**1)** Problem Statement

Design and implement a data structure for a Least Recently Used (LRU) cache. It should support the following operations: get and put.

Constraints

The number of get and put operations will be in the range [1, 10^5].

The capacity of the cache is between 1 and 10^5.

Program:

```java
import java.util.HashMap;
import java.util.LinkedList;
import java.util.Map;

class LRUCache {
  private int capacity;
  private Map<Integer, Integer> cache;
  private LinkedList<Integer> accessOrder;

  public LRUCache(int capacity) {
    this.capacity = capacity;
    this.cache = new HashMap<>();
    this.accessOrder = new LinkedList<>();
  }

  public int get(int key) {
    if (cache.containsKey(key)) {
      // Move the accessed key to the front of the access order
      accessOrder.remove((Integer) key);
      accessOrder.addFirst(key);
      return cache.get(key);
    } else {
      return -1;
    }
  }

  public void put(int key, int value) {
    if (cache.containsKey(key)) {
      // Update the value and move the key to the front of the access order
      cache.put(key, value);
      accessOrder.remove((Integer) key);
      accessOrder.addFirst(key);
    } else {
      if (cache.size() >= capacity) {
        // Evict the least recently used key-value pair
        int lruKey = accessOrder.removeLast();
        cache.remove(lruKey);
```

```
                }
                // Add the new key-value pair and move the key to the front of the access order
                cache.put(key, value);
                accessOrder.addFirst(key);
            }
        }
    }

    public class Main {
        public static void main(String[] args) {
            // Example usage of LRUCache
            LRUCache cache = new LRUCache(2); // Capacity is 2

            cache.put(1, 1);
            cache.put(2, 2);
            System.out.println(cache.get(1)); // Output: 1
            cache.put(3, 3); // Evicts key 2
            System.out.println(cache.get(2)); // Output: -1 (key 2 not found)
            cache.put(4, 4); // Evicts key 1
            System.out.println(cache.get(1)); // Output: -1 (key 1 not found)
            System.out.println(cache.get(3)); // Output: 3
            System.out.println(cache.get(4)); // Output: 4
        }
    }
```

2)Write a Java program that demonstrates the `ConcurrentModificationException`. Explain why the exception is thrown and how to handle it properly.

Program:
```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Iterator;
import java.util.List;

public class ConcurrentModificationDemo {
    public static void main(String[] args) {

        List<String> list = new ArrayList<>(Arrays.asList("A", "B", "C"));

        // Attempting to remove element "B" using enhanced for loop (incorrect way)
        try {
            for (String s : list) {
                if (s.equals("B")) {
                    list.remove(s); // ConcurrentModificationException may occur here
                }
            }
        } catch (Exception e) {
            System.out.println("Concurrent modification exception caught!");
        }
        Iterator<String> iterator = list.iterator();
        while (iterator.hasNext()) {
            String s = iterator.next();
            if (s.equals("B")) {
                iterator.remove(); // Safe to remove element "B" using iterator
            }
        }


        System.out.println("List after removing 'B': " + list);
    }
}
```

3) Create a custom annotation `@LogExecutionTime` to log the execution time of annotated methods. Implement an annotation processor to handle this annotation.

Program:

```
import java.lang.annotation.*;
import java.lang.reflect.Method;

@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
@interface LogExecutionTime {
}

class AnnotationProcessor {
    public static void logExecutionTime(Object obj) throws Exception {
        for (Method method : obj.getClass().getDeclaredMethods()) {
            if (method.isAnnotationPresent(LogExecutionTime.class)) {
                long start = System.currentTimeMillis();
                method.setAccessible(true);
                method.invoke(obj);
                long end = System.currentTimeMillis();
                System.out.println("Execution time of " + method.getName() + " : " + (end - start) + "ms");
            }
        }
    }
}

class TestClass {
    @LogExecutionTime
    public void methodToLog() {
        // Simulate method execution time
        try {
            Thread.sleep(200);
        } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        public static void main(String[] args) throws Exception {
            TestClass test = new TestClass();
            AnnotationProcessor.logExecutionTime(test);
        }
    }
```

4) Design an algorithm to serialize and deserialize a binary tree ?

