Assignment Week 4 Collections

Program 1: Create a custom hash map.

```java
import java.util.ArrayList;
import java.util.List;

public class _01_CustomHashMapTest {
    public static void main(String[] args) {
        HashMapCustom<String, Integer> hm = new HashMapCustom<>();

        hm.put("A1", 1);
        hm.put("A2",2);
        hm.put("A3",3);
        hm.put("A4",4);


        System.out.println(hm);

        System.out.println(hm.get("A1"));
        System.out.println(hm.get("Kuldeep"));

        hm.remove("A1");

        System.out.println(hm);

        System.out.println(hm.get("A1"));
    }
}


 class HashMapCustom<K, V> {

    // Class representing the <Key,Value> pair node
    private class MapNode<K, V> {
        private K key; // to store the key
        private V value; // to store the value
        private MapNode<K, V> next; // to store the reference of the next element

        MapNode(K key, V value) {
            this.key = key;
            this.value = value;
        }

        @Override
        public String toString() {
            return "MapNode [key=" + key + ", value=" + value + ", next=" + next + "]";
        }

    }

    // K is the key type
    // V is the value type
    private List<MapNode<K, V>> bucket;
    private int capacity; // length of the bucket
    private int size; // number of elements in the map
```

```
    private final int INITIAL_CAPACITY = 5; // initial length of the bucket array
    private final double THRESHOLD_LOAD_FACTOR = 0.75d; // the threashold load factor

    public HashMapCustom() {
        bucket = new ArrayList<>();

        capacity = INITIAL_CAPACITY;
        // we add the "capacity" number of elements to the list
        // because if we don't the list would have a size 0
        // we need to ensure that the required indices exist
        for (int i = 0; i < capacity; i++) {
            bucket.add(null);
        }

        /*
         * Note- We are using List here and not array.
         * Since, we are dealing with generic types, we cannot use
         * array. Also, since the size is dynamic, we cannot use
         * array.
         */
    }

    // this method will give the bucket index by
    // finding the hashcode and applying the compression function
    private int getBucketIndex(K key) {
        int hashcode = key.hashCode(); // Object hashcode() method returns the hashcode
        System.out.println("Hashcode for " + key + " is: " + hashcode + " index: " + (hashcode
% capacity));
        return Math.abs(hashcode % capacity); //return absolute to prevent negative values
    }

    public V get(K key) {
        // get the index
        int bucketIndex = getBucketIndex(key);

        // Get the head of the list at the bucketIndex
        MapNode<K, V> head = bucket.get(bucketIndex);

        while (head != null) {
            // use equals() method and not the == operator
            // since the key can be an object and == will
            // just compare the memory address
            // This also shows that we must always implement the
            // equals and hashcode methods for whichever key we want
            // to use
            if (head.key.equals(key)) {
                return head.value;
            }
            head = head.next;
        }

        // if the key does not exist
        return null;
    }

    public void put(K key, V value) {
```

```
        // get the index
        int bucketIndex = getBucketIndex(key);
        // get the head of the list
        MapNode<K, V> head = bucket.get(bucketIndex);

        // check if the corresponding entry is present or not
        while (head != null) {
            if (head.key.equals(key)) {
                head.value = value;
                return;
            }
            head = head.next;
        }

        // if the key is not present, then insert it
        size++;
        MapNode<K, V> newEntry = new MapNode<>(key, value);
        head = bucket.get(bucketIndex);
        newEntry.next = head;// add the new node at the first position
        bucket.set(bucketIndex, newEntry);

        // Once added now we need to take care of the load factor.
        // Calculate the current load factor
        double loadFactor = (1.0 * size) / capacity; // number of elements/number of buckets
        System.out.println("Load factor: " + loadFactor);
        if (loadFactor > THRESHOLD_LOAD_FACTOR) {
            rehash();
        }
    }

    private void rehash(){
        System.out.println("Rehashing...");
        List<MapNode<K,V>> temp=bucket;
        bucket=new ArrayList<>(); //re-initialize
        capacity*=2; //double the capacity
        for(int i=0;i<capacity;i++){
            bucket.add(null);
        }
        size=0; //for the new bucket

        //rehash each entry
        for(int i=0;i<temp.size();i++){
            MapNode<K,V> head=temp.get(i);
            while(head!=null){
                put(head.key,head.value);
                head=head.next;
            }
        }

    }

    public void remove(K key) {
        // get the index
        int bucketIndex = getBucketIndex(key);
        MapNode<K, V> head = bucket.get(bucketIndex);
        MapNode<K, V> prev = null;
```

```
        while (head != null) {
            if (head.key.equals(key)) {
                if (prev == null) {
                    bucket.set(bucketIndex, head.next);
                } else {
                    prev.next = head.next;
                }
                head.next = null;
                size--;
                break;
            }
            prev = head;
            head = head.next;
        }
    }

    @Override
    public String toString() {
        return "HashMapCustom [bucket=" + bucket + ", capacity=" + capacity + ", size=" + size
+ ", INITIAL_CAPACITY="
                + INITIAL_CAPACITY + "]";
    }

}
```

