**Level1**

What is the difference between the throw and throws keywords in Java?

is used to throw exceptions explicitly within your code, while **throws** is used to declare the exceptions that a method may throw, which helps the caller of the method know which exceptions to handle.

What is the purpose of the finalize() method in Java? Why is it considered a bad practice to use it?

The **finalize**  method was used for resource cleanup in earlier Java versions, but it is now considered bad practice due to its unreliable timing and the availability of more modern and reliable alternatives like try-with-resources.

What is the difference between an abstract class and an interface in Java? When would you use one over the other?

* Abstract classes can have instance variables (fields), constructors, and method implementations Subclasses of an abstract class must either provide concrete implementations for all of its abstract methods or be declared as abstract themselves.

What is a Lambda expression in Java? Provide an example to explain its use.

This defines the implementation of the lambda expression. It can be a single expression or a block of statements.

What is the difference between the continue and break statements in Java?

the statement is used to exit a loop or switch statement prematurely, while the statement is used to skip the remaining statements in the current iteration of a loop and move to the next iteration. Both statements are valuable for controlling the flow of loops in Java and can be used to implement specific behaviors based on conditions.

What is the purpose of the volatile keyword in Java? Provide an example to explain its use.

Lambda expressions are typically used when working with functional interfaces, which are interfaces with a single abstract method (SAM). Lambda expressions can be used to provide a concise implementation for that single abstract method.

What is the difference between ArrayList and LinkedList in Java? When would you use one over the other?

**ArrayList** is more efficient for random access but slower for inserting elements in the middle. On the other hand, **LinkedList** is faster for insertions and deletions in the middle but slower for random access. Your choice between them should depend on your specific use case and the operations you need to perform most frequently.

What is the difference between public, protected, and private access modifiers in Java?

It's important to note that the default (package-private) access modifier is not mentioned here. If no access modifier is specified, the member is accessible within the same package but not from outside the package.

What is an exception in Java? Provide an example to explain the concept.

The **finally** block is used for cleanup code that should be executed whether an exception occurred or not.

What is the purpose of the static keyword in Java? Provide an example to explain its use.

is used to declare members (fields, methods, and nested classes) that belong to the class rather than to instances of the class. This means that a static member is associated with the class itself, not with any specific instance of the class. Here are the main uses of the **static** keyword:

What is a constructor in Java? Provide an example to explain its use.

A constructor is a special type of method that is used to initialize objects. It is called when an object of a class is created, and it has the same name as the class. Constructors are primarily used to set initial values for the object's attributes or perform any necessary setup during object creation.

What is the purpose of the interface keyword in Java? Provide an example to explain its use.

is used to declare an interface, which is a collection of abstract methods. An interface in Java is similar to a class, but it only contains method signatures (declarations) without method bodies. It defines a contract that classes implementing the interface must adhere to by providing concrete implementations for all the methods declared in the interface.

What is a thread in Java? Provide an example to explain the concept.

n Java, a thread is the smallest unit of execution within a process. It is a lightweight, independent path of execution that runs concurrently with other threads. Threads in Java are used to achieve parallelism and to perform multiple tasks concurrently. Java provides built-in support for multithreading through the **Thread** class and the **Runnable** interface.

What is the purpose of the super keyword in Java? Provide an example to explain its use.

keyword is used to refer to the immediate parent class object. It is particularly useful in situations where a subclass has overridden a method or shadowed a field, and you want to explicitly call the superclass version of the method or access the superclass field.

What is a package in Java? Provide an example to explain the concept.

In Java, a package is a way to organize related classes and interfaces into a single directory hierarchy. It helps in grouping related types, providing a namespace for the types contained in the package, and preventing naming conflicts.

What is a method in Java? Provide an example to explain the concept.

In Java, a method is a collection of statements that perform a specific task and is defined within a class. Methods in Java are used to define the behavior of objects, encapsulate functionality, and promote code reuse. They are invoked or called to perform certain operations.

What is the difference between a HashMap and a TreeMap in Java? When would you use one over the other?

**HashMap:** Does not guarantee any specific order of the elements. The order of the elements is not guaranteed to be the same as the order in which they were added to the map. It uses the hash code of the keys to determine the storage location and does not maintain any order based on the keys.

**reeMap:** Has a higher time complexity for basic operations compared to **HashMap**. The time complexity for operations is O(log n), where n is the number of elements in the map.

What is the purpose of the assert keyword in Java? Provide an example to explain its use

The **assert** keyword in Java is used to perform runtime testing and debugging. It helps developers express certain assumptions about code in a concise way and is used to check whether those assumptions hold true. If an assertion fails (evaluates to false), an **AssertionError** is thrown, typically terminating the program or triggering some custom handling.

What is the difference between final, finally, and finalize keywords in Java?

The **final** keyword is used in different contexts to define constants, make a class not extendable (final class), make a method not overrideable (final method), or prevent modification of variables (final variables).

What is polymorphism in Java? Explain with an example.

In Java, polymorphism refers to the ability of a single method, class, or interface to work in different ways or to take on multiple forms. There are two types of polymorphism in Java: compile-time (method overloading) and runtime (method overriding).

