Hashing Gone Crazy

Here is a video walkthrough of the solutions.

For this question, use the following `TA` class for reference.

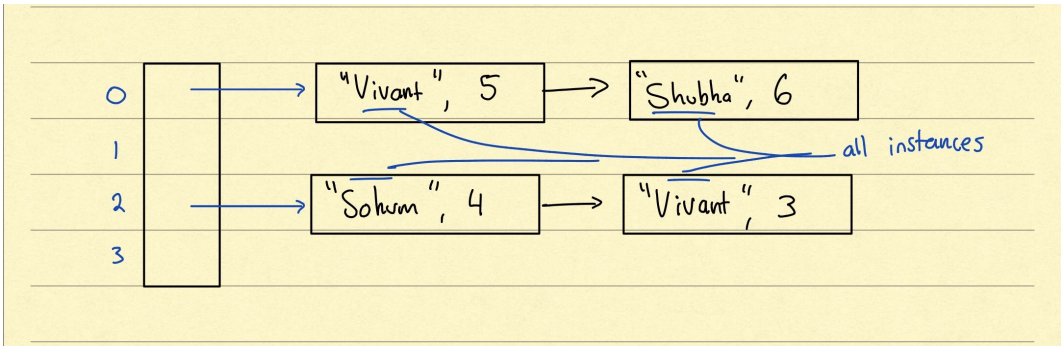
```
public class TA {
    int charisma;
    String name;
    TA(String name, int charisma) {
        this.name = name;
        this.charisma = charisma;
    }
    @Override
    public boolean equals(Object o) {
        TA other = (TA) o;
        return other.name.charAt(0) == this.name.charAt(0);
    }
    @Override
    public int hashCode() {
        return charisma;
    }
}
```

Assume that the `hashCode` of a `TA` object returns `charisma`, and the `equals` method returns `true` if and only if two `TA` objects have the same first letter in their `name`.

Assume that the `ECHashMap` is a `HashMap` implemented with external chaining as depicted in lecture. The `ECHashMap` instance begins at size 4 and, for simplicity, does not resize. Draw the contents of `map` after the executing the insertions below:

```
ECHashMap<TA, Integer> map = new ECHashMap<>();
TA sohum = new TA("Sohum", 10);
TA vivant = new TA("Vivant", 20);
map.put(sohum, 1);
map.put(vivant, 2);

vivant.charisma += 2;
map.put(vivant, 3);

sohum.name = "Vohum";
map.put(vivant, 4);

sohum.charisma += 2;
map.put(sohum, 5);

sohum.name = "Sohum";
TA shubha = new TA("Shubha", 24);
map.put(shubha, 6);
```

Solution:



Explanation:

Line 4: `sohum` has `charisma` value `10`. `10 % 4 = 2`, so `sohum` is placed in bucket 2 with value 1.

`0: [], 1: [], 2: [(sohum, 1)], 3: []`

Line 5: `vivant` is placed in bucket 0 with value 2.

`0: [(vivant, 2)], 1: [], 2: [(sohum, 1)], 3: []`

Line 7: Increasing the `charisma` value of `vivant` does *not* cause it to be rehashed! (This is why modifying objects in a Hashmap is dangerous–it can change the hash-code of your object and make it impossible to find which bucket it belongs to).

Line 8: `vivant` now has `charisma` 4, so bucket 2 also has a node pointing to `vivant`, with value 3. (Note that the two `vivants` refer to the same object).

`0: [(vivant, 2)], 1: [], 2: [(sohum, 1), (vivant, 3)], 3: []`

Line 11, 12: `vivant` with `charisma` 22 hashes to bucket 2. However, since we have changed `sohum`'s name to be `"Vohum"`, `vivant.equals(sohum)` returns true. Since we are hashing a key that is already present in the dictionary according to `.equals`, we replace `sohum`'s old value with the new value, 4.

`0: [(vivant, 2)], 1: [], 2: [(sohum, 4), (vivant, 3)], 3: []`

Line 13, 14: `sohum` with `charisma` 12 hashes to bucket 0. However, since we have changed `sohum`'s name to be `"Vohum"`, `sohum.equals(vivant)` returns true. Since we are hashing a key that is already present in the dictionary according to `.equals`, we replace `vivant`'s old value with the new value, 5.

`0: [(vivant, 5)], 1: [], 2: [(sohum, 4), (vivant, 3)], 3: []`

Line 16, 17, 18: `shuba` hashes to bucket 0. `shuba.equals(vivant)` returns `false`, so we add a new node after `vivant` with value 6.

`0: [(vivant, 5), (shuba, 6)], 1: [], 2: [(sohum, 4), (vivant, 3)], 3: []`