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

**Code.**

class LRUCache {

class Node{

int key;

int value;

Node prev;

Node next;

Node(int key, int value){

this.key= key;

this.value= value;

}

}

public Node[] map;

public int count, capacity;

public Node head, tail;

public LRUCache(int capacity) {

this.capacity= capacity;

count= 0;

map= new Node[10\_000+1];

head= new Node(0,0);

tail= new Node(0,0);

head.next= tail;

tail.prev= head;

head.prev= null;

tail.next= null;

}

public void deleteNode(Node node){

node.prev.next= node.next;

node.next.prev= node.prev;

return;

}

public void addToHead(Node node){

node.next= head.next;

node.next.prev= node;

node.prev= head;

head.next= node;

return;

}

public int get(int key) {

if( map[key] != null ){

Node node= map[key];

int nodeVal= node.value;

deleteNode(node);

addToHead(node);

return nodeVal;

}

else

return -1;

}

public void put(int key, int value) {

if(map[key] != null){

Node node= map[key];

node.value= value;

deleteNode(node);

addToHead(node);

} else {

Node node= new Node(key,value);

map[key]= node;

if(count < capacity){

count++;

addToHead(node);

}

else {

map[tail.prev.key]= null;

deleteNode(tail.prev);

addToHead(node);

}

}

return;

}

}

public class TestLRUCache {

public static void main(String[] args)

{

LRUCache cache = new LRUCache(2);

cache.put(1, 10);

cache.put(2, 20);

System.out.println("Value for the key: 1 is " + cache.get(1));

cache.put(3, 30);

System.out.println("Value for the key: 2 is " + cache.get(2));

cache.put(4, 40);

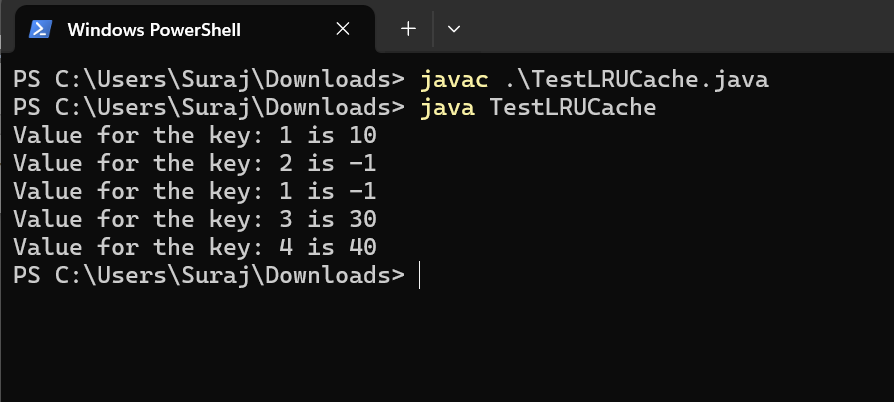
System.out.println("Value for the key: 1 is " + cache.get(1));

System.out.println("Value for the key: 3 is " + cache.get(3));

System.out.println("Value for the key: 4 is " + cache.get(4));

}

}



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

**Code:**

The ConcurrentModificationException is thrown when a collection is modified while it is being iterated over by an iterator.

**Example.**

import java.util.\*;

public class ConcurrentModificationExceptionDemo {

public static void main(String[] args) {

// Create a list and add some elements

List<String> list = new ArrayList<>();

list.add("One");

list.add("Two");

list.add("Three");

list.add("Four");

System.out.println("Original List: " + list);

try {

// Obtain an iterator for the list

Iterator<String> iterator = list.iterator();

// Iterate through the list and modify it at the same time

while (iterator.hasNext()) {

String item = iterator.next();

System.out.println("Current Item: " + item);

System.out.println("\n\nTrying to add" + " an element in " + "between iteration\n");

list.add("Five");

}

} catch (ConcurrentModificationException e) {

System.out.println("Exception caught: " + e);

}

System.out.println("List after modification attempt: " + list);

}

}

3. 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.

**Code.**

// A Java program to demonstrate serialization and

// deserialization of Binary Tree

import java.util.\*;

// A binary tree Node has key,

// pointer to left and right children

class TreeNode {

int val;

TreeNode left;

TreeNode right;

TreeNode(int x) { val = x; }

}

class BinaryTree {

TreeNode root;

// Encodes a tree to a single string.

public static String serialize(TreeNode root)

{

if (root == null) {

return null;

}

Stack<TreeNode> s = new Stack<>();

s.push(root);

List<String> l = new ArrayList<>();

while (!s.isEmpty()) {

TreeNode t = s.pop();

// If current node is NULL, store marker

if (t == null) {

l.add("#");

}

else {

// Else, store current node

// and recur for its children

l.add("" + t.val);

s.push(t.right);

s.push(t.left);

}

}

return String.join(",", l);

}

static int t;

// Decodes your encoded data to tree.

public static TreeNode deserialize(String data)