**10 Programming Questions**

Write a program to implement a binary search tree in Java.

public class BinarySearchTree {

// Node class for the binary search tree

static class Node {

int key;

Node left, right;

public Node(int item) {

key = item;

left = right = null;

}

}

// Root of the binary search tree

Node root;

// Constructor

BinarySearchTree() {

root = null;

}

// Method to insert a key into the binary search tree

void insert(int key) {

root = insertRec(root, key);

}

// Recursive helper method to insert a key into the binary search tree

Node insertRec(Node root, int key) {

// If the tree is empty, create a new node

if (root == null) {

root = new Node(key);

return root;

}

// Otherwise, traverse the tree

if (key < root.key) {

// Recur for the left subtree

root.left = insertRec(root.left, key);

} else if (key > root.key) {

// Recur for the right subtree

root.right = insertRec(root.right, key);

}

// Return the updated root

return root;

}

// Method to perform in-order traversal of the binary search tree

void inOrderTraversal(Node root) {

if (root != null) {

inOrderTraversal(root.left);

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

inOrderTraversal(root.right);

}

}

// Main method to test the binary search tree

public static void main(String[] args) {

BinarySearchTree tree = new BinarySearchTree();

// Inserting keys into the tree

tree.insert(50);

tree.insert(30);

tree.insert(20);

tree.insert(40);

tree.insert(70);

tree.insert(60);

tree.insert(80);

// Performing in-order traversal

System.out.println("In-order traversal of the binary search tree:");

tree.inOrderTraversal(tree.root);

}

}

Write a program to implement a doubly-linked list in Java.

class Node {

int data;

Node prev;

Node next;

public Node(int data) {

this.data = data;

this.prev = null;

this.next = null;

}

}

class DoublyLinkedList {

private Node head;

private Node tail;

public DoublyLinkedList() {

this.head = null;

this.tail = null;

}

public void insertAtEnd(int data) {

Node newNode = new Node(data);

if (head == null) {

head = newNode;

tail = newNode;

} else {

tail.next = newNode;

newNode.prev = tail;

tail = newNode;

}

}

public void display() {

Node current = head;

while (current != null) {

System.out.print(current.data + " ");

current = current.next;

}

System.out.println();

}

public void reverseDisplay() {

Node current = tail;

while (current != null) {

System.out.print(current.data + " ");

current = current.prev;

}

System.out.println();

}

public static void main(String[] args) {

DoublyLinkedList dll = new DoublyLinkedList();

dll.insertAtEnd(1);

dll.insertAtEnd(2);

dll.insertAtEnd(3);

System.out.println("Doubly Linked List:");

dll.display();

System.out.println("Reversed Doubly Linked List:");

dll.reverseDisplay();

}

}

Write a program to sort an array of objects in Java using the quicksort algorithm.

import java.util.Arrays;

class Person implements Comparable<Person> {

private String name;

private int age;

public Person(String name, int age) {

this.name = name;

this.age = age;

}

// Implementing compareTo method for comparing Persons based on their age

@Override

public int compareTo(Person other) {

return Integer.compare(this.age, other.age);

}

@Override

public String toString() {

return "Person{name='" + name + "', age=" + age + '}';

}

}

public class QuickSortExample {

public static <T extends Comparable<T>> void quickSort(T[] array, int low, int high) {

if (low < high) {

int partitionIndex = partition(array, low, high);

quickSort(array, low, partitionIndex - 1);

quickSort(array, partitionIndex + 1, high);

}

}

private static <T extends Comparable<T>> int partition(T[] array, int low, int high) {

T pivot = array[high];

int i = low - 1;

for (int j = low; j < high; j++) {

if (array[j].compareTo(pivot) <= 0) {

i++;

swap(array, i, j);

}

}

swap(array, i + 1, high);

return i + 1;

}

private static <T> void swap(T[] array, int i, int j) {

T temp = array[i];

array[i] = array[j];

array[j] = temp;

}

public static void main(String[] args) {

Person[] people = {

new Person("Alice", 25),

new Person("Bob", 30),

new Person("Charlie", 20),

new Person("David", 35),

new Person("Eva", 28)

};

System.out.println("Before

Write a program to implement a stack using a linked list in Java.

class Node {

int data;

Node next;

public Node(int data) {

this.data = data;

this.next = null;

}

}

class Stack {

private Node top;

public Stack() {

this.top = null;

}

public boolean isEmpty() {

return top == null;

}

public void push(int data) {

Node newNode = new Node(data);

newNode.next = top;

top = newNode;

}

public int pop() {

if (isEmpty()) {

throw new IllegalStateException("Stack is empty");

}

int data = top.data;

top = top.next;

return data;

}

public int peek() {

if (isEmpty()) {

throw new IllegalStateException("Stack is empty");

}

return top.data;

}

public void display() {

Node current = top;

while (current != null) {

System.out.print(current.data + " ");

current = current.next;

}

System.out.println();

}

}

public class StackUsingLinkedList {

public static void main(String[] args) {

Stack stack = new Stack();

stack.push(1);

stack.push(2);

stack.push(3);

System.out.println("Stack elements:");

stack.display();

System.out.println("Peek: " + stack.peek());

System.out.println("Pop: " + stack.pop());

System.out.println("Stack elements after pop:");

stack.display();

}

}

Write a program to implement a queue using two stacks in Java.

import java.util.Stack;

public class QueueUsingTwoStacks {

// Two stacks to implement the queue

private Stack<Integer> stack1 = new Stack<>();

private Stack<Integer> stack2 = new Stack<>();

// Enqueue operation (push into stack1)

public void enqueue(int item) {

stack1.push(item);

}

// Dequeue operation

public int dequeue() {

// If both stacks are empty, the queue is empty

if (isEmpty()) {

throw new IllegalStateException("Queue is empty");

}

// If stack2 is empty, transfer elements from stack1 to stack2

if (stack2.isEmpty()) {

while (!stack1.isEmpty()) {

stack2.push(stack1.pop());

}

}

// Pop the front element from stack2

return stack2.pop();