Output-

```
Hashcode for A1 is: 2064 index: 4
Load factor: 0.2
Hashcode for A2 is: 2065 index: 0
Load factor: 0.4
Hashcode for A3 is: 2066 index: 1
Load factor: 0.6
Hashcode for A4 is: 2067 index: 2
Load factor: 0.8
Rehashing...
Hashcode for A2 is: 2065 index: 5
Load factor: 0.1
Hashcode for A3 is: 2066 index: 6
Load factor: 0.2
Hashcode for A4 is: 2067 index: 7
Load factor: 0.3
Hashcode for A1 is: 2064 index: 4
Load factor: 0.4
HashMapCustom [bucket=[null, null, null, null, MapNode [key=A1, value=1, next=null], MapNode [key=A2, value=2, next=null], MapNode [key=A3, value=3, next=null], MapNode [key=A4, value=4, next=null], null, null], capacity=10, si
ze=4, INITIAL_CAPACITY=5]
Hashcode for A1 is: 2064 index: 4
1
Hashcode for Kuldeep is: 1295729678 index: 8
null
Hashcode for A1 is: 2064 index: 4
HashMapCustom [bucket=[null, null, null, null, null, MapNode [key=A2, value=2, next=null], MapNode [key=A3, value=3, next=null], MapNode [key=A4, value=4, next=null], null, null], capacity=10, size=3, INITIAL_CAPACITY=5]
Hashcode for A1 is: 2064 index: 4
null
```

Program 2: Implement Stack using Arrays or List

```
public class _02_StackImpl {
    public static void main(String[] args) {
        Stack_Array stk1 = new Stack_Array(5);

        testArrayStack(stk1);
    }


    private static void testArrayStack(Stack_Array stk1) {
        stk1.push(209);
        stk1.push(345);
        stk1.push(45);

        System.out.println("Top element of the array stack: " + stk1.peek());
```

```
        System.out.println("Is empty: " + stk1.isEmpty());
        System.out.println("Is full: " + stk1.isFull());

        stk1.push(24);
        stk1.push(25);
        stk1.push(25);

        System.out.println("Is full: " + stk1.isFull());

        stk1.pop();
        stk1.pop();
        stk1.pop();
        stk1.pop();
        stk1.pop();
        stk1.pop();
    }

}

class Stack_Array {
    private int maxSize;
    private int[] stackArray;
    private int top;

    public Stack_Array(int size) {
        this.maxSize = size;
        this.stackArray = new int[maxSize];
        this.top = -1;
    }

    public void push(int value) {
        if (isFull()) {
            System.out.println("Stack overflow! " + value);
            return;
        }
        stackArray[++top] = value;
        System.out.println("Pushed " + value);
    }

    // Method to pop an element from the stack
    public int pop() {
        if (isEmpty()) {
            System.out.println("Stack underflow!");
            return -1;
        }
        int poppedElement = stackArray[top--];
        System.out.println("Popped " + poppedElement);
        return poppedElement;
    }

    public int peek() {
        if (isEmpty()) {
            System.out.println("Stack is empty!");
            return -1;
        }
        return stackArray[top];
```

```
    }

    public boolean isEmpty() {
        return top == -1;
    }

    public boolean isFull() {
        return top == maxSize - 1;
    }

}
```

Output:

```
Pushed 209
Pushed 345
Pushed 45
Top element of the array stack: 45
Is empty: false
Is full: false
Pushed 24
Pushed 25
Stack overflow! 25
Is full: true
Popped 25
Popped 24
Popped 45
Popped 345
Popped 209
Stack underflow!
```

Program 3: Sort array of 0s and 1s efficiently

```
import java.util.Arrays;
public class _03_Sort_Arrays_of_0and1 {
    public static void main(String[] args) {
        int arr[] = { 1, 0, 1, 0, 0, 1, 1, 0, 0, 0 };
        System.out.println("Array before sorting: " + Arrays.toString(arr));
        int n = arr.length;
        sortArray(arr, 0, n - 1);
        System.out.println("Array after sorting: " + Arrays.toString(arr));
    }

    private static void sortArray(int[] arr, int i, int j) {
        while (i < j) {
            if (arr[i] == 1) {
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
                j--;
            } else {
                i++;
            }
        }
    }
}
```

Output:

```
Array before sorting: [1, 0, 1, 0, 0, 1, 1, 0, 0, 0]
Array after sorting: [0, 0, 0, 0, 0, 0, 1, 1, 1, 1]
```

Program 4: Check for balanced parentheses

```
import java.util.Stack;
public class _4_BalancedParentheses {
    public static void main(String[] args) {
        String s="[{()}]";
        String s2="[{()}";
        String s3="[{}{}([]";
        System.out.println("Is "+s+" valid: "+isValid(s));
        System.out.println("Is "+s2+" valid: "+isValid(s2));
        System.out.println("Is "+s3+" valid: "+isValid(s3));
    }

    public static boolean isValid(String s) {
        Stack<Character> stack = new Stack<Character>();

        for (char c : s.toCharArray()) {
            if (c == '(' || c == '[' || c == '{') {
                stack.push(c);
            } else {

                if (stack.isEmpty()) {
                    return false;
                }

                char top = stack.peek();
                if ((c == ')' && top == '(') || (c == ']' && top == '[') || (c == '}' && top ==
'{')) {
                    stack.pop();
                } else {
                    return false;
                }
            }
        }

        return stack.isEmpty();
    }
}
```

Output:

```
Is [{()}] valid: true
Is [{()} valid: false
Is [{}{}([] valid: false
```