Program:

```
import java.util.*;

class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;

    TreeNode(int x) {
        val = x;
    }
}

class Codec {
    // Encodes a tree to a single string.
    public String serialize(TreeNode root) {
        if (root == null) {
            return "null";
        }
        Queue<TreeNode> queue = new LinkedList<>();
        StringBuilder sb = new StringBuilder();
        queue.add(root);

        while (!queue.isEmpty()) {
            TreeNode node = queue.poll();
            if (node == null) {
                sb.append("null,");
            } else {
                sb.append(node.val).append(",");
                queue.add(node.left);
                queue.add(node.right);
            }
        }

        return sb.toString();
    }
```

```
        // Decodes your encoded data to tree.
        public TreeNode deserialize(String data) {
            if (data.equals("null")) {
                return null;
            }
            String[] values = data.split(",");
            Queue<TreeNode> queue = new LinkedList<>();
            TreeNode root = new TreeNode(Integer.parseInt(values[0]));
            queue.add(root);
            int i = 1;

            while (!queue.isEmpty()) {
                TreeNode node = queue.poll();
                if (!values[i].equals("null")) {
                    node.left = new TreeNode(Integer.parseInt(values[i]));
                    queue.add(node.left);
                }
                i++;
                if (!values[i].equals("null")) {
                    node.right = new TreeNode(Integer.parseInt(values[i]));
                    queue.add(node.right);
                }
                i++;
            }

            return root;
        }
    }
```

5) Given a string containing just the characters '(', ')', '{', '}', '[', and ']', determine if the input string is valid.

Program:

```
    import java.util.*;

    public class ValidParentheses {
        public boolean isValid(String s) {
            Stack<Character> stack = new Stack<>();
            for (char c : s.toCharArray()) {
                if (c == '(' || c == '{' || c == '[') {
                    stack.push(c);
                } else {
                    if (stack.isEmpty()) {
                        return false;
                    }
                    char top = stack.pop();
                    if ((c == ')' && top != '(') ||
                        (c == '}' && top != '{') ||
                    (c == ']' && top != '[')) {
                    return false;
                }
            }
        }
        return stack.isEmpty();
    }
}
```

6)Implement a trie with insert, search, and startsWith methods insert(word): Inserts a word into the trie. earch(word): Returns if the word is in the trie startsWith(prefix): Returns if there is any word in the trie that starts with the given prefix.

Constraints

You may assume that all inputs are consist of lowercase letters a-z.

All inputs are guaranteed to be non-empty strings.

```
class TrieNode {
    boolean isEndOfWord;
    TrieNode[] children;

    public TrieNode() {
        isEndOfWord = false;
        // Initialize children array for 26 lowercase English letters
        children = new TrieNode[26];
    }
}

class Trie {
    private TrieNode root;

    public Trie() {
        // Initialize the Trie with an empty root node
        root = new TrieNode();
    }

    public void insert(String word) {
        TrieNode node = root;
        // Traverse each character in the word
        for (char c : word.toCharArray()) {
            // Calculate index for current character 'c'
            int index = c - 'a';
            // If current character node does not exist, create a new node
            if (node.children[index] == null) {
                node.children[index] = new TrieNode();
            }
            // Move to the next node
            node = node.children[index];
        }
        // Mark the end of the word
        node.isEndOfWord = true;
    }

    public boolean search(String word) {
        TrieNode node = root;
        // Traverse each character in the word
        for (char c : word.toCharArray()) {
            int index = c - 'a';
            // If current character node does not exist, return false
            if (node.children[index] == null) {
                return false;
            }
            // Move to the next node
            node = node.children[index];
        }
        // Return true if we reached the end of a valid word
        return node.isEndOfWord;
    }

    public boolean startsWith(String prefix) {
        TrieNode node = root;
        // Traverse each character in the prefix
        for (char c : prefix.toCharArray()) {
            int index = c - 'a';
            // If current character node does not exist, return false
            if (node.children[index] == null) {
                return false;
            }
            // Move to the next node
            node = node.children[index];
        }
        // Return true if the prefix is found
        return true;
    }
}

public class Main {
    public static void main(String[] args) {



       trie.insert("apple");
        trie.insert("banana");
        trie.insert("apricot");
        // Search for words in the Trie
        System.out.println(trie.search("apple"));   // true
        System.out.println(trie.search("apricot")); // true
        System.out.println(trie.search("banana"));  // true
        System.out.println(trie.search("orange"));  // false

        // Check if prefixes exist in the Trie
        System.out.println(trie.startsWith("ap"));   // true
        System.out.println(trie.startsWith("ban"));  // true
        System.out.println(trie.startsWith("ora"));  // false
    }
}
```