}

// Check if the queue is empty

public boolean isEmpty() {

return stack1.isEmpty() && stack2.isEmpty();

}

// Get the size of the queue

public int size() {

return stack1.size() + stack2.size();

}

public static void main(String[] args) {

QueueUsingTwoStacks queue = new QueueUsingTwoStacks();

// Enqueue elements

queue.enqueue(1);

queue.enqueue(2);

queue.enqueue(3);

// Dequeue elements

System.out.println("Dequeued element: " + queue.dequeue());

System.out.println("Dequeued element: " + queue.dequeue());

// Enqueue more elements

queue.enqueue(4);

queue.enqueue(5);

// Dequeue the remaining elements

while (!queue.isEmpty()) {

System.out.println("Dequeued element: " + queue.dequeue());

}

}

}

Write a program to find the longest common subsequence of two strings in Java.

public class LongestCommonSubsequence {

public static String findLCS(String str1, String str2) {

int m = str1.length();

int n = str2.length();

// Create a 2D array to store the lengths of LCS

int[][] dp = new int[m + 1][n + 1];

// Build the dp array using bottom-up approach

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

for (int j = 0; j <= n; j++) {

if (i == 0 || j == 0) {

dp[i][j] = 0; // Base case: an empty string has no common subsequence

} else if (str1.charAt(i - 1) == str2.charAt(j - 1)) {

dp[i][j] = dp[i - 1][j - 1] + 1;

} else {

dp[i][j] = Math.max(dp[i - 1][j], dp[i][j - 1]);

}

}

}

// Reconstruct the LCS from the dp array

int len = dp[m][n];

char[] lcs = new char[len];

int i = m, j = n;

while (i > 0 && j > 0) {

if (str1.charAt(i - 1) == str2.charAt(j - 1)) {

lcs[len - 1] = str1.charAt(i - 1);

i--;

j--;

len--;

} else if (dp[i - 1][j] > dp[i][j - 1]) {

i--;

} else {

j--;

}

}

return new String(lcs);

}

public static void main(String[] args) {

String str1 = "ABCBDAB";

String str2 = "BDCAB";

String lcs = findLCS(str1, str2);

System.out.println("Longest Common Subsequence: " + lcs);

}

}

Write a program to find the maximum subarray sum in an array of integers in Java.

public class MaximumSubarraySum {

public static int maxSubArraySum(int[] nums) {

int maxEndingHere = nums[0];

int maxSoFar = nums[0];

for (int i = 1; i < nums.length; i++) {

maxEndingHere = Math.max(nums[i], maxEndingHere + nums[i]);

maxSoFar = Math.max(maxSoFar, maxEndingHere);

}

return maxSoFar;

}

public static void main(String[] args) {

int[] nums = {-2, 1, -3, 4, -1, 2, 1, -5, 4};

int maxSum = maxSubArraySum(nums);

System.out.println("Maximum Subarray Sum: " + maxSum);

}

}

Write a program to implement a merge sort algorithm in Java.

public class MergeSort {

public static void mergeSort(int[] arr) {

if (arr == null || arr.length <= 1) {

return;

}

int length = arr.length;

int[] temp = new int[length];

mergeSortHelper(arr, temp, 0, length - 1);

}

private static void mergeSortHelper(int[] arr, int[] temp, int left, int right) {

if (left < right) {

int mid = left + (right - left) / 2;

mergeSortHelper(arr, temp, left, mid);

mergeSortHelper(arr, temp, mid + 1, right);

merge(arr, temp, left, mid, right);

}

}

private static void merge(int[] arr, int[] temp, int left, int mid, int right) {

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

temp[i] = arr[i];

}

int i = left;

int j = mid + 1;

int k = left;

while (i <= mid && j <= right) {

if (temp[i] <= temp[j]) {

arr[k++] = temp[i++];

} else {

arr[k++] = temp[j++];

}

}

while (i <= mid) {

arr[k++] = temp[i++];

}

}

public static void main(String[] args) {

int[] arr = {12, 11, 13, 5, 6, 7};

System.out.println("Original array:");

printArray(arr);

mergeSort(arr);

System.out.println("\nArray after Merge Sort:");

printArray(arr);

}

private static void printArray(int[] arr) {

for (int value : arr) {

System.out.print(value + " ");

}

System.out.println();

}

}

Write a program to implement a binary search algorithm for a rotated sorted array in Java.

public class RotatedSortedArraySearch {

public static int search(int[] nums, int target) {

if (nums == null || nums.length == 0) {

return -1;

}

int left = 0;

int right = nums.length - 1;

while (left <= right) {

int mid = left + (right - left) / 2;

if (nums[mid] == target) {

return mid;

}

// Check if the left half is sorted

if (nums[left] <= nums[mid]) {

// Check if the target is in the left half

if (nums[left] <= target && target < nums[mid]) {

right = mid - 1;

} else {

left = mid + 1;

}

} else { // The right half is sorted

// Check if the target is in the right half

if (nums[mid] < target && target <= nums[right]) {

left = mid + 1;

} else {

right = mid - 1;

}

}

}

return -1; // Target not found

}

public static void main(String[] args) {

int[] nums = {4, 5, 6, 7, 0, 1, 2};

int target = 0;

int result = search(nums, target);

if (result != -1) {

System.out.println("Target " + target + " found at index " + result);

} else {

System.out.println("Target " + target + " not found in the array");

}

}

}

Write a program to find the shortest path in a weighted graph using Dijkstra's algorithm in Java.

import java.util.\*;

public class DijkstraAlgorithm {

static class Edge {

int destination;

int weight;

public Edge(int destination, int weight) {

this.destination = destination;

this.weight = weight;

