RandomizedCollection is a data structure that contains a collection of numbers, possibly duplicates (i.e., a multiset). It should support inserting and removing specific elements and also reporting a random element.

Implement the RandomizedCollection class:

- **RandomizedCollection() Initializes the empty RandomizedCollection object.**
- **bool insert(int val) Inserts an item val into the multiset, even if the item is already present. Returns true if the item is not present, false otherwise.**
- **bool remove(int val) Removes an item val from the multiset if present. Returns true if the item is present, false otherwise. Note that if val has multiple occurrences in the multiset, we only remove one of them.**
- **int getRandom() Returns a random element from the current multiset of elements. The probability of each element being returned is linearly related to the number of the same values the multiset contains.**

You must implement the functions of the class such that each function works on average O(1) time complexity.

Note: The test cases are generated such that getRandom will only be called if there is at least one item in the RandomizedCollection.

Example 1:

```
Input
["RandomizedCollection", "insert", "insert", "insert", "getRandom", "remove", "getRandom"]
[[], [1], [1], [2], [], [1], []]
Output
[null, true, false, true, 2, true, 1]

Explanation
RandomizedCollection randomizedCollection = new RandomizedCollection();
randomizedCollection.insert(1);   // return true since the collection does not contain 1.
                                  // Inserts 1 into the collection.
randomizedCollection.insert(1);   // return false since the collection contains 1.
                                  // Inserts another 1 into the collection. Collection now contains [1,1].
randomizedCollection.insert(2);   // return true since the collection does not contain 2.
                                  // Inserts 2 into the collection. Collection now contains [1,1,2].
randomizedCollection.getRandom(); // getRandom should:
                                  // - return 1 with probability 2/3, or
                                  // - return 2 with probability 1/3.
randomizedCollection.remove(1);   // return true since the collection contains 1.
                                  // Removes 1 from the collection. Collection now contains [1,2].
randomizedCollection.getRandom(); // getRandom should return 1 or 2, both equally likely.
```

Constraints:

- $-2^{31} <= val <= 2^{31} - 1$
- At most $2 * 10^5$ calls in total will be made to insert, remove, and getRandom.
- There will be at least one element in the data structure when getRandom is called.

Solution:

```
class RandomizedCollection {
    List<Integer> nums; // To store the elements
    Map<Integer, Set<Integer>> idxMap; // To store the indices of each element
    Random rand;

    public RandomizedCollection() {
        nums = new ArrayList<>();
        idxMap = new HashMap<>();
        rand = new Random();
    }

    public boolean insert(int val) {
        boolean notPresent = !idxMap.containsKey(val);
        idxMap.putIfAbsent(val, new HashSet<>());
        idxMap.get(val).add(nums.size());
        nums.add(val);
        return notPresent;
    }

    public boolean remove(int val) {
        if (!idxMap.containsKey(val) || idxMap.get(val).isEmpty()) {
            return false;
        }
        int removeIdx = idxMap.get(val).iterator().next(); // Get an index of the element to remove
        idxMap.get(val).remove(removeIdx);
        if (removeIdx < nums.size() - 1) {
            int lastVal = nums.get(nums.size() - 1);
            nums.set(removeIdx, lastVal);
            idxMap.get(lastVal).remove(nums.size() - 1);
            idxMap.get(lastVal).add(removeIdx);
        }
        nums.remove(nums.size() - 1);
        if (idxMap.get(val).isEmpty()) {
            idxMap.remove(val);
        }
        return true;
    }
    public int getRandom() {
        return nums.get(rand.nextInt(nums.size()));
    }
}

/**
 * Your RandomizedCollection object will be instantiated and called as such:
 * RandomizedCollection obj = new RandomizedCollection();
 * boolean param_1 = obj.insert(val);
 * boolean param_2 = obj.remove(val);
 * int param_3 = obj.getRandom();
 */
```