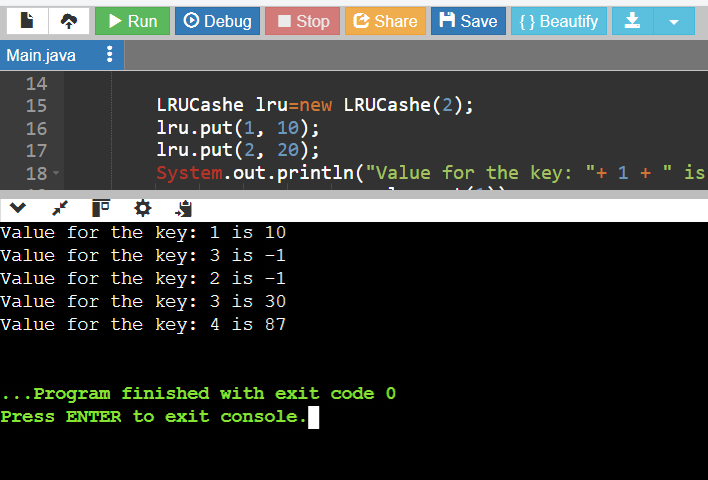
# Least Recently Used (LRU) cache

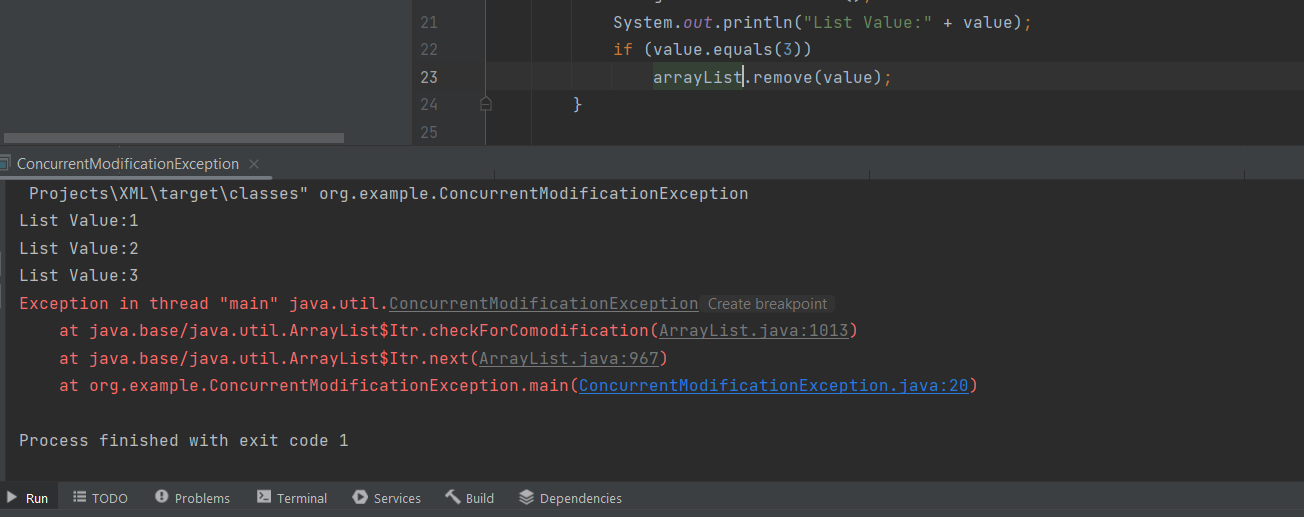
package org.example;  
  
import java.util.\*;  
public class Main  
{  
 public static void main(String[] args) {  
 //System.out.println("Hello World");  
  
 LRUCashe lru=new LRUCashe(2);  
 lru.put(1, 10);  
 lru.put(2, 20);  
 System.*out*.println("Value for the key: "+ 1 + " is "  
 + lru.get(1));  
  
 System.*out*.println("Value for the key: "+ 3 + " is " + lru.get(3));  
  
 lru.put(3, 30);  
  
 System.*out*.println(  
 "Value for the key: 2 is "  
 + lru.get(2)); // returns -1 (not found)  
  
  
 lru.put(4,87);  
  
 System.*out*.println("Value for the key: "+ 3 + " is " + lru.get(3));  
 System.*out*.println("Value for the key: "+ 4 + " is " + lru.get(4));  
  
 }  
}  
  
class Node{  
 int key;  
 int value;  
 Node previous;  
 Node next;  
  