}

}

static class Vertex implements Comparable<Vertex> {

int index;

int distance;

public Vertex(int index, int distance) {

this.index = index;

this.distance = distance;

}

@Override

public int compareTo(Vertex other) {

return Integer.compare(this.distance, other.distance);

}

}

public static int[] dijkstra(List<List<Edge>> graph, int start) {

int vertices = graph.size();

int[] distance = new int[vertices];

Arrays.fill(distance, Integer.MAX\_VALUE);

PriorityQueue<Vertex> minHeap = new PriorityQueue<>();

minHeap.add(new Vertex(start, 0));

distance[start] = 0;

while (!minHeap.isEmpty()) {

Vertex current = minHeap.poll();

for (Edge neighbor : graph.get(current.index)) {

int newDist = current.distance + neighbor.weight;

if (newDist < distance[neighbor.destination]) {

distance[neighbor.destination] = newDist;

minHeap.add(new Vertex(neighbor.destination, newDist));

}

}

}

return distance;

}

**3 Selenium Framework Questions:**

Create a Page Object Model for the login page of a website. Use Page Factory to initialize the elements. Write a TestNG test to verify successful login with valid credentials.

Create a Page Object Model for the shopping cart page of a website. Use Page Factory to initialize the elements. Write a TestNG test to add a product to the cart, navigate to the cart page, and verify that the product is added to the cart.

Create a Page Object Model for the search page of a website. Use Page Factory to initialize the elements. Write a TestNG test to search for a product, verify that the search results are displayed, and click on a specific product to navigate to its details page.

------------------------------------------------------------------------------------------------------------------------------------------

**Level2**

What is the difference between an abstract class and an interface in Java?

In summary, use an abstract class when you want to provide a common base class with some shared functionality, and use an interface when you want to define a contract for multiple classes to implement.

What is the purpose of the static keyword in Java?

the **static** keyword is associated with the class and not with instances of the class. It is commonly used for utility methods, constants, and other situations where the behavior or data is shared across all instances of a class.

What is the purpose of the final keyword in Java?

Using the **final** keyword can contribute to code safety, readability, and maintainability by enforcing constraints on variables, methods, and classes. It helps in creating robust and predictable code, especially in the context of constants, immutability, and preventing unintended overrides.

What is the difference between a checked and an unchecked exception in Java?

In summary, the key distinction is that checked exceptions must be either caught or declared in the method signature, whereas unchecked exceptions (runtime exceptions) do not have this requirement. Checked exceptions are typically used for recoverable conditions, while unchecked exceptions indicate errors in the program logic.

What is the difference between a stack and a queue data structure in Java?

In Java, a stack and a queue are both data structures that store and manage collections of elements. However, they differ in their principles of operation, order of access, and the way elements are added and removed.

What is the purpose of the synchronized keyword in Java?

In Java, the **synchronized** keyword is used to control access to shared resources or critical sections of code by allowing only one thread at a time to execute a block of code that has been marked as synchronized. This ensures that multiple threads do not interfere with each other when accessing or modifying shared data, which helps prevent race conditions and maintain data consistency.

What is the purpose of the volatile keyword in Java?

In Java, the **volatile** keyword is used to declare a variable as volatile. The main purpose of the **volatile** keyword is to indicate that a variable's value may be changed by multiple threads simultaneously. This informs the compiler and the runtime environment that the variable should not be cached, and its value should always be read from and written to the main memory, rather than relying on thread-specific caches.

What is the purpose of the transient keyword in Java?

In Java, the **transient** keyword is used as a modifier for instance variables. Its primary purpose is to indicate to the Java Virtual Machine (JVM) that a particular variable should not be serialized when the object is transformed into a stream of bytes. Serialization is the process of converting an object into a format that can be easily stored or transmitted and reconstructed later.

What is the difference between the equals() and hashCode() methods in Java?

n Java, the **equals()** and **hashCode()** methods are related to object comparison and hash code generation, respectively. These methods are crucial when working with collections, such as **HashMap** and **HashSet**, as they define how objects are compared and stored.

What is the purpose of the finalize() method in Java?

The **finalize()** method in Java is a method provided by the **Object** class. It is called by the garbage collector before an object is reclaimed by the memory management system. The purpose of the **finalize()** method is to give an object an opportunity to perform cleanup or resource release operations before it is garbage collected.

What is the purpose of the this keyword in Java?

In Java, the **this** keyword is a reference variable that is used to refer to the current object. It is primarily used in the context of instance methods and constructors to differentiate between instance variables and parameters when they have the same name.

What is the difference between a superclass and a subclass in Java?

In summary, a superclass is the class from which another class inherits, providing a common set of attributes and behaviors. A subclass is the class that inherits from a superclass, extending or specializing its attributes and behaviors while potentially introducing new ones. Inheritance promotes code reuse and abstraction in object-oriented programming.

What is the purpose of the package keyword in Java?

the **package** keyword is used to define a package, which is a way of organizing and encapsulating related classes and interfaces. A package is a namespace that helps prevent naming conflicts and provides a logical structure for organizing Java code.

What is the difference between an instance variable and a class variable in Java?

In Java, both instance variables and class variables are types of fields or member variables within a class. However, they serve different purposes and have distinct characteristics.

What is the purpose of the instanceof operator in Java?

the **instanceof** operator is used to test whether an object is an instance of a particular class or interface. It is a binary operator that returns a boolean value indicating whether the object on the left-hand side is an instance of the type specified on the right-hand side.

What is the purpose of the super keyword in Java?

The **super** keyword is primarily used in the context of inheritance to differentiate between members of the subclass and members of the superclass.

