Реализации интерфейса Set, Queue

Java Collections
6 уровень, 9 лекция

ОТКРЫТА

— Ну как твой процессор?

— Норм. Посидел час в жидком азоте и как новенький!

— Отлично. Тогда давай продолжим.

Коллекции Set.

Set переводится как множество. А множество, с математической точки зрения, — это набор уникальных элементов. Но т.к. не все программисты – математики, то обычно говорят, что Set – это коллекция уникальных элементов, или коллекция, которая не позволяет хранить одинаковые элементы.

Не знаю, давала Элли тебе иерархию наследования для Set, но если нет, то вот она:

HashSet – это коллекция, которая для хранения элементов внутри использует их хэш-значения, которые возвращает метод **hashCode**().

Для простоты внутри HashSet<E> хранится объект HashMap<E, Object>, который и хранит в качестве ключей значения HashSet.

— Ничего себе!

— Использование hash-кодов позволяет довольно быстро искать, добавлять и удалять элементы из множества (Set).

Но учти, чтобы объекты твоих классов можно было класть в Set и правильно находить их там, у твоего класса должны быть правильно реализованы методы **hashCode & equals.**

И тот и другой активно используются внутри **HashSet/HashMap.**

Если ты забудешь реализовать метод **hashCode**(), то рискуешь, что твой объект в коллекции Set не будет найден, даже если он там есть.

— Да, помню, я помню. Ты мне уже рассказывал раньше об этом. Все робоуши прожужжал.

— Ок. Тогда вот тебе еще полезная информация.

Допустим, ты правильно реализовал **hashCode и equals** в своем классе и такой весь радостный хранишь объекты в Set’е.

Но потом ты взял и поменял один из объектов, при этом поменялись его внутренние данные, которые используются в вычислении **хэша**. И хэш объекта стал другим.

А это значит, что когда ты будешь его искать в Set’е, его скорее всего не найдут.

— Ничего себе! Это как же?

— Это всем известный косяк с работой хешей. Грубо говоря, поиск в HashSet (и в HashMap) гарантированно работает правильно, только если объекты – **immutable**.

— Ничего себе! И что, никто ничего не делает?

НАЧАТЬ ОБУЧЕНИЕ

— Все делают вид, что проблемы не существует. Но на собеседованиях это частенько спрашивают, так что возможно, тебе стоит запомнить этот факт…

LinkedHashSet – это HashSet, в котором элементы хранятся еще и в связном списке. Обычный HashSet не поддерживает порядок элементов. Во-первых, официально его просто нет, во-вторых, даже внутренний порядок может сильно поменяться при добавлении всего одного элемента.

А у **LinkedHashSet** можно получить итератор и с его помощью обойти все элементы именно в том порядке, в котором они добавлялись в **LinkedHashSet**. Не часто, но иногда это может очень понадобится.

— Ясно. Люблю, когда у классов есть такие «на всякий случай» разновидности. Обычно такие случаи наступают не так уж и редко.

— **TreeSet** – это коллекция, которая хранит элементы в виде упорядоченного по значениям дерева. Внутри **TreeSet**<E> содержится **TreeMap**<E, Object> который и хранит все эти значения. А этот **TreeMap** использует **красно-черное сбалансированное бинарное дерево** для хранения элементов. Поэтому у него очень быстрые операции **add, remove, contains.**

— Ага. Я помню, мы же совсем недавно это разбирали. А я еще думал – и где это применяется.

А оказывается, одни из самых популярных коллекций в Java используют это.

— Ага, кстати, на собеседованиях часто спрашивают про TreeSet. Обычно стараются подловить. Мол, если в TreeSet используется бинарное дерево, то тогда все элементы могут образовывать одну длинную ветку и при этом поиск будет очень долгим. Тут, кстати, стоит поставить такого наглеца на место, и заявить, что даже ребенку известно, что TreeSet и TreeMap используют сбалансированные **красно-черные** бинарные деревья, и поэтому такая ситуация невозможна в принципе.

— Ага. Хотелось бы увидеть лицо спрашивающего в этот момент. Я, пожалуй, даже заучу эту фразу. …

А в принципе, Set оказался не таким уж и простым, как мне казалось в самом начале.

— Зато с **Queue** ситуация гораздо проще:

Queue – это очередь. Элементы добавляются в конец очереди, а выбираются из ее начала.

PriorityQueue – это фактически единственная классическая реализация интерфейса **Queue**, не считая **LinkedList**, который формально тоже является очередью.

Ладно, что-то я устал, на сегодня все. Давай до свидания.

Предыдущая лекция