7) Given n non-negative integers a1, a2, ..., an, find two lines which together with the x-axis form a container, such that the container contains the most water.

Program:

```
public class ContainerWithMostWater {
    public int maxArea(int[] height) {
        int maxArea = 0;
        int left = 0, right = height.length - 1;
        while (left < right) {
            int width = right - left;
            int currentHeight = Math.min(height[left], height[right]);
            maxArea = Math.max(maxArea, width * currentHeight);
            if (height[left] < height[right]) {
                left++;
            } else {
                right--;
            }
        }
        return maxArea;
    }
}
```

8)Find the kth largest element in an unsorted array?

Program:

```
import java.util.Arrays;
    import java.util.Scanner;
   public class KthLargestNumber {
        public static void main(String[] args) {

            Scanner sc = new Scanner(System.in);
            System.out.println("Enter the value of k:");
            int k = sc.nextInt();
             int[] a = new int[n];
            System.out.println("Enter the elements of the array:");
```

```
                    for (int i = 0; i < n; i++) {
                         a[i] = sc.nextInt(); }
                     sc.close();
                    Arrays.sort(a);
            if (k > 0 && k <= a.length) {
              System.out.println("The " + k + "-th largest element is: " + a[a.length - k]);
              }
            else {
                 System.out.println("Invalid input for k");
    }
  }
}
```

9) Design an interval tree to efficiently find all intervals that overlap with a given interval.?

Program:

```
import java.util.*;

class Interval {
    int start, end;

    public Interval(int start, int end) {
        this.start = start;
        this.end = end;
    }
}

class IntervalTree {
    private TreeMap<Integer, Interval> treeMap;

    public IntervalTree() {
        this.treeMap = new TreeMap<>();
    }

    public void insertInterval(int start, int end) {
        treeMap.put(start, new Interval(start, end));
    }

    public void deleteInterval(int start, int end) {
        treeMap.remove(start);
    }

    public List<Interval> findOverlappingIntervals(int start, int end) {
        List<Interval> result = new ArrayList<>();
        for (Interval interval : treeMap.values()) {
            if (interval.start <= end && interval.end >= start) {
                result.add(interval);
            }
        }
        return result;
    }
}
```
10) Write a Java program that checks if a given string is a palindrome.

Program:

```
import java.util.*;

public class Palindrome {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a string: ");
        String input = sc.nextLine();
        sc.close();

        if (isPalindrome(input)) {
            System.out.println("Palindrome");
        } else {
            System.out.println("Not a palindrome");
        }
    }

    // Function to check if a string is a palindrome
    public static boolean isPalindrome(String str) {
        // Remove all non-alphanumeric characters and convert to lowercase
        String cleanStr = str.replaceAll("[^a-zA-Z0-9]", "").toLowerCase();

        int left = 0;
        int right = cleanStr.length() - 1;

        while (left < right) {
            if (cleanStr.charAt(left) != cleanStr.charAt(right)) {
                return false; // Characters do not match, not a palindrome
            }
            left++;
            right--;
        }
        return true; // All characters matched, it's a palindrome
    }
}
```