What is the purpose of the abstract keyword in Java?

In Java, the **abstract** keyword is used to declare abstract classes and abstract methods. An abstract class is a class that cannot be instantiated on its own and may contain abstract methods, which are methods that are declared but have no implementation in the abstract class.

What is the purpose of the interface keyword in Java?

In Java, the **interface** keyword is used to declare interfaces. An interface in Java is a collection of abstract methods (methods without an implementation) and constant fields. It defines a contract for classes that implement it, specifying a set of methods that must be implemented by any class that wants to conform to the interface.

What is the difference between a private and a protected access modifier in Java?

In summary, the **interface** keyword in Java is used to define interfaces, which are collections of abstract methods and constant fields. Interfaces provide a way to achieve multiple inheritance, define a common set of methods, promote code reusability, and create a contract for classes that implement them.

What is the purpose of the throws keyword in Java?

In Java, the throws keyword is used in the method declaration to indicate that a particular method might throw certain exceptions during its execution. This keyword is part of the method signature and is followed by a list of exception types that the method can potentially throw.

**10 Java Programming Questions:**

Write a Java program to find the factorial of a number using recursion.

public class FactorialCalculator {

public static void main(String[] args) {

// Test the factorial calculation

int number = 5;

long factorial = calculateFactorial(number);

System.out.println("Factorial of " + number + " is: " + factorial);

}

// Recursive method to calculate factorial

public static long calculateFactorial(int n) {

if (n == 0 || n == 1) {

return 1; // Base case: factorial of 0 and 1 is 1

} else {

return n \* calculateFactorial(n - 1); // Recursive case

}

}

}

Write a Java program to check if a given number is a palindrome or not.

import java.util.Scanner;

public class PalindromeChecker {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Input: Get the number from the user

System.out.print("Enter a number: ");

int number = scanner.nextInt();

// Check if the number is a palindrome

if (isPalindrome(number)) {

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

} else {

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

}

scanner.close();

}

// Function to check if a number is a palindrome

public static boolean isPalindrome(int num) {

int originalNumber = num;

int reversedNumber = 0;

// Reverse the number

while (num > 0) {

int digit = num % 10;

reversedNumber = reversedNumber \* 10 + digit;

num /= 10;

}

// Check if the reversed number is equal to the original number

return originalNumber == reversedNumber;

}

}

Write a Java program to check if a given string is a palindrome or not.

import java.util.Scanner;

public class StringPalindromeChecker {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Input: Get the string from the user

System.out.print("Enter a string: ");

String inputString = scanner.nextLine();

// Check if the string is a palindrome

if (isPalindrome(inputString)) {

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

} else {

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

}

scanner.close();

}

// Function to check if a string is a palindrome

public static boolean isPalindrome(String str) {

str = str.toLowerCase().replaceAll("[^a-z0-9]", "");

int left = 0;

int right = str.length() - 1;

while (left < right) {

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

return false;

}

left++;

right--;

}

return true;

}

}

Write a Java program to reverse a given string without using any built-in functions.

import java.util.Scanner;

public class StringReversal {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Input: Get the string from the user

System.out.print("Enter a string: ");

String inputString = scanner.nextLine();

// Reverse the string

String reversedString = reverseString(inputString);

// Output: Display the reversed string

System.out.println("Reversed String: " + reversedString);

scanner.close();

}

// Function to reverse a string without using built-in functions

public static String reverseString(String str) {

char[] charArray = str.toCharArray();

int left = 0;

int right = charArray.length - 1;

// Swap characters from both ends

while (left < right) {

char temp = charArray[left];

charArray[left] = charArray[right];

charArray[right] = temp;

left++;

right--;

}

// Convert the character array back to a string

return new String(charArray);

}

}

Write a Java program to find the second highest number in an array.

import java.util.Scanner;

public class SecondHighestNumber {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Input: Get the size of the array from the user

System.out.print("Enter the size of the array: ");

int size = scanner.nextInt();

// Input: Get the elements of the array from the user

int[] array = new int[size];

System.out.println("Enter the elements of the array:");

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

array[i] = scanner.nextInt();

}

// Find the second highest number

int secondHighest = findSecondHighest(array);

// Output: Display the result

System.out.println("The second highest number is: " + secondHighest);

scanner.close();

}

// Function to find the second highest number in an array

public static int findSecondHighest(int[] array) {

if (array.length < 2) {

System.out.println("Array size should be at least 2.");

return -1; // Return a sentinel value to indicate an error

}

int firstHighest = Integer.MIN\_VALUE;

int secondHighest = Integer.MIN\_VALUE;

for (int num : array) {

if (num > firstHighest) {

secondHighest = firstHighest;

firstHighest = num;

} else if (num > secondHighest && num < firstHighest) {

secondHighest = num;

}

}

return secondHighest;

}

}

Write a Java program to implement bubble sort algorithm.

import java.util.Scanner;

public class BubbleSort {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Input: Get the size of the array from the user

System.out.print("Enter the size of the array: ");

int size = scanner.nextInt();

// Input: Get the elements of the array from the user

int[] array = new int[size];

System.out.println("Enter the elements of the array:");

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

array[i] = scanner.nextInt();

}

// Sorting the array using Bubble Sort

bubbleSort(array);

// Output: Display the sorted array

System.out.println("Sorted Array:");

printArray(array);

scanner.close();

}

// Function to perform Bubble Sort on an array

public static void bubbleSort(int[] array) {

int n = array.length;

for (int i = 0; i < n - 1; i++) {

for (int j = 0; j < n - 1 - i; j++) {

// Swap if the element found is greater than the next element

if (array[j] > array[j + 1]) {

int temp = array[j];

array[j] = array[j + 1];

array[j + 1] = temp;

