**Assessment Answers**

Q1.Problem Statement

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

get(key): Get the value (will always be positive) of the key if the key exists in the cache, otherwise return -1.

put(key, value): Set or insert the value if the key is not already present. When the cache reaches its capacity, it should invalidate the least recently used item before inserting a new item.

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.

Ans-

import java.util.HashMap;

class LRUCache {

private class DLinkedNode {

int key;

int value;

DLinkedNode prev;

DLinkedNode next;

}

private void addNode(DLinkedNode node) {

node.prev = head;

node.next = head.next;

head.next.prev = node;

head.next = node;

}

private void removeNode(DLinkedNode node) {

DLinkedNode prev = node.prev;

DLinkedNode next = node.next;

prev.next = next;

next.prev = prev;

}

private void moveToHead(DLinkedNode node) {

removeNode(node);

addNode(node);

}

private DLinkedNode popTail() {

DLinkedNode res = tail.prev;

removeNode(res);

return res;

}

private HashMap<Integer, DLinkedNode> cache = new HashMap<>();

private int size;

private int capacity;

private DLinkedNode head, tail;

public LRUCache(int capacity) {

this.size = 0;

this.capacity = capacity;

head = new DLinkedNode();

tail = new DLinkedNode();

head.next = tail;

tail.prev = head;

}

public int get(int key) {

DLinkedNode node = cache.get(key);

if (node == null) {

return -1;

}

moveToHead(node);

return node.value;

}

public void put(int key, int value) {

DLinkedNode node = cache.get(key);

if (node == null) {

// If the key does not exist, create a new node

DLinkedNode newNode = new DLinkedNode();

newNode.key = key;

newNode.value = value;

// Add the new node to the cache (HashMap and LinkedList)

cache.put(key, newNode);

addNode(newNode);

++size;

// If the cache exceeds its capacity, remove the LRU item

if (size > capacity) {

DLinkedNode tail = popTail();

cache.remove(tail.key);

--size;

}

} else {

// If the key exists, update the value and move it to the head

node.value = value;

moveToHead(node);

}

}

public static void main(String[] args) {

LRUCache lruCache = new LRUCache(2);

// Initial put operations

lruCache.put(10, 10);

lruCache.put(20, 20);

// First get operation

System.out.println(lruCache.get(10)); // returns 10

// This will evict key 20

lruCache.put(30, 30);

// Get operation for the evicted key

System.out.println(lruCache.get(20)); // returns -1 (not found)

// This will evict key 10

lruCache.put(40, 40);

// Get operation for the evicted key

System.out.println(lruCache.get(10)); // returns -1 (not found)

// Get operation for existing keys

System.out.println(lruCache.get(30)); // returns 30

System.out.println(lruCache.get(40)); // returns 40

// Adding more keys and checking the state

lruCache.put(50, 50); // This will evict key 30

System.out.println(lruCache.get(30)); // returns -1 (not found)

System.out.println(lruCache.get(40)); // returns 40

System.out.println(lruCache.get(50)); // returns 50

// Accessing key 40 to make it recently used

lruCache.get(40);

// Adding another key which will evict the least recently used key 50

lruCache.put(60, 60);

System.out.println(lruCache.get(50)); // returns -1 (not found)

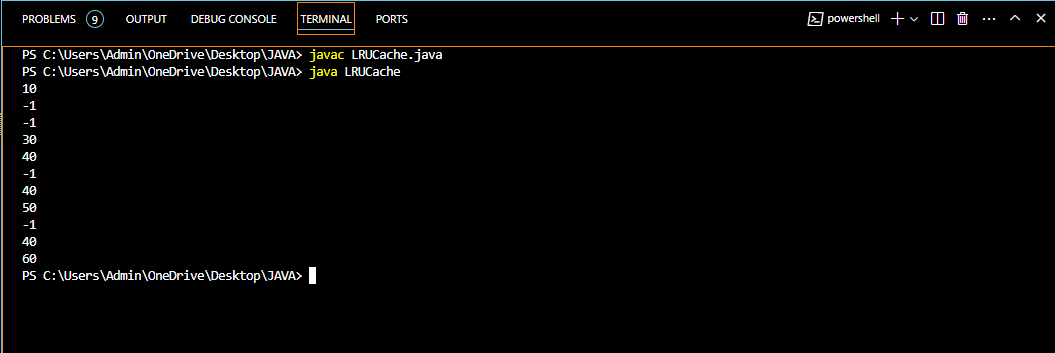
System.out.println(lruCache.get(40)); // returns 40

