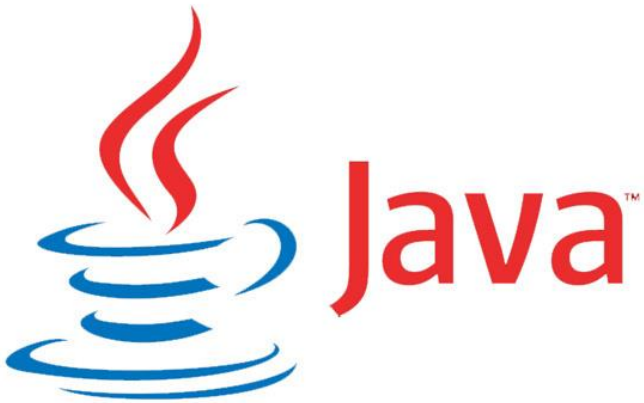
Тема 12.
Коллекции (часть 2)



Коллекции

Коллекция - это объект, способный хранить группу одинаковых элементов. Она содержит большое количество готовых методов для работы с однородными данными.

Интерфейс **Collection** является родительским интерфейсом **Set**, **Queue** и **List**. Интерфейс Collection определяет множество методов, которые могут быть реализованы его подклассами для реализации операций с данными.

Интерфейс Set

Интерфейс Set расширяет **интерфейс Collection** и представляет набор уникальных элементов. Set не добавляет новых методов, только вносит изменения в унаследованные. В частности, метод **add()** добавляет элемент в коллекцию и возвращает true, если в коллекции еще нет такого элемента.

класс HashSet

Обобщенный класс **HashSet** представляет **хеш-таблицу**.

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

Для корректной работы этого класса необходимы некоторые условия:

- переопределить метод **equals()**
- переопределить метод **hashCode()**

метод equals()

equals() - метод определения равенства двух объектов.

Обычное сравнение двух объектов через оператор “==” это плохая идея, потому что “==” сравнивает ссылки.

Метод **equlas()** необходимо переопределять с учетом специфики вашего класса. Основная идея — создатель класса сам определяет характеристики, по которым проверяется равенство объектов этого класса.

Самый наглядный пример это сравнение значений всех полей двух объектов. Конечно же это будет очень слабая реализация, и в реальных проектах надо проверить еще ряд вещей, но для наглядности пока так.

метод hashCode()

Теперь поговорим о методе **hashCode()**. Зачем он нужен?

Ровно для той же цели — сравнения объектов. Но ведь у нас уже есть **equals()** метод. Зачем же еще один метод?

Ответ прост: **для повышения производительности**.

Хэш-функция, которая представлена в Java методом **hashCode()**, возвращает числовое значение фиксированной длины для любого объекта. В случае с Java метод **hashCode()** возвращает для любого объекта 32-битное число типа int.

Сравнить два числа между собой — гораздо быстрее, чем сравнить два объекта методом **equals()**, особенно если в нем используется много полей.

Понятие коллизии

Ситуация, когда у разных объектов одинаковые хеш-коды называется — **коллизией**. Вероятность возникновения коллизии зависит от используемого алгоритма генерации хеш-кода.

Причина: максимальное количество вариантов хеш-кода ограничено пределами типа int. А количество объектов в программе ограничено лишь нашей фантазией.