}

}

}

}

// Function to print an array

public static void printArray(int[] array) {

for (int num : array) {

System.out.print(num + " ");

}

System.out.println();

}

}

Write a Java program to implement selection sort algorithm.

import java.util.Scanner;

public class SelectionSort {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Input: Get the size of the array from the user

System.out.print("Enter the size of the array: ");

int size = scanner.nextInt();

// Input: Get the elements of the array from the user

int[] array = new int[size];

System.out.println("Enter the elements of the array:");

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

array[i] = scanner.nextInt();

}

// Sorting the array using Selection Sort

selectionSort(array);

// Output: Display the sorted array

System.out.println("Sorted Array:");

printArray(array);

scanner.close();

}

// Function to perform Selection Sort on an array

public static void selectionSort(int[] array) {

int n = array.length;

for (int i = 0; i < n - 1; i++) {

int minIndex = i;

// Find the index of the minimum element in the unsorted part of the array

for (int j = i + 1; j < n; j++) {

if (array[j] < array[minIndex]) {

minIndex = j;

}

}

// Swap the found minimum element with the first element in the unsorted part

int temp = array[minIndex];

array[minIndex] = array[i];

array[i] = temp;

}

}

// Function to print an array

public static void printArray(int[] array) {

for (int num : array) {

System.out.print(num + " ");

}

System.out.println();

}

}

Write a Java program to implement insertion sort algorithm.

import java.util.Scanner;

public class InsertionSort {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Input: Get the size of the array from the user

System.out.print("Enter the size of the array: ");

int size = scanner.nextInt();

// Input: Get the elements of the array from the user

int[] array = new int[size];

System.out.println("Enter the elements of the array:");

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

array[i] = scanner.nextInt();

}

// Sorting the array using Insertion Sort

insertionSort(array);

// Output: Display the sorted array

System.out.println("Sorted Array:");

printArray(array);

scanner.close();

}

// Function to perform Insertion Sort on an array

public static void insertionSort(int[] array) {

int n = array.length;

for (int i = 1; i < n; i++) {

int key = array[i];

int j = i - 1;

// Move elements greater than key to one position ahead of their current position

while (j >= 0 && array[j] > key) {

array[j + 1] = array[j];

j--;

}

// Insert the key into its correct position

array[j + 1] = key;

}

}

// Function to print an array

public static void printArray(int[] array) {

for (int num : array) {

System.out.print(num + " ");

}

System.out.println();

}

}

Write a Java program to implement binary search algorithm.

import java.util.Scanner;

public class BinarySearch {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Input: Get the size of the array from the user

System.out.print("Enter the size of the sorted array: ");

int size = scanner.nextInt();

// Input: Get the elements of the sorted array from the user

int[] sortedArray = new int[size];

System.out.println("Enter the sorted elements of the array:");

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

sortedArray[i] = scanner.nextInt();

}

// Input: Get the element to search for

System.out.print("Enter the element to search for: ");

int target = scanner.nextInt();

// Perform Binary Search

int resultIndex = binarySearch(sortedArray, target);

// Output: Display the result

if (resultIndex != -1) {

System.out.println("Element " + target + " found at index " + resultIndex);

} else {

System.out.println("Element " + target + " not found in the array.");

}

scanner.close();

}

// Function to perform Binary Search on a sorted array

public static int binarySearch(int[] array, int target) {

int left = 0;

int right = array.length - 1;

while (left <= right) {

int mid = left + (right - left) / 2;

// Check if the target is present at the middle

if (array[mid] == target) {

return mid;

}

// If the target is greater, ignore the left half

if (array[mid] < target) {

left = mid + 1;

}

// If the target is smaller, ignore the right half

else {

right = mid - 1;

}

}

// If the target is not present in the array

return -1;

}

}

Write a Java program to count the number of words in a given string.

import java.util.Scanner;

public class WordCount {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Input: Get the string from the user

System.out.print("Enter a string: ");

String inputString = scanner.nextLine();

// Count the number of words in the string

int wordCount = countWords(inputString);

// Output: Display the result

System.out.println("Number of words in the string: " + wordCount);

scanner.close();

}

// Function to count the number of words in a string

public static int countWords(String str) {

// Use regex to split the string into words

String[] words = str.split("\\s+");

// Return the number of words

return words.length;

}

}

**3 Selenium Framework Questions:**

Create a Page Object Model for the registration page of a website. Use Page Factory to initialize the elements. Write a TestNG test to verify successful registration with valid credentials.

Create a Page Object Model for the checkout page of a website. Use Page Factory to initialize the elements. Write a TestNG test to add a product to the cart, navigate to the checkout page, and verify that the correct product is being purchased.

Create a Page Object Model for the home page of a website. Use Page Factory to initialize the elements. Write a TestNG test to navigate to the home page, verify the presence of certain elements, and click on a link to navigate to a different page.

------------------------------------------------------------------------------------------------------------------------------------------

**Level3**

What is the purpose of the public keyword in Java?

In Java, the **public** keyword is an access modifier used to declare the visibility of classes, methods, and fields. It indicates that the declared entity (class, method, or field) is accessible from any other class or package.

What is the purpose of the private keyword in Java?

In Java, the **private** keyword is an access modifier that restricts the visibility of classes, methods, and fields to only the same class in which they are declared. It provides the most restrictive access level and is used for encapsulation, helping to hide the internal implementation details of a class from external classes or code.