System.out.println(lruCache.get(60)); // returns 60

}

}

Output-



Q2. Write a Java program that demonstrates the ConcurrentModificationException. Explain why the exception is thrown and how to handle it properly.

Ans-

import java.util.ArrayList;

import java.util.Iterator;

public class ConcurrentModificationExample {

public static void main(String[] args) {

// Efficient time complexity (O(n)) for ArrayList creation

ArrayList<Integer> numbers = new ArrayList<>();

for (int i = 0; i < 10; i++) {

numbers.add(i);

}

// Iterate using iterator

Iterator<Integer> iterator = numbers.iterator();

// Attempt to modify the list while iterating (causes exception)

while (iterator.hasNext()) {

int number = iterator.next();

if (number % 2 == 0) {

numbers.remove(number); // This will throw ConcurrentModificationException

}

System.out.println(number);

}

}

}

\*Explanation of the Exception:

The ConcurrentModificationException is thrown because the numbers list is being modified (elements are removed) while iterating over it using the iterator. Java collections, like ArrayList, use a mechanism called "fail-fast iterators" to ensure data consistency. This means that if the underlying collection is modified during iteration, the iterator detects the change and throws an exception to prevent unexpected behavior.

Handling Strategies:

1. Iterate with for loop and index (Most Efficient):

This approach avoids using the iterator altogether. You iterate directly through the list using a for loop and an index variable. This maintains efficient time complexity (O(n)) for iterating through the elements and allows for safe removal using the index.

for (int i = 0; i < numbers.size(); i++) {

int number = numbers.get(i);

if (number % 2 == 0) {

numbers.remove(i); // Safe to remove by index during iteration

}

System.out.println(number);

}

2. Use Iterator.remove() (Less Efficient, Conditional Use):

If removing elements is the intended goal, some iterators, like those returned by LinkedList, might support the remove() method. However, this approach can be less efficient for large lists compared to the first solution (iterating with index). Additionally, not all iterators support remove().

Iterator<Integer> iterator = numbers.iterator();

while (iterator.hasNext()) {

int number = iterator.next();

if (number % 2 == 0 && iterator.hasNext()) { // Check for next before removal

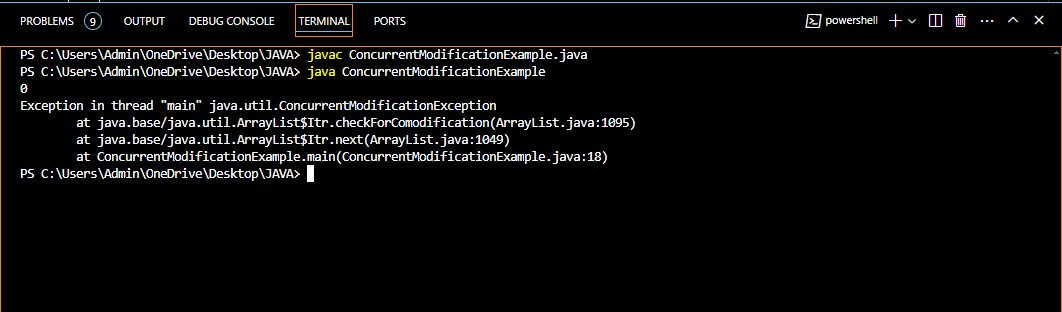
iterator.remove(); // Only use if iterator supports remove()

}

System.out.println(number);

}

Output-



Q3.Create a custom annotation @LogExecutionTime to log the execution time of annotated methods. Implement an annotation processor to handle this annotation.