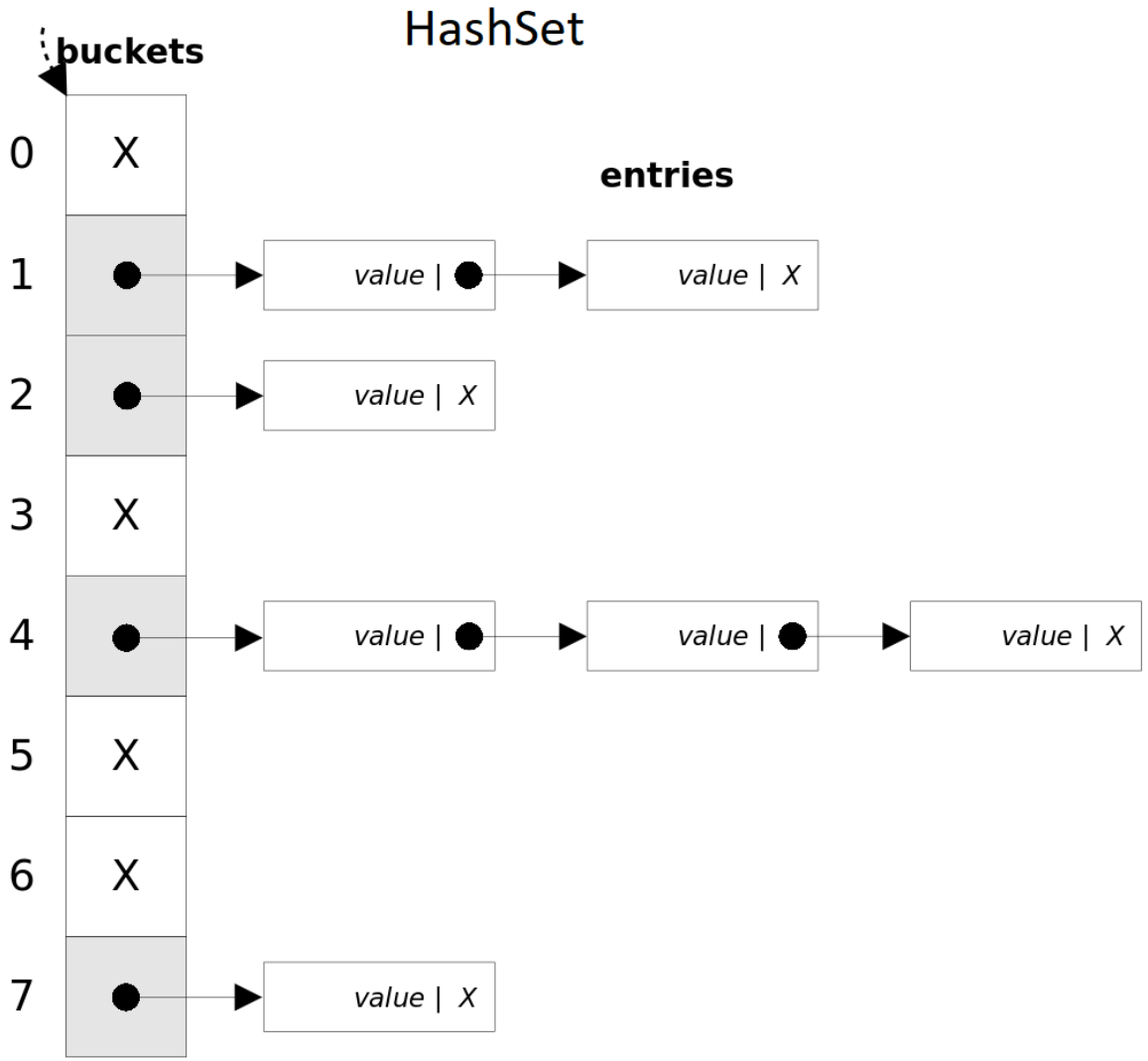
В связи с этим надо понимать следующие правила:

- Если хеш-коды разные, то и входные объекты гарантированно разные.
- Если хеш-коды равны, то входные объекты не всегда равны

класс HashSet

HashSet внутри состоит с так называемых корзин и списка элементов, на которые ссылаются корзины.

По сути корзины это обычный массив, каждым элементом которого является двусвязный список.



класс HashSet

Механизм добавления

Когда приходит новый элемент, хэш-код ключа определяет корзину, для элемента. В корзине идет проверка, есть ли у нее элементы. Если нету, то корзина получает ссылку нового элемента, если есть, то происходит прохождение по списку элементов и сравнивание элементов в списке. Проверяется равенство hashcode. Так как мы не можем однозначно судить на счет эквивалентности, зная о случаях коллизии, проводится еще сравнивание ключей методом equals.