  
 Node(int key,int value){  
 this.key=key;  
 this.value=value;  
 }  
}  
  
class LRUCashe{  
 private HashMap<Integer,Node> mp;  
 private int capacity;  
 private int count;  
 private Node head;  
 private Node tail;  
  
 LRUCashe(int capacity){  
 this.capacity=capacity;  
 mp=new HashMap<>();  
 head = new Node(0, 0);  
 tail = new Node(0, 0);  
 head.next = tail;  
 tail.previous = head;  
 head.previous = null;  
 tail.next = null;  
 count = 0;  
 }  
  
 public void deleteNode(Node node)  
 {  
 node.previous.next = node.next;  
 node.next.previous = node.previous;  
 }  
  
 public void addToHead(Node node)  
 {  
 node.next = head.next;  
 node.next.previous = node;  
 node.previous = head;  
 head.next = node;  
 }  
  
  
 public int get(int key)  
 {  
 if (mp.get(key) != null) {  
 Node node = mp.get(key);  
 int result = node.value;  
 deleteNode(node);  
 addToHead(node);  
 return result;  
 }  
 return -1;  
 }  
  
 public void put(int key, int value)  
 {  
  
 if (mp.get(key) != null) {  
 Node node = mp.get(key);  
 node.value = value;  
 deleteNode(node);  
 addToHead(node);  
 }  
 else {  
 Node node = new Node(key, value);  
 mp.put(key, node);  
 if (count < capacity) {  
 count++;  
 addToHead(node);  
 }  
 else {  
 mp.remove(tail.previous.key);  
 deleteNode(tail.previous);  
 addToHead(node);  
 }  
 }  
 }  
}

The output for the above programme

# Concurrent Modification Exception

1. public class ConcurrentModificationException extends Exception {  
    public static void main(String[] args) {  
     
    ArrayList<Integer> arrayList = new ArrayList<>();  
     
    arrayList.add(1);  
    arrayList.add(2);  
    arrayList.add(3);  
    arrayList.add(4);  
    arrayList.add(5);  
     
    Iterator<Integer> it = arrayList.iterator();  
    while (it.hasNext()) {  
    Integer value = it.next();  
    System.*out*.println("List Value:" + value);  
    if (value.equals(3))  
    arrayList.remove(value);  
    }  
     
    }  
   }

the output for the above program

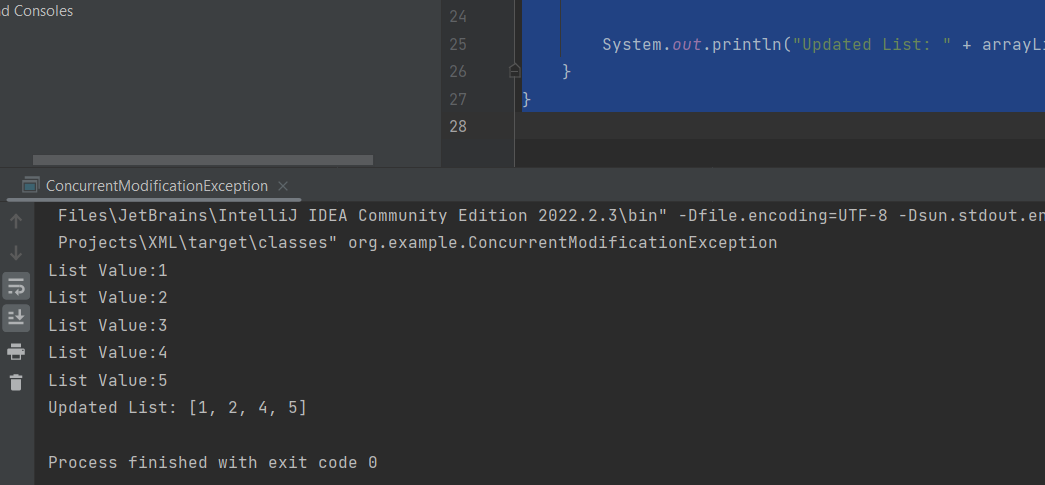


The exception is thrown when the next method is called as the iterator is iterating the list and we are making modifications in it simultaneously.