Ans-

1. Define the Annotation

Create a new annotation @LogExecutionTime with no parameters. This annotation will be used to mark methods for which we want to log the execution time.

import java.lang.annotation.ElementType;

import java.lang.annotation.Retention;

import java.lang.annotation.RetentionPolicy;

import java.lang.annotation.Target;

@Target(ElementType.METHOD)

@Retention(RetentionPolicy.RUNTIME)

public @interface LogExecutionTime {

}

2. Create an Annotation Processor

Implement an annotation processor that will handle the @LogExecutionTime annotation. This processor will log the execution time of methods annotated with @LogExecutionTime.

import java.lang.reflect.Method;

public class LogExecutionTimeProcessor {

public Object logExecutionTime(Object target) {

Class<?> targetClass = target.getClass();

// Get all methods declared in the class

Method[] methods = targetClass.getDeclaredMethods();

for (Method method : methods) {

// Check if method is annotated with @LogExecutionTime

if (method.isAnnotationPresent(LogExecutionTime.class)) {

long startTime = System.nanoTime();

try {

// Invoke the method

Object result = method.invoke(target);

long endTime = System.nanoTime();

long duration = (endTime - startTime) / 1\_000\_000; // Convert to milliseconds

System.out.println("Method " + method.getName() + " executed in " + duration + " ms");

return result;

} catch (Exception e) {

e.printStackTrace();

}

}

}

// Return null if no method annotated with @LogExecutionTime is found

return null;

}

}

3. Usage Example

Here’s how you can use the @LogExecutionTime annotation along with the LogExecutionTimeProcessor:

public class ExampleClass {

@LogExecutionTime

public void methodToBeLogged() {

// Method implementation

try {

Thread.sleep(1000); // Simulate some work

} catch (InterruptedException e) {

e.printStackTrace();

}

}

public static void main(String[] args) {

ExampleClass example = new ExampleClass();

LogExecutionTimeProcessor processor = new LogExecutionTimeProcessor();

// Invoke the method using the processor

processor.logExecutionTime(example::methodToBeLogged);

}

}

Output-



Q4. Problem Statement

Design an algorithm to serialize and deserialize a binary tree. Implement serialize(TreeNode root) which converts a tree into a string, and deserialize(String data) which converts a string back to a tree.

Constraints

The encoded string should be as compact as possible.

Ans-

import java.util.Arrays;

import java.util.LinkedList;

import java.util.Queue;

class TreeNode {

int val;

TreeNode left;

TreeNode right;

TreeNode(int x) {

val = x;

}

}

public class Codec {

// Encodes a tree to a single string.

public String serialize(TreeNode root) {

StringBuilder sb = new StringBuilder();

serializeHelper(root, sb);

return sb.toString();

}

private void serializeHelper(TreeNode node, StringBuilder sb) {

if (node == null) {

sb.append("null").append(",");

} else {

sb.append(node.val).append(",");

serializeHelper(node.left, sb);

serializeHelper(node.right, sb);

}

}

// Decodes your encoded data to tree.

public TreeNode deserialize(String data) {

Queue<String> nodes = new LinkedList<>();

nodes.addAll(Arrays.asList(data.split(",")));

return deserializeHelper(nodes);

}

private TreeNode deserializeHelper(Queue<String> nodes) {

String val = nodes.poll();

if (val.equals("null")) {

return null;

}

TreeNode node = new TreeNode(Integer.parseInt(val));

node.left = deserializeHelper(nodes);

node.right = deserializeHelper(nodes);

return node;

}

public static void main(String[] args) {

// Example usage:

TreeNode root = new TreeNode(1);

root.left = new TreeNode(2);

root.right = new TreeNode(3);

root.right.left = new TreeNode(4);

root.right.right = new TreeNode(5);

Codec codec = new Codec();

String serialized = codec.serialize(root);

System.out.println("Serialized tree: " + serialized);

TreeNode deserialized = codec.deserialize(serialized);