- Если оба равны: идет перезапись элемента
- Если не равен equals: добавляется элемент в список

класс HashSet

Пример

```
HashSet<String> h = new HashSet<String>();

// Добавляем элементы в HashSet с помощью метода add()
h.add("India");
h.add("Australia");
h.add("South Africa");
h.add("India");// пытаемся добавить еще один такой же элемент

// Выводим элементы HashSet в консоль
System.out.println(h);
System.out.println("List contains India or not:" +
                   h.contains("India"));

// Удаляем элементы из множества с помощью метода remove()
h.remove("Australia");
System.out.println("List after removing Australia:"+h);
```

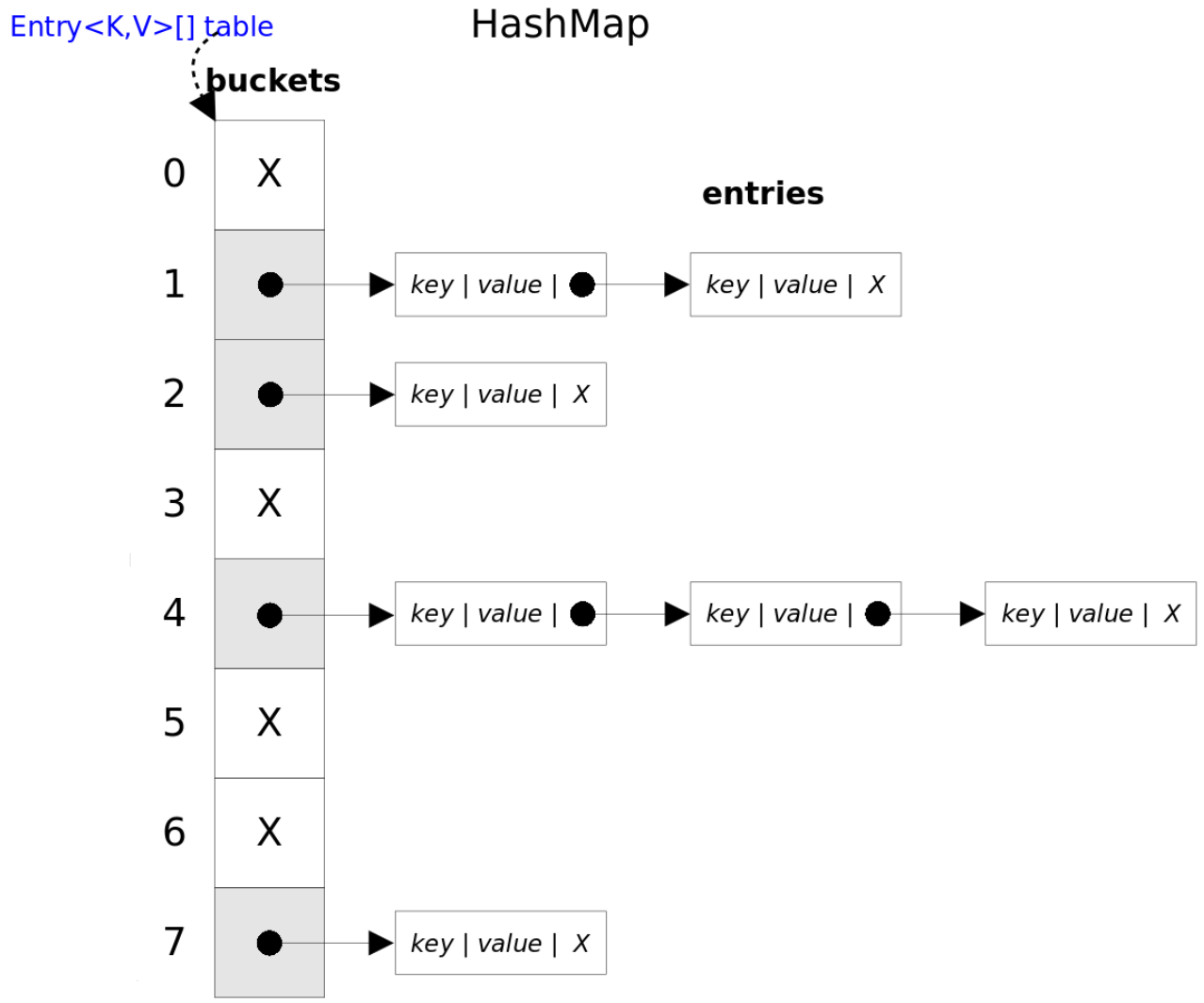
интерфейс Map

Интерфейс Map<K, V> представляет отображение или иначе говоря словарь, где каждый элемент представляет пару "ключ-значение". При этом все ключи уникальные в рамках объекта Map. Такие коллекции облегчают поиск элемента, если нам известен ключ - уникальный идентификатор объекта.