To handle this Exception we simply use this method shown in below

package org.example;  
  
import java.util.ArrayList;  
import java.util.Iterator;  
import java.util.List;  
  
public class ConcurrentModificationException extends Exception {  
 public static void main(String[] args) {  
 ArrayList<Integer> arrayList = new ArrayList<>();  
 arrayList.add(1);  
 arrayList.add(2);  
 arrayList.add(3);  
 arrayList.add(4);  
 arrayList.add(5);  
  
 Iterator<Integer> it = arrayList.iterator();  
 while (it.hasNext()) {  
 Integer value = it.next();  
 System.*out*.println("List Value:" + value);  
 if (value.equals(3)) {  
 it.remove(); // Proper way to remove elements during iteration  
 }  
 }  
  
 System.*out*.println("Updated List: " + arrayList);  
 }  
}

the out put for this above problem is



# 3. Logging Method Execution Time with Custom Annotation in Spring Boot

@SpringBootApplication  
@EnableAspectJAutoProxy  
public class Application {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(Application.class, args);  
 }  
}

package org.example;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
@Service  
public class ExampleService {  
 @Autowired  
 ExecutionTimeAspect executionTimeAspect;  
  
 @LogExecutionTime  
 public void performTask() {  
 // Simulate a task by sleeping for a random time  
 try {  
 Thread.*sleep*((long) (Math.*random*() \* 1000));  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public ExecutionTimeAspect getExecutionTimeAspect() {  
 return executionTimeAspect;  
 }  
}

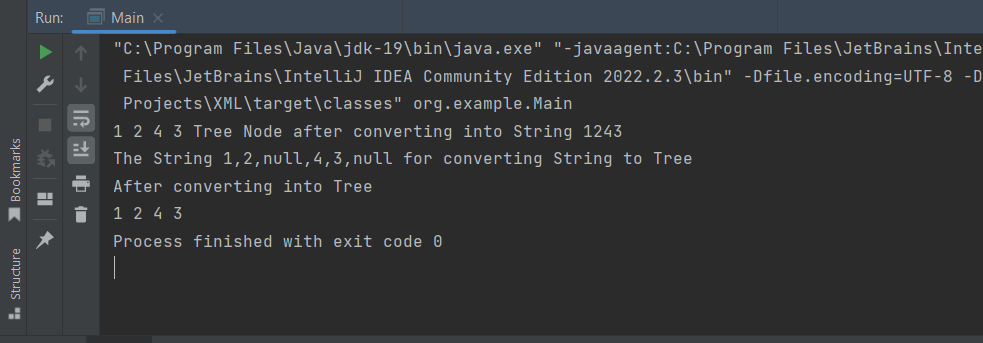
import java.lang.annotation.ElementType;  
import java.lang.annotation.Retention;  
import java.lang.annotation.RetentionPolicy;  
import java.lang.annotation.Target;  
  
@Retention(RetentionPolicy.*RUNTIME*)  
@Target(ElementType.*METHOD*)  
public @interface LogExecutionTime {  
  
}

import org.aspectj.lang.ProceedingJoinPoint;  
import org.aspectj.lang.annotation.Around;  
import org.aspectj.lang.annotation.Aspect;  
import org.springframework.stereotype.Component;  
  
@Aspect  
@Component  
public class ExecutionTimeAspect {  
  
 @Around("@annotation(LogExecutionTime)")  
 public Object logExecutionTime(ProceedingJoinPoint joinPoint) throws Throwable {  
 long startTime = System.*currentTimeMillis*();  
  
 Object returnValue = joinPoint.proceed();  
  
 long endTime = System.*currentTimeMillis*();  
 long executionDuration = endTime - startTime;  
  
 System.*out*.println(joinPoint.getSignature() + " executed in " + executionDuration + "ms");  
  
 return returnValue;  
 }  
}

# Serialize and Deserialize a Binary Tree

1. import java.util.\*;  
    public class Main  
    {  
    static StringBuilder *resString* = new StringBuilder("");  
     
    public static void main(String[] args) {  
     
    // creating tree  
    TreeNode root = new TreeNode(1);  
    root.left = new TreeNode(2);  
    root.right = new TreeNode(3);  
    root.left.left = new TreeNode(4);  
    System.*out*.println("Tree Node after converting into String "+*serlize*(root)); // calling and printing methode serialize tree node to string  
    System.*out*.println("The String "+"1,2,null,4,3,null"+" for converting String to Tree");  
    TreeNode node=*deserialize*("1,2,null,4,3,null"); // calling deserialize method so that it can convert string to tree  
    System.*out*.println("After converting into Tree");  
    *preOrder*(node); // printing converted tree  
     
    }  
    // method for converting tree to string  
    public static String serlize(TreeNode root) {  
    if(root == null) return *resString*.toString();  
    *resString*.append(root.val); // append root node  
     
    if((root.left == null && root.right == null)) return *resString*.toString();  
    *traverseTree*(root.left, 'l'); // method traverse to left child of root node  
    *traverseTree*(root.right, 'r'); // method traverse to right child of root node  
     
    return *resString*.toString(); // return the resutent string  
    }  
     
    // method to traverse Tree  
    public static void traverseTree(TreeNode root, char dir) {  
    if(root == null ){  
    return;  
    }  
     
     
    *resString*.append(root.val);  
    if(!(root.left == null && root.right == null)){  
    *traverseTree*(root.left, 'l');  
    *traverseTree*(root.right, 'r');  
     
    }  
     
    }  
     
     
    // method for converting string to tree  
    public static TreeNode deserialize(String data) {  
    if (data == null || data.isEmpty()) return null; //base condition  
     
    String[] nodes = data.split(","); // spliting string to make an array  
     
    TreeNode root = new TreeNode(Integer.*parseInt*(nodes[0]));  
    Queue<TreeNode> queue = new LinkedList<>();  
    queue.add(root);  
     
    int i = 1;  
    while (!queue.isEmpty()) {  
    TreeNode node = queue.poll();  
    if(i>=nodes.length) break;  
    if (!nodes[i].equals("null")) {  
    node.left = new TreeNode(Integer.*parseInt*(nodes[i]));  
    queue.add(node.left);  
    }  
    i++;  
    if(i>=nodes.length) break;  
    if (!nodes[i].equals("null")) {  
    node.right = new TreeNode(Integer.*parseInt*(nodes[i]));  
    queue.add(node.right);  
    }  
    i++;  
    }  
     
    return root;  
    }  
     
    public static void preOrder(TreeNode root){  
    if(root==null) return;  
    System.*out*.print(root.val+" ");  
    *preOrder*(root.left);  
    *preOrder*(root.right);  
    }  
     
     
    public static class TreeNode {  
    int val;  
    TreeNode left;  
    TreeNode right;  
    TreeNode() {}  
    TreeNode(int val) { this.val = val; }  
    TreeNode(int val, TreeNode left, TreeNode right) {  
    this.val = val;  
    this.left = left;  
    this.right = right;  
    }  
    }  
     
    }

the output for the above problem is



# Implementation Of Trie

1. package org.example;  
     
   import java.util.HashMap;  
     
   class TrieNode {  
    HashMap<Character, TrieNode> children;  
    boolean isEndOfWord;  
     
    public TrieNode() {  
    children = new HashMap<>();  
    isEndOfWord = false;  
    }  
   }  
     
   class Trie {  
    private TrieNode root;  
     
    public Trie() {  
    root = new TrieNode();  
    }  
     
    public void insert(String word) {  
    TrieNode node = root;  
    for (char c : word.toCharArray()) {  
    if (!node.children.containsKey(c)) {  
    node.children.put(c, new TrieNode());  
    }  
    node = node.children.get(c);  
    }  
    node.isEndOfWord = true;  
    }  
     
    public String search(String word) {  
    TrieNode node = root;  
    StringBuilder sb = new StringBuilder();  
    for (char c : word.toCharArray()) {  
    if (!node.children.containsKey(c)) {  
    return ""; // or return null,  
    }  
    sb.append(c);  
    node = node.children.get(c);  
    }  
    return node.isEndOfWord ? sb.toString() : ""; // Return the word only if it's a complete word  
    }  
     
    public boolean startsWith(String prefix) {  
    TrieNode node = root;  
    for (char c : prefix.toCharArray()) {  
    if (!node.children.containsKey(c)) {  
    return false;  
    }  
    node = node.children.get(c);  
    }  
    return true;  
    }  
   }  
     
   // Example usage:  
    class Main2 {  
    public static void main(String[] args) {  
    Trie trie = new Trie();  
     
    // Inserting words  
    trie.insert("apple");  
    trie.insert("banana");  
    trie.insert("app");  
     
    // Searching for words  
    System.*out*.println("Searching in trie "+trie.search("apple")); // Output: "apple"  
    System.*out*.println("Searching in trie " +trie.search("app")); // Output: "app"  
    System.*out*.println("Searching in trie "+trie.search("banana")); // Output: "banana"  
    System.*out*.println("Searching in trie "+trie.search("orange")); // Output: "" since word is not present in trie  
      
     
    // Checking prefixes  
    System.*out*.println(trie.startsWith("Searching in prefix trie"+" app")); // Output: true  
    System.*out*.println(trie.startsWith("Searching in prefix trie"+" ban")); // Output: true  
    System.*out*.println(trie.startsWith("Searching in prefix trie"+" or")); // Output: false  
    }  
   }

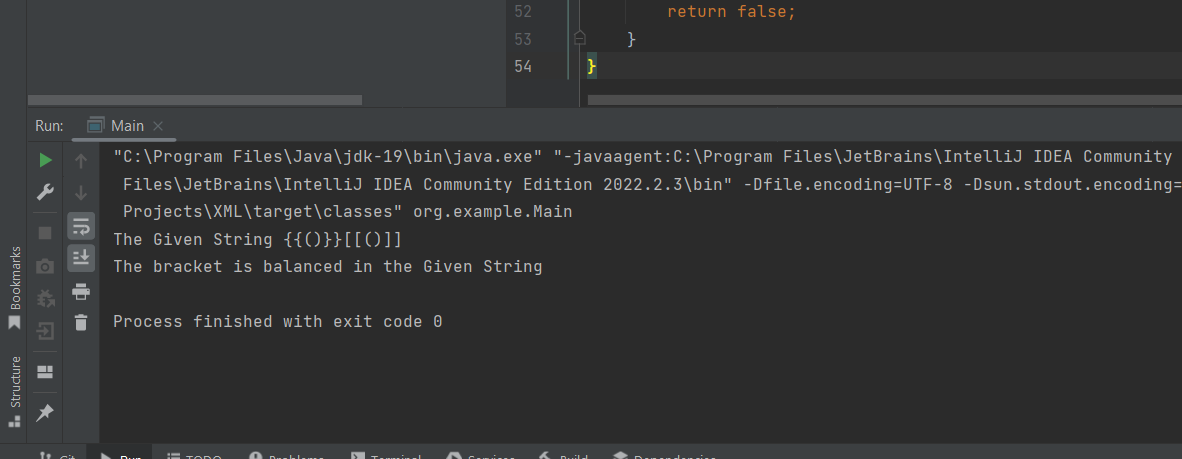
the output for the above program is

# 

6. Balanced Brackets.

import java.util.Scanner;  
import java.util.Stack;  
  
public class Main  
{  
 public static void main (String[] args) throws java.lang.Exception  
 {  
 //your code here  
 Scanner sc=new Scanner(System.*in*);  
  
 String s="{{()}}[[()]]";  
 System.*out*.println("The Given String "+s);  
 boolean ans=*balancedBracket*( s);  
 if(ans==true)  
 {  
 System.*out*.println("The bracket is balanced in the Given String");  
 }else  
 {  
 System.*out*.println("The bracket is not balanced in the Given String");  
 }  
 }  
 public static boolean balancedBracket(String s)  
 {  
 Stack< Character > st=new Stack<>();  
 for(int i=0;i<s.length();i++) //process through given string  
 {  
  
 if((s.charAt(i)=='(')||(s.charAt(i)=='[')||(s.charAt(i)=='{')) // if the string has open brackets push to stack  
 {  
 st.push(s.charAt(i));  
 }  
 else{  
 if(!st.empty()&& *match*(st.peek(),s.charAt(i))) // if stack is not empty and if the peek of stack match with closing bracket  
 {  
 st.pop(); // then pop  
 }else{  
 return false; // return false  
 }  
 }  
 }  
 if(st.empty()) return true; // finally check whether stack is empty if yes return true else false  
 return false;  
 }  
 public static boolean match(char open,char close)  
 {  
 if(open=='['&& close==']') return true;  
 if(open=='('&& close==')') return true;  
 if(open=='{'&& close=='}') return true;  
  
 return false;  
 }  
}

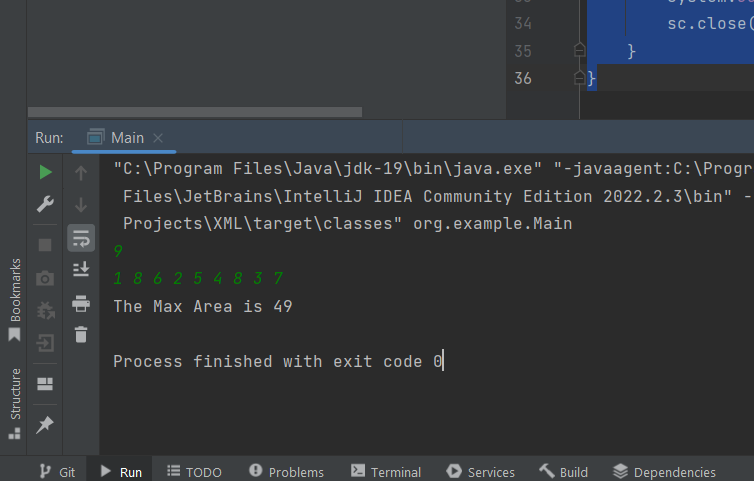
the out put for the given String



# Container With Most Water

1. package org.example;  
   import java.util.\*;  
     
   class Solution {  
    public int maxArea(int[] heights) {  
    //Write code here  
    int l=0;  
    int r=heights.length-1;  
    int area=0;  
    while(l<r)  
    {  
    area=Math.*max*(area,Math.*min*(heights[l],heights[r])\*(r-l));  
    if(heights[l]<=heights[r])  
    {  
    l++;  
    }else  
    r--;  
    }  
    return area;  
    }  
   }  
     
   public class Main {  
    public static void main(String[] args) {  
    Scanner sc = new Scanner(System.*in*);  
    int n;  
    n = sc.nextInt();  
    int arr[] = new int[n];  
    for (int i = 0; i < n; i++)  
    arr[i] = sc.nextInt();  
    Solution Obj = new Solution();  
    int result = Obj.maxArea(arr);  
    System.*out*.println("The Max Area is "+result);  
    sc.close();  
    }  
   }

the out put for this programme is



# 8. Kth Largest In Unsorted Array

package org.example;  
  
import java.util.\*;  
  
public class Main {  
  
 public static void main(String[] args) {  
 Scanner sc = new Scanner(System.*in*);  
 int n = sc.nextInt();  
 int[] arr = new int[n];  
 for (int i = 0; i < n; i++) arr[i] = sc.nextInt();  
 int k = sc.nextInt();  
 sc.close();  
 System.*out*.println("Kth Largest element in the array is "+*findKthLargest*(arr, k));  
 }  
  
 public static int findKthLargest(int[] nums, int k) {  
 // your code here  
 PriorityQueue<Integer> pq=new PriorityQueue<>( Collections.*reverseOrder*());  
 for(int i=0;i<nums.length;i++){  
 pq.add(nums[i]);  
 }  
  
 while(!pq.isEmpty()&&k>1){  
 pq.remove();  
 k--;  
 }  
 return pq.peek();  
 }  
}

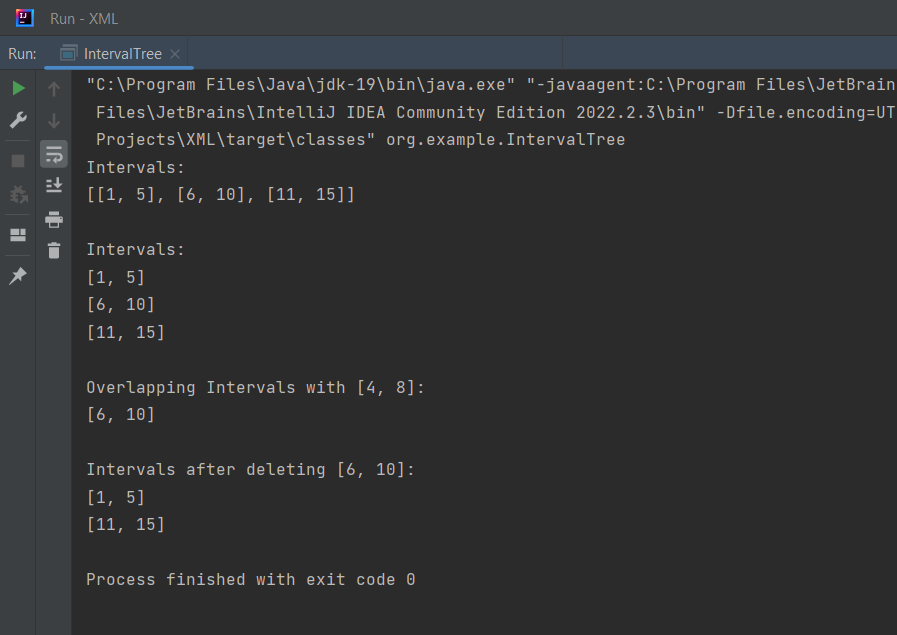
the output is

# 

# Intervals

1. import java.util.\*;  
     
   class Interval {  
    int start;  
    int end;  
     
    public Interval(int start, int end) {  
    this.start = start;  
    this.end = end;  
    }  
     
