CS 314 Final Review — Hash Set Difference – **Solution**

Hash Tables/Sets

Write an instance method for the provided `HashSet314` class which will find the difference of this set with an `otherSet` parameter. This method will return a new `HashSet314` object and will leave both `this` and `otherSet` unaltered.

The `HashSet314` class uses a hash table as its internal storage container. There are no duplicate elements are in the set. The internal array for the hash table uses linear probing to handle hash collisions. If an element which was previously in the set was removed, its spot in the array will have been replaced with a refernce to `EMPTY`, a static, empty `Object`.

When creating the new set, you can assume the `HastSet314` class's `add(E val)` method (and all necessary resizing/rehashing) has been implemented correctly.

An element will be present in the resulting set if and only if it is present in `this` but not in `otherSet`.

You may use the following `HashSet314` implementation.

```
public class HashSet314<E> {
    private static final int INITIAL_CAPACITY = 10;
    private static final Object EMPTY = new Object();

    // All non-null elements in this array are guaranteed to
    // be either EMPTY or of type E.
    private Object[] con;
    private int size;

    //HashSet constructor
    public HashSet314(){
      con = (E[]) new Object[INITIAL_CAPACITY];
    }

    //You may use this method, assume it has been implemented here
    public boolean add(E val);
}
```

Notice that the internal array of `HashSet314` is of type `Object[]`. This is necessary to hold on to the reference to `EMPTY`, but you can safely assume that all non-`null` and non-`EMPTY` reference stored in the array will be to objects of type `E`. (This may result you in making some casts which a compiler would typically call unsafe, but it is mostly unavoidable in this situation).

You may not use methods in the `HashSet314` class except for `add(E val)` and the constructor. You may use the `hashCode()` method and methods from the `Math` class, but do not use any other Java classes or methods.

```
 /* Pre: otherSet != null
  * Post: returns a new set which represents this - otherSet.
  *       Both this and otherSet are unaltered by this method call.
  */
public HashSet314<E> difference(HashSet314<E> otherSet){
    HashSet314<E> result = new HashSet314<>();
    int elementsSeen = 0;
    for(int i = 0; i < con.length && elementsSeen < size; i++) {
        if(con[i] != null && con[i] != EMPTY){
            elementsSeen++;
            if(!otherSet.contains(con[i]))
                result.add((E) con[i]); //Gacky cast, but necessary
        }
    }
    return result;
}

//New, private instance method to check if val is contained in a set.
//We made this private because we would want the publicly-accessible
//contains() method to only accept vals of type E
private boolean contains(Object val){
    int index = Math.abs(val.hashCode() % con.length);
    int numVisited = 0;
    while(con[index] != null && !con[index].equals(val)
        && numVisited < con.length){

        numVisited++;
        index = (index + 1) & con.length;
    }
    //Be careful of nulls here
    return val.equals(con[index]);
}
```

This question requires you to deal with two different hash sets, where indexes of elements in one set are most likely not the same as the in the other set. The hardest part of this problem is checking to see if an element is in the other set. We have to first find the index we would expect the value to occur by taking the absolute value of `hashCode() % con.length`. Then, since the hash set uses linear probing, we may have to iterate a few times down the array before we find the element of interest. To be more specific, we have to keep checking subsequent indices in the array until: we've found the element, we hit a null (this signifies the end of a block of elements with the same hash code), or we've visited every element in the array (no null spots, element not present).

In the for loop of the `difference` method, there's an additional condition to stop looping: if we've seen every element in the hash set before we've gone through the entire array. If this problem appeared on an exam, not having this early stopping condition would almost certainly result in losing points.