CS 314 Final Review — Most Frequent Element — **Solution**

Maps

Implement an instance method which, for a given list of elements returns the element which appears in the list the most frequently. This is also known as the *mode*. This method will be a part of the `List314` class, an array-based list similar to the one created in lecture. Return the element in the list which is present most often, if there is a tie, return the element which occurs first in the list. The list will not be altered as a result of this call.

Complete the following method.

```
// Returns the most frequent element in the list
// pre: list != null, list.size() > 0
// post: returns the most frequent element, list is unchanged
public E mostFrequent()
```

Here are some example calls to `mostFrequent()`:
`["A", "A", "B", "A", "B"].mostFrequent())` → `"A"`
`["A", "B", "C", "A", "B", "A", "BB"].mostFrequent()` → `"A"`
`[1, 2, 3, 2, 1].mostFrequent()` → `1`
`[3, 1, 4].mostFrequent()` → `3`

Your method will be in the following `List314` class:

```
public class List314<E>{
  private int size;
  private E[] con;
  // ...
}
```

You may create a `TreeMap` or `HashMap`.
Do not use any other Java classes or methods.

```
// Returns the most frequent element in the list
// pre: list != null, list.size() > 0
// post: returns the most frequent element, list is unchanged
public E mostFrequent(){
    HashMap<E, Integer> freqs = new HashMap<E, Integer>();
    E mostSeen = null;
    int maxFreq = -1;
    for(int i = 0; i < size; i++){
        int freq = freqs.getOrDefault(con[i], 0);
        freq++;
        freqs.put(con[i], freq);
        if(freq > maxFreq){
            mostSeen = con[i];
            maxFreq = freq;
        }
    }
    return mostSeen;
}
```

Relatively straightforward map and ArrayList problem. Using a map for this problem makes keeping track of the frequencies of the elements easy and efficient. However, using a TreeMap for this problem would be incorrect because TreeMaps require elements whose class implements the `Comparable` interface, and since this was never guaranteed in a precondition, using a TreeMap could result in a `ClassCastException`.

Also, keeping track of the maximum frequency we've seen so far as we traverse the array makes our lives a lot easier. It prevents us from having to go through the map once we're done to find the max and it makes it easy to break ties.