интерфейс Map

HashMap очень похож на HashSet. Разница лишь в том, что каждый элемент здесь является парой <ключ, значение>.

А в **HashSet**-е каждый элемент - только значение.



интерфейс Map

Среди методов интерфейса Map можно выделить следующие:

- **void clear():** очищает коллекцию
- **boolean containsKey(Object k):** возвращает true, если коллекция содержит ключ k
- **boolean containsValue(Object v):** возвращает true, если коллекция содержит значение v
- **Set<Map.Entry<K, V>> entrySet():** возвращает набор элементов коллекции. Все элементы представляют объект Map.Entry
- **boolean equals(Object obj):** возвращает true, если коллекция идентична коллекции, передаваемой через параметр obj
- **boolean isEmpty():** возвращает true, если коллекция пуста
- **V get(Object k):** возвращает значение объекта, ключ которого равен k. Если такого элемента не окажется, то возвращается значение null

интерфейс Map

- **V getOrDefault(Object k, V defaultValue):** возвращает значение объекта, ключ которого равен k. Если такого элемента не окажется, то возвращается значение defaultVlue
- **V put(K k, V v):** помещает в коллекцию новый объект с ключом k и значением v. Если в коллекции уже есть объект с подобным ключом, то он перезаписывается. После добавления возвращает предыдущее значение для ключа k, если он уже был в коллекции. Если же ключа еще не было в коллекции, то возвращается значение null
- **V putIfAbsent(K k, V v):** помещает в коллекцию новый объект с ключом k и значением v, если в коллекции еще нет элемента с подобным ключом.
- **Set<K> keySet():** возвращает набор всех ключей отображения
- **Collection<V> values():** возвращает набор всех значений отображения
- **void putAll(Map<? extends K, ? extends V> map):** добавляет в коллекцию все объекты из отображения map
- **V remove(Object k):** удаляет объект с ключом k
- **int size():** возвращает количество элементов коллекции

интерфейс Map

Чтобы положить объект в коллекцию, используется метод **put()**, а чтобы получить по ключу - метод **get()**. Реализация интерфейса Map также позволяет получить наборы как ключей, так и значений. А метод **entrySet()** возвращает набор всех элементов в виде объектов **Map.Entry<K, V>**.

интерфейс Map.Entry<K, V>

Обобщенный интерфейс **Map.Entry<K, V>** представляет объект с ключом типа K и значением типа V и определяет следующие методы:

• **boolean equals(Object obj)**: возвращает true, если объект obj, представляющий интерфейс Map.Entry, идентичен текущему

• **K getKey()**: возвращает ключ объекта отображения

• **V getValue()**: возвращает значение объекта отображения

• **V setValue(V v)**: устанавливает для текущего объекта значение v

• **int hashCode()**: возвращает хеш-код данного объекта При переборе объектов отображения мы будем оперировать этими методами для работы с ключами и значениями объектов.

Наиболее распространенным классом отображений является **HashMap**, который реализует интерфейс Map и наследуется от класса AbstractMap.

Пример

```
class Person{
    private String name;

    public Person(String value){
        name=value;
    }
    String getName(){return name;}
}
```

```
Map<Integer, String> states = new HashMap<Integer, String>();
states.put(1, "Germany");
states.put(2, "Spain");
states.put(4, "France");
states.put(3, "Italy");
// получим объект по ключу 2
String first = states.get(2);
System.out.println(first);
// получим весь набор ключей
Set<Integer> keys = states.keySet();
// получить набор всех значений
Collection<String> values = states.values();
//заменить элемент
states.replace(1, "Poland");
// удаление элемента по ключу 2
states.remove(2);
// перебор элементов
for(Map.Entry<Integer, String> item : states.entrySet()){
    System.out.printf("Key: %d  Value: %s \n", item.getKey(), item.getValue());
}
Map<String, Person> people = new HashMap<String, Person>();
people.put("1240i54", new Person( value: "Tom"));
people.put("1564i55", new Person( value: "Bill"));
people.put("4540i56", new Person( value: "Nick"));
for(Map.Entry<String, Person> item : people.entrySet()){
    System.out.printf("Key: %s  Value: %s \n", item.getKey(), item.getValue().getName());
}
```