    @Override  
    public String toString() {  
    return "[" + start + ", " + end + "]";  
    }  
   }  
   class IntervalTree {  
    TreeMap<Integer, Interval> tree;  
     
    public IntervalTree() {  
    tree = new TreeMap<>();  
    }  
     
    // Insert a new interval [start, end] into the tree  
    public void insertInterval(int start, int end) {  
    Interval interval = new Interval(start, end);  
    tree.put(start, interval);  
    }  
     
    // Delete an interval [start, end] from the tree  
    public void deleteInterval(int start, int end) {  
    Iterator<Map.Entry<Integer, Interval>> iterator = tree.entrySet().iterator();  
    List<Integer> toRemove = new ArrayList<>();  
     
    while (iterator.hasNext()) {  
    Map.Entry<Integer, Interval> entry = iterator.next();  
    Interval interval = entry.getValue();  
    if (interval.start == start && interval.end == end) {  
    toRemove.add(entry.getKey());  
    }  
    }  
     
    for (int key : toRemove) {  
    tree.remove(key);  
    }  
    }  
     
    // Return a list of all intervals that overlap with the interval [start, end]  
    public List<Interval> findOverlappingIntervals(int start, int end) {  
    List<Interval> result = new ArrayList<>();  
     
    // Ceiling entry: Find the smallest entry >= start  
    Map.Entry<Integer, Interval> ceilingEntry = tree.ceilingEntry(start);  
    while (ceilingEntry != null) {  
    Interval interval = ceilingEntry.getValue();  
    if (interval.start > end) {  
    break;  
    }  
    if (interval.end >= start) {  
    result.add(interval);  
    }  
    ceilingEntry = tree.higherEntry(ceilingEntry.getKey());  
    }  
     
    return result;  
    }  
     
    public static void main(String[] args) {  
    IntervalTree intervalTree = new IntervalTree();  
     
    // Example usage  
    intervalTree.insertInterval(1, 5);  
    intervalTree.insertInterval(6, 10);  
    intervalTree.insertInterval(11, 15);  
     
    System.*out*.println("Intervals:");  
    System.*out*.println(intervalTree.tree.values()); // Printing TreeMap values directly (incorrect approach)  
     
    List<Interval> intervals = new ArrayList<>(intervalTree.tree.values());  
    System.*out*.println("\nIntervals:");  
    for (Interval interval : intervals) {  
    System.*out*.println(interval);  
    }  
     
    List<Interval> overlappingIntervals = intervalTree.findOverlappingIntervals(4, 8);  
    System.*out*.println("\nOverlapping Intervals with [4, 8]:");  
    for (Interval interval : overlappingIntervals) {  
    System.*out*.println(interval);  
    }  
     
    intervalTree.deleteInterval(6, 10);  
    System.*out*.println("\nIntervals after deleting [6, 10]:");  
    intervals = new ArrayList<>(intervalTree.tree.values());  
    for (Interval interval : intervals) {  
    System.*out*.println(interval);  
    }  
    }  
   }

The output is



# 10. Palindrome

package org.example;  
  
import java.util.\*;  
import java.lang.\*;  
import java.io.\*;  
  
public class Main {  
 public static void main(String[] args) throws java.lang.Exception {  
 //your code here  
 Scanner sc = new Scanner(System.*in*);  
 String str = sc.nextLine();  
 boolean ans = *palindromeOrNot*(str);  
 if (ans == true) {  
 System.*out*.println("The Given input is Palindrome ");  
 } else  
 System.*out*.println("The Given input is not Palindrome ");  
 }  
  
 public static boolean palindromeOrNot(String s) {  
  
 int i = 0;  
 int j = s.length() - 1;  
 while (i <= j) {  
 char c1 = s.charAt(i);  
 char c2 = s.charAt(j);  
 if (c1 != c2) {  
 return false;  
 }  
 i++;  
 j--;  
 }  
 return true;  
 }  
}

the out put is