{

if (data == null)

return null;

t = 0;

String[] arr = data.split(",");

return helper(arr);

}

public static TreeNode helper(String[] arr)

{

if (arr[t].equals("#"))

return null;

// Create node with this item

// and recur for children

TreeNode root

= new TreeNode(Integer.parseInt(arr[t]));

t++;

root.left = helper(arr);

t++;

root.right = helper(arr);

return root;

}

// A simple inorder traversal used

// for testing the constructed tree

static void inorder(TreeNode root)

{

if (root != null) {

inorder(root.left);

System.out.print(root.val + " ");

inorder(root.right);

}

}

// Driver code

public static void main(String args[])

{

// Construct a tree shown in the above figure

BinaryTree tree = new BinaryTree();

tree.root = new TreeNode(20);

tree.root.left = new TreeNode(8);

tree.root.right = new TreeNode(22);

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

tree.root.left.right = new TreeNode(12);

tree.root.left.right.left = new TreeNode(10);

tree.root.left.right.right = new TreeNode(14);

String serialized = serialize(tree.root);

System.out.println("Serialized view of the tree:");

System.out.println(serialized);

System.out.println();

// Deserialize the stored tree into root1

TreeNode t = deserialize(serialized);

System.out.println(

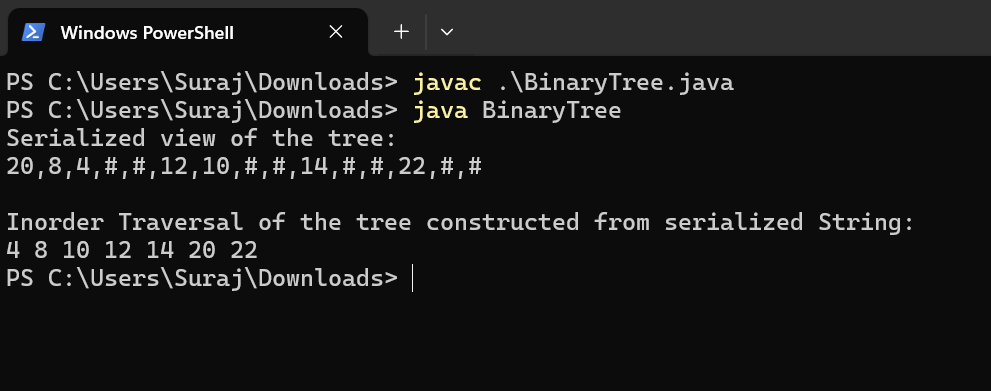
"Inorder Traversal of the tree constructed"

+ " from serialized String:");

inorder(t);

}

}



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

**Code.**

6. 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.

**Code**.

import java.util.\*;

public class ValidParentheses {

static boolean areBracketsBalanced(String s)

{

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

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

if (c == '(')

stack.push(')');

else if (c == '{')

stack.push('}');

else if (c == '[')

stack.push(']');

else if (stack.isEmpty() || stack.pop() != c)

return false;

}

return stack.isEmpty();

}

public static void main(String[] args)

{

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

if (areBracketsBalanced(expr))

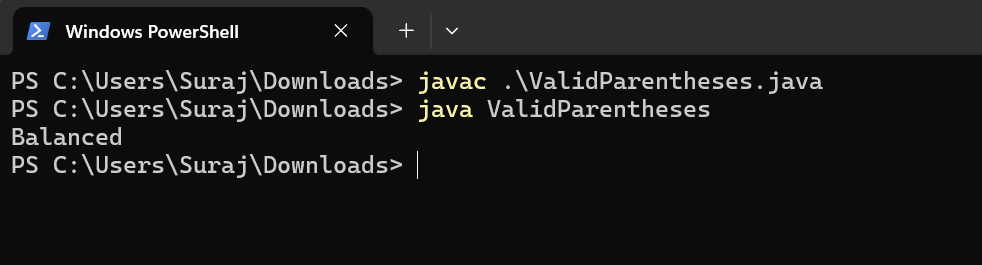
System.out.println("Balanced ");

else

System.out.println("Not Balanced ");

}

}



10. 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).

Code.

public class PalindromeChecker {

public static void main(String[] args) {

String[] testStrings = {"madam", "racecar", "hello", "A man, a plan, a canal, Panama", "No 'x' in Nixon", "Was it a car or a cat I saw?", "Yo, Banana Boy!", "Red roses run no risk, sir, on Nurse’s order."};

for (String testString : testStrings) {

if (isPalindrome(testString)) {

System.out.println("\"" + testString + "\" is a palindrome.");

} else {

System.out.println("\"" + testString + "\" is not a palindrome.");

}

}

}

public static boolean isPalindrome(String str) {

// Remove non-alphanumeric characters and convert to lowercase

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

int left = 0;

int right = cleanedStr.length() - 1;

while (left < right) {

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

return false; // Not a palindrome

}

left++;

right--;

}

return true; // It is a palindrome

}

}