What is the purpose of the protected keyword in Java?

the **protected** keyword is an access modifier used to restrict the visibility of classes, methods, and fields. The **protected** modifier provides a level of access that is more permissive than **private** but more restrictive than **public**

What is the purpose of the static keyword in Java?

In Java, the **static** keyword is used to define a member (variable or method) that belongs to the class rather than to instances of the class. Here are the main purposes and use cases of the **static**

What is the purpose of the final keyword in Java?

n Java, the **final** keyword is a modifier that can be applied to classes, methods, variables, and fields. It serves various purposes depending on where it is used.

What is the purpose of the abstract keyword in Java?

the **abstract** keyword is to allow the definition of common behavior in an abstract class while deferring the implementation of certain details to its concrete subclasses.

What is the purpose of the interface keyword in Java?

s used to declare an interface. An interface in Java is a collection of abstract methods (methods without a body) and constants (final variables).

What is the purpose of the new keyword in Java?

keyword is primarily used for creating instances of classes and arrays. Here are the main purposes of the **new**

What is the difference between a class and an object in Java?

a class is a blueprint or template used to define the structure and behavior of objects, while an object is an instance of a class, representing a specific entity with its own state and behavior at runtime.

What is the purpose of the main() method in Java?

the **main()** method is a crucial component of every Java program, providing a standardized entry point for the JVM to start execution. It is where the program's logic begins, and it allows for the handling of command-line arguments and exception management.

What is the difference between a float and a double in Java?

In general, **double** is the default choice for representing floating-point numbers in Java due to its higher precision, and it is commonly used for most applications where precision is important. **float** is used in scenarios where memory constraints are critical, or in specialized situations where lower precision is acceptable.

What is the difference between an int and a long in Java?

**int** is a 32-bit signed integer type, suitable for most integer values within its range, while **long** is a 64-bit signed integer type, used when a larger range is needed or when dealing with large numbers.

What is the purpose of the System.out.println() method in Java?

is a versatile method in Java for printing text to the console. It is widely used for debugging, displaying program output, and providing information to users during program execution.

What is the purpose of the Scanner class in Java?

he **Scanner** class in Java is versatile and widely used for reading input from various sources. It simplifies the process of obtaining user input, parsing data, and reading from files, making it a valuable tool for interactive console applications and data processing tasks.

What is the purpose of the Math class in Java?

The **Math** class provides a standardized set of mathematical functions that are often used in scientific and engineering applications. It offers a convenient way to perform common mathematical operations without having to implement them from scratch in Java programs. Note that the methods in the **Math** class are static, meaning they are invoked on the class itself rather than on an instance of the class.

What is the purpose of the String class in Java?

the **String** class in Java is essential for working with textual data. Its immutability, extensive set of methods, and integration into the language make it a versatile and widely used class.

What is the purpose of the StringBuilder class in Java?

he **StringBuilder** class in Java is designed for efficient manipulation of mutable character sequences. It is particularly useful for scenarios where you need to concatenate, insert, or modify strings dynamically. Using **StringBuilder** can lead to better performance compared to repeated string concatenation using the **+** operator.

What is the purpose of the equals() method in Java?

the **equals()** method in Java is used for comparing the equality of objects. It is often overridden in classes to provide a content-based comparison, and it plays a crucial role in maintaining the integrity of various Java collections that rely on equality.

What is the purpose of the compareTo() method in Java?

the **compareTo()** method is part of the **Comparable** interface, and it is used to compare the current object with another object. This method is typically implemented by classes whose instances can be ordered or sorted.

What is the purpose of the toString() method in Java?

the **toString()** method in Java is used to provide a string representation of an object. It is often overridden in user-defined classes to offer a more meaningful representation for debugging and logging purposes.

**10 Java Programming Questions:**

Write a Java program to print "Hello, World!" to the console.

public class HelloWorld {

public static void main(String[] args) {

System.out.println("Hello, World!");

}

}

javac HelloWorld.java

java HelloWorld

Hello, World!

Write a Java program to find the sum of two numbers.

import java.util.Scanner;

public class SumCalculator {

public static void main(String[] args) {

// Create a Scanner object to read input from the user

Scanner scanner = new Scanner(System.in);

// Prompt the user to enter the first number

System.out.print("Enter the first number: ");

double num1 = scanner.nextDouble();

// Prompt the user to enter the second number

System.out.print("Enter the second number: ");

double num2 = scanner.nextDouble();

// Calculate the sum

double sum = num1 + num2;

// Display the result

System.out.println("The sum of " + num1 + " and " + num2 + " is: " + sum);

// Close the scanner to avoid resource leakage

scanner.close();

}

}

Write a Java program to find the average of three numbers.

import java.util.Scanner;

public class AverageCalculator {

public static void main(String[] args) {

// Create a Scanner object to read input from the user

Scanner scanner = new Scanner(System.in);

// Prompt the user to enter the first number

System.out.print("Enter the first number: ");

double num1 = scanner.nextDouble();

// Prompt the user to enter the second number

System.out.print("Enter the second number: ");

double num2 = scanner.nextDouble();

// Prompt the user to enter the third number

System.out.print("Enter the third number: ");

double num3 = scanner.nextDouble();

// Calculate the average

double average = (num1 + num2 + num3) / 3;

// Display the result

System.out.println("The average of " + num1 + ", " + num2 + ", and " + num3 + " is: " + average);

// Close the scanner to avoid resource leakage

scanner.close();

}

}

Write a Java program to check if a given number is even or odd.

import java.util.Scanner;

public class EvenOddChecker {

public static void main(String[] args) {

// Create a Scanner object to read input from the user

Scanner scanner = new Scanner(System.in);

// Prompt the user to enter a number

System.out.print("Enter a number: ");

int number = scanner.nextInt();