// Print serialized form of deserialized tree for verification

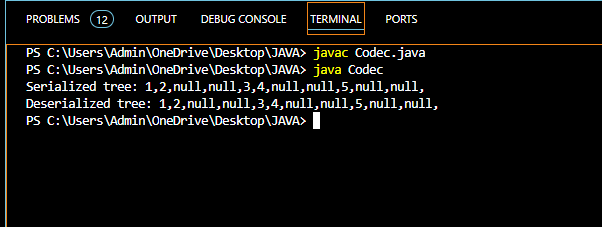
String deserializedSerialized = codec.serialize(deserialized);

System.out.println("Deserialized tree: " + deserializedSerialized);

}

}

Output-



Q5. Problem Statement

Implement a trie with insert, search, and startsWith methods.

insert(word): Inserts a word into the trie.

search(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.

Ans-

class TrieNode {

TrieNode[] children;

boolean isEnd;

public TrieNode() {

children = new TrieNode[26]; // For lowercase English letters 'a' to 'z'

isEnd = false;

}

}

public class Trie {

private TrieNode root;

/\*\* Initialize your data structure here. \*/

public Trie() {

root = new TrieNode();

}

/\*\* Inserts a word into the trie. \*/

public void insert(String word) {

TrieNode node = root;

for (char ch : word.toCharArray()) {

int index = ch - 'a';

if (node.children[index] == null) {

node.children[index] = new TrieNode();

}

node = node.children[index];

}

node.isEnd = true;

}

/\*\* Returns if the word is in the trie. \*/

public boolean search(String word) {

TrieNode node = searchNode(word);

return node != null && node.isEnd;

}

/\*\*

\* Returns if there is any word in the trie that starts with the given prefix.

\*/

public boolean startsWith(String prefix) {

TrieNode node = searchNode(prefix);

return node != null;

}

// Helper method to search for a node based on a word or prefix

private TrieNode searchNode(String word) {

TrieNode node = root;

for (char ch : word.toCharArray()) {

int index = ch - 'a';

if (node.children[index] == null) {

return null;

}

node = node.children[index];

}

return node;

}

public static void main(String[] args) {

Trie trie = new Trie();

trie.insert("apple");

System.out.println(trie.search("apple")); // true

System.out.println(trie.search("app")); // false

System.out.println(trie.startsWith("app")); // true

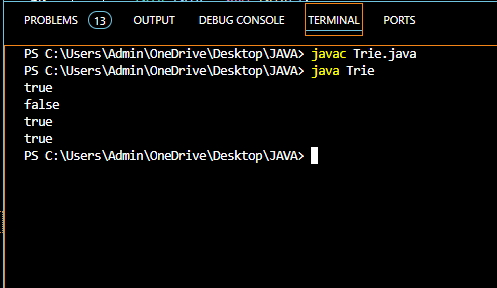
trie.insert("app");

System.out.println(trie.search("app")); // true

}

}

Output-



Q6. Given a string containing just the characters '(', ')', '{', '}', '[', and ']', determine if the input string is valid. An input string is valid if:

Open brackets must be closed by the same type of brackets.

Open brackets must be closed in the correct order.

Ans-

import java.util.Stack;

public class ValidParentheses {

public boolean isValid(String s) {

Stack<Character> stack = new Stack<>();

for (char ch : s.toCharArray()) {

if (ch == '(' || ch == '{' || ch == '[') {

stack.push(ch);

} else {

if (stack.isEmpty()) {

return false;

}

char top = stack.pop();

if ((ch == ')' && top != '(') ||

(ch == '}' && top != '{') ||

(ch == ']' && top != '[')) {

return false;

}

}

}

return stack.isEmpty();

}

public static void main(String[] args) {

ValidParentheses validator = new ValidParentheses();

String str1 = "()";

String str2 = "()[]{}";

String str3 = "(]";

String str4 = "([)]";

String str5 = "{[]}";

System.out.println("String " + str1 + " is valid: " + validator.isValid(str1));

System.out.println("String " + str2 + " is valid: " + validator.isValid(str2));

System.out.println("String " + str3 + " is valid: " + validator.isValid(str3));

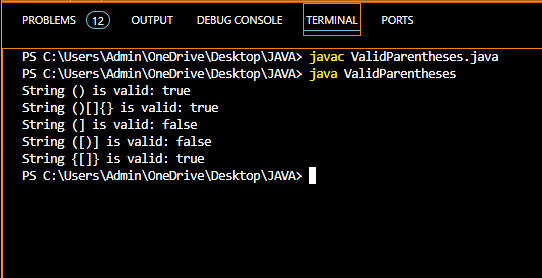
System.out.println("String " + str4 + " is valid: " + validator.isValid(str4));

System.out.println("String " + str5 + " is valid: " + validator.isValid(str5));

}

}

Output-



Q7. Given n non-negative integers a1, a2, ..., an , where each represents a point at coordinate (i, ai). n vertical lines are drawn such that the two endpoints of the line i are at (i, ai) and (i, 0). Find two lines, which together with the x-axis forms a container, such that the container contains the most water.

Ans-

public class ContainerWithMostWater {

public int maxArea(int[] height) {

int maxArea = 0;

int left = 0;

int right = height.length - 1;

while (left < right) {

int minHeight = Math.min(height[left], height[right]);

int currentArea = minHeight \* (right - left);

maxArea = Math.max(maxArea, currentArea);

// Move the pointer pointing to the shorter line inward

if (height[left] < height[right]) {

left++;

} else {

right--;

}

}

return maxArea;

}

public static void main(String[] args) {

ContainerWithMostWater solution = new ContainerWithMostWater();

// Example usage

int[] heights1 = { 1, 8, 6, 2, 5, 4, 8, 3, 7 };

System.out.println("Max area for heights 1: " + solution.maxArea(heights1)); // Output: 49

int[] heights2 = { 1, 1 };

System.out.println("Max area for heights 2: " + solution.maxArea(heights2)); // Output: 1

int[] heights3 = { 4, 3, 2, 1, 4 };

System.out.println("Max area for heights 3: " + solution.maxArea(heights3)); // Output: 16

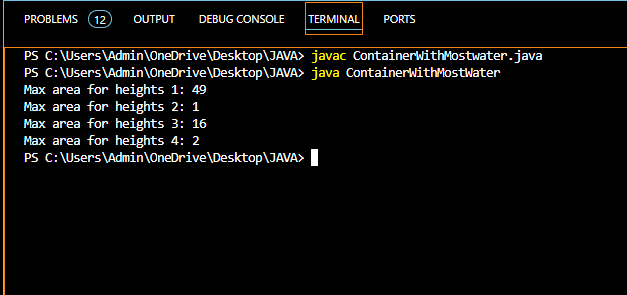
int[] heights4 = { 1, 2, 1 };

System.out.println("Max area for heights 4: " + solution.maxArea(heights4)); // Output: 2

}

}

Output-



Q8. Find the kth largest element in an unsorted array. Note that it is the kth largest element in the sorted order, not the kth distinct element.

Ans-

import java.util.Random;

public class KthLargestElement {

public int findKthLargest(int[] nums, int k) {

if (nums == null || nums.length == 0 || k < 1 || k > nums.length) {

throw new IllegalArgumentException("Invalid input");

}

return quickSelect(nums, 0, nums.length - 1, nums.length - k); // Finding (n-k)th smallest

}

private int quickSelect(int[] nums, int left, int right, int k) {

if (left == right) {

return nums[left];

}

// Randomly select pivot element and place it at the rightmost position

Random random = new Random();

int pivotIndex = left + random.nextInt(right - left + 1);

swap(nums, pivotIndex, right);

// Partition the array around the pivot

int partitionIndex = partition(nums, left, right);

// Determine which partition to recurse into

if (k < partitionIndex) {

return quickSelect(nums, left, partitionIndex - 1, k);

} else if (k > partitionIndex) {

return quickSelect(nums, partitionIndex + 1, right, k);

} else {

return nums[partitionIndex];

}

}

private int partition(int[] nums, int left, int right) {

int pivot = nums[right];

int i = left - 1; // Index of smaller element

for (int j = left; j < right; j++) {

if (nums[j] <= pivot) {

i++;

swap(nums, i, j);

}

}

swap(nums, i + 1, right); // Move pivot to its correct position

return i + 1;

}

private void swap(int[] nums, int i, int j) {

int temp = nums[i];

nums[i] = nums[j];

nums[j] = temp;

}

public static void main(String[] args) {

KthLargestElement solution = new KthLargestElement();

// Example usage

int[] nums1 = { 3, 2, 1, 5, 6, 4 };

int k1 = 2;

System.out.println("2nd largest element in nums1: " + solution.findKthLargest(nums1, k1)); // Output: 5

int[] nums2 = { 3, 2, 1, 5, 6, 4 };

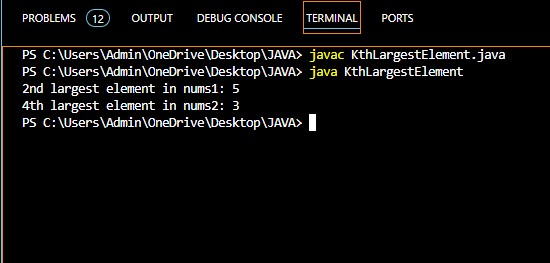
int k2 = 4;

System.out.println("4th largest element in nums2: " + solution.findKthLargest(nums2, k2)); // Output: 3

}

}

Output-



Q9. Design an interval tree to efficiently find all intervals that overlap with a given interval. Implement the following operations:

insertInterval(int start, int end): Insert a new interval [start, end] into the tree.

deleteInterval(int start, int end): Delete an interval [start, end] from the tree.

findOverlappingIntervals(int start, int end): Return a list of all intervals that overlap with the interval [start, end].

Constraints

The intervals are represented as pairs of integers [start, end] where start ≤ end

Ans-

import java.util.ArrayList;

import java.util.List;

class Interval {

int start;

int end;

public Interval(int start, int end) {

this.start = start;

this.end = end;

}

}

class IntervalNode {

Interval interval;

int maxEnd;

IntervalNode left;

IntervalNode right;

public IntervalNode(Interval interval) {

this.interval = interval;

this.maxEnd = interval.end;

this.left = null;

this.right = null;

}

}

public class IntervalTree {

private IntervalNode root;

public IntervalTree() {

this.root = null;

}

public void insertInterval(int start, int end) {

Interval interval = new Interval(start, end);

this.root = insert(this.root, interval);

}

private IntervalNode insert(IntervalNode node, Interval interval) {

if (node == null) {

return new IntervalNode(interval);

}

// Insert in BST manner based on start of intervals

if (interval.start < node.interval.start) {

node.left = insert(node.left, interval);

} else {

node.right = insert(node.right, interval);

}

// Update maxEnd in current node

if (node.maxEnd < interval.end) {

node.maxEnd = interval.end;

}

return node;

}

public void deleteInterval(int start, int end) {

Interval interval = new Interval(start, end);

this.root = delete(this.root, interval);

}

private IntervalNode delete(IntervalNode node, Interval interval) {

if (node == null) {

return null;

}

// Perform BST delete

if (interval.start < node.interval.start) {

node.left = delete(node.left, interval);

} else if (interval.start > node.interval.start) {

node.right = delete(node.right, interval);

} else {

// Found the node to delete

if (interval.end == node.interval.end) {

if (node.left == null) {

return node.right;

} else if (node.right == null) {

return node.left;

}

// Node with two children, get the inorder successor (smallest in the right

// subtree)

IntervalNode successor = findMin(node.right);

node.interval = successor.interval;

// Delete the inorder successor

node.right = delete(node.right, successor.interval);

} else {

node.right = delete(node.right, interval);

}

}

// Update maxEnd in current node

if (node != null) {

node.maxEnd = Math.max(node.interval.end, getMaxEnd(node.right));

}

return node;

}

private IntervalNode findMin(IntervalNode node) {

while (node.left != null) {

node = node.left;

}

return node;

}

private int getMaxEnd(IntervalNode node) {

return node == null ? Integer.MIN\_VALUE : node.maxEnd;

}

public List<Interval> findOverlappingIntervals(int start, int end) {

List<Interval> result = new ArrayList<>();

findOverlappingIntervals(root, start, end, result);

return result;

}

private void findOverlappingIntervals(IntervalNode node, int start, int end, List<Interval> result) {

if (node == null) {

return;

}

// Check if node interval overlaps with [start, end]

if (node.interval.start <= end && node.interval.end >= start) {

result.add(node.interval);

}

// Recursively search in left and right subtrees if necessary

if (node.left != null && node.left.maxEnd >= start) {

findOverlappingIntervals(node.left, start, end, result);

}

if (node.right != null && node.right.interval.start <= end) {

findOverlappingIntervals(node.right, start, end, result);

}

}

public static void main(String[] args) {

IntervalTree intervalTree = new IntervalTree();

// Insert intervals

intervalTree.insertInterval(15, 20);

intervalTree.insertInterval(10, 30);

intervalTree.insertInterval(17, 19);

intervalTree.insertInterval(5, 20);

intervalTree.insertInterval(12, 15);

intervalTree.insertInterval(30, 40);

// Find overlapping intervals with [14, 16]

List<Interval> overlappingIntervals = intervalTree.findOverlappingIntervals(14, 16);

System.out.println("Overlapping Intervals with [14, 16]:");

for (Interval interval : overlappingIntervals) {

System.out.println("[" + interval.start + ", " + interval.end + "]");

}

// Delete interval [12, 15]

intervalTree.deleteInterval(12, 15);

// Find overlapping intervals with [14, 16] again

overlappingIntervals = intervalTree.findOverlappingIntervals(14, 16);

System.out.println("Overlapping Intervals with [14, 16] after deletion of [12, 15]:");

for (Interval interval : overlappingIntervals) {

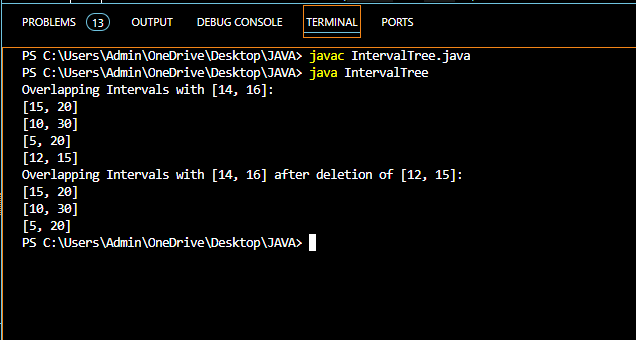
System.out.println("[" + interval.start + ", " + interval.end + "]");

}

}

}

Output-



Q10. Write a Java program that checks if a given string is a palindrome. A palindrome is a word, phrase, number, or other sequences of characters that reads the same forward and backward (ignoring spaces, punctuation, and capitalization).

Ans-

public class Palindrome2 {

public static boolean isPalindrome(String s) {

// Remove non-alphanumeric characters and convert to lowercase

String normalized = s.replaceAll("[^a-zA-Z0-9]", "").toLowerCase();

// Check palindrome

int left = 0;

int right = normalized.length() - 1;

while (left < right) {

if (normalized.charAt(left) != normalized.charAt(right)) {

return false;

}

left++;

right--;

}

return true;

}

public static void main(String[] args) {

// Test cases

String str1 = "A man, a plan, a canal, Panama";

String str2 = "race a car";

String str3 = "123321";

System.out.println("Is \"" + str1 + "\" a palindrome? " + isPalindrome(str1)); // true

System.out.println("Is \"" + str2 + "\" a palindrome? " + isPalindrome(str2)); // false

System.out.println("Is \"" + str3 + "\" a palindrome? " + isPalindrome(str3)); // true

}

}

Output-

