**5) Как реализован TreeSet ? За счет чего реализовано его структура хранения элементов ?**

TreeSet построен на бинарном дереве. Внутри работает вместе с NavigableMap который инициализируется с экземляром TreeMap при создании обьекта TreeSet.

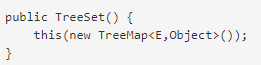
Хорошо описано в <https://www.codeflow.site/ru/article/java-tree-set>

Метод add чрезвычайно важен, так как детали реализации метода иллюстрируют внутреннюю работу TreeSet \*\* , как он использует метод TreeMap’s put для хранения элементов. Переменная m ссылается на внутреннюю основу TreeMap





Следовательно, TreeSet внутренне зависит от вспомогательного NavigableMap , который инициализируется с экземпляром TreeMap при создании экземпляра TreeSet :



**6) Как реализован HashSet ?**

Реализован на хэш-таблице, в качестве ключа используется значение (сам обьект добавляемый).

Класс предлагает постоянную производительность времени для основных операций (добавление, удаление, наличие и размер).

На самом деле значение, которые мы передаем в HashSet, является ключом к объекту HashMap, а в качестве значения в HashMap используется константа. Таким образом, в каждой паре «ключ-значение» все ключи будут иметь одинаковые значения.

**7) Основные методы HashSet**

add() – добавляет элемент в коллекцию \ есть похожий addAll() – добавляет группу обьектов,

contains() – проверяет есть ли элемент в коллекции,

isEmpty() – пустая коллекция или нет? ,

iterator() – возвращает ссылку на итератор,

size() – получаем размер коллекции,

remove() – удаляем указанный элемент из коллекции

**8) без IDEA, что будет выведено в консоль ?**

public void test() {

Set<String> countryHashSet = new HashSet<>();

countryHashSet.add("Россия");

countryHashSet.add("Болгария");

countryHashSet.add("Франция");

countryHashSet.add("Россия");

countryHashSet.add("Гондурас");

countryHashSet.add("Болгария");

System.out.println(countryHashSet.size());

}

Размер коллекции countryHashSet = 4

**9) Время вставки в начало, середину, конец для HashSet**

Время вставки во всех случаях = О(1)

**10)Время вставки в начало, середину, конец для TreeSet**

Время добавления = O(log n)

Гарантирует log (n) временную стоимость для основных операций (добавление, удаление и содержит)

**11)Время вставки в начало, середину, конец для LinkedHashSet**

Время будет О(n)

**12)При помощи чего в TreeSet можно производить сортировку ?**

Можно с помощью Comparator переданного в конструктор TreeSet и при добавлении элементов – будет производиться сортировка с использованием переданного Comparator.

**13) Сложность поиска элемента в HashSet, TreeSet, LinkedHashSet**

В **HashSet** - нельзя просто получить элемент по индексу или по значению, нету таких методов, можно проверить присутствует ли данный обьект в массиве. Для поиска элемента в данной коллекции, нужно получить обьект Iterator и работать дальше с ним.

В **LinkedHashSet** – анологично.

В **TreeSet** – имеются методы которые возвращают последний элемент или первый эллемент в данном наборе, но опять же прямого доступа к середине не имеем – только через Iterator

**14) Что такое HashSet ?**

Неупорядоченное множество элементов, не позволяет хранить дубликаты.

Хранит элементы в произвольном порядке, но зато быстро ищет. Подходит, если порядок Вам не важен, но важна скорость. Более того, для оптимизации поиска, **HashSet**будет хранить элементы так, как ему удобно.

**15) Что такое TreeSet ?**

Основан на бинарном дереве, сортирует элементы по возрастанию\по алфовиту или по порядку указанному в Comparator

**16) Что такое LinkedHashSet ?-+**

Расчиряет HashSet элементы хранятся в порядке их добавления. Элемент имеет ссыку на пердыдущий и следующий элемент.

**17) \* Что быстрее: TreeSet или LinkedHashSet ?**

Я бы позадавал дополнительные вопросы, быстрее в чём конкретно, в добавлении? Поиске ?

**Заметки:**

* **HashSet** не допускает двух одинаковых элементов, null допускается.
* **TreeSet** больше не поддерживает добавление **null** . Когда мы добавляем элементы в TreeSet, элементы сортируются в соответствии с их естественным порядком или в соответствии с указаниями comparator. Следовательно, добавление null, по сравнению с существующими элементами, приводит к **NullPointerException** , поскольку null нельзя сравнивать с любым значением.