// Check if the number is even or odd

if (number % 2 == 0) {

System.out.println(number + " is an even number.");

} else {

System.out.println(number + " is an odd number.");

}

// Close the scanner to avoid resource leakage

scanner.close();

}

}

Write a Java program to check if a given number is prime or not.

import java.util.Scanner;

public class PrimeNumberChecker {

public static void main(String[] args) {

// Create a Scanner object to read input from the user

Scanner scanner = new Scanner(System.in);

// Prompt the user to enter a number

System.out.print("Enter a number: ");

int number = scanner.nextInt();

// Check if the number is prime

if (isPrime(number)) {

System.out.println(number + " is a prime number.");

} else {

System.out.println(number + " is not a prime number.");

}

// Close the scanner to avoid resource leakage

scanner.close();

}

// Function to check if a number is prime

private static boolean isPrime(int num) {

if (num <= 1) {

return false;

}

for (int i = 2; i <= Math.sqrt(num); i++) {

if (num % i == 0) {

return false;

}

}

return true;

}

}

Write a Java program to check if a given string is a palindrome or not.

import java.util.Scanner;

public class PalindromeChecker {

public static void main(String[] args) {

// Create a Scanner object to read input from the user

Scanner scanner = new Scanner(System.in);

// Prompt the user to enter a string

System.out.print("Enter a string: ");

String input = scanner.nextLine();

// Check if the string is a palindrome

if (isPalindrome(input)) {

System.out.println("The string \"" + input + "\" is a palindrome.");

} else {

System.out.println("The string \"" + input + "\" is not a palindrome.");

}

// Close the scanner to avoid resource leakage

scanner.close();

}

// Function to check if a string is a palindrome

private static boolean isPalindrome(String str) {

// Remove spaces and convert to lowercase for a case-insensitive check

String cleanStr = str.replaceAll("\\s", "").toLowerCase();

int length = cleanStr.length();

for (int i = 0; i < length / 2; i++) {

if (cleanStr.charAt(i) != cleanStr.charAt(length - 1 - i)) {

return false;

}

}

return true;

}

}

Write a Java program to reverse a given string.

import java.util.Scanner;

public class StringReverser {

public static void main(String[] args) {

// Create a Scanner object to read input from the user

Scanner scanner = new Scanner(System.in);

// Prompt the user to enter a string

System.out.print("Enter a string: ");

String input = scanner.nextLine();

// Reverse the string

String reversedString = reverseString(input);

// Display the reversed string

System.out.println("Reversed string: " + reversedString);

// Close the scanner to avoid resource leakage

scanner.close();

}

// Function to reverse a string

private static String reverseString(String str) {

char[] charArray = str.toCharArray();

int start = 0;

int end = str.length() - 1;

while (start < end) {

// Swap characters at start and end positions

char temp = charArray[start];

charArray[start] = charArray[end];

charArray[end] = temp;

// Move towards the center

start++;

end--;

}

// Convert the character array back to a string

return new String(charArray);

}

}

Write a Java program to implement a simple calculator.

import java.util.Scanner;

public class SimpleCalculator {

public static void main(String[] args) {

// Create a Scanner object to read input from the user

Scanner scanner = new Scanner(System.in);

// Prompt the user to enter two numbers

System.out.print("Enter the first number: ");

double num1 = scanner.nextDouble();

System.out.print("Enter the second number: ");

double num2 = scanner.nextDouble();

// Prompt the user to choose an operation

System.out.println("Choose an operation:");

System.out.println("1. Addition");

System.out.println("2. Subtraction");

System.out.println("3. Multiplication");

System.out.println("4. Division");

// Read the user's choice

System.out.print("Enter the operation number: ");

int choice = scanner.nextInt();

// Perform the selected operation

double result = 0;

switch (choice) {

case 1:

result = num1 + num2;

System.out.println("Result: " + result);

break;

case 2:

result = num1 - num2;

System.out.println("Result: " + result);

break;

case 3:

result = num1 \* num2;

System.out.println("Result: " + result);

break;

case 4:

if (num2 != 0) {

result = num1 / num2;

System.out.println("Result: " + result);

} else {

System.out.println("Cannot divide by zero.");

}

break;

default:

System.out.println("Invalid operation choice");

}

// Close the scanner to avoid resource leakage

scanner.close();

}

}

Write a Java program to convert Fahrenheit to Celsius.

import java.util.Scanner;

public class FahrenheitToCelsiusConverter {

public static void main(String[] args) {

// Create a Scanner object to read input from the user

Scanner scanner = new Scanner(System.in);

// Prompt the user to enter the temperature in Fahrenheit

System.out.print("Enter the temperature in Fahrenheit: ");

double fahrenheit = scanner.nextDouble();

// Convert Fahrenheit to Celsius

double celsius = (fahrenheit - 32) \* 5/9;

// Display the result

System.out.println("Temperature in Celsius: " + celsius);

// Close the scanner to avoid resource leakage

scanner.close();

}

}

Write a Java program to generate a random number between 1 and 100.

import java.util.Random;

public class RandomNumberGenerator {

public static void main(String[] args) {

// Create a Random object

Random random = new Random();

// Generate a random number between 1 and 100 (inclusive)

int randomNumber = random.nextInt(100) + 1;

// Display the generated random number

System.out.println("Random Number: " + randomNumber);

}

}

**3 Selenium Framework Questions:**

Write a TestNG test to navigate to a website and verify the title of the page.

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.chrome.ChromeDriver;

import org.testng.Assert;

import org.testng.annotations.AfterClass;

import org.testng.annotations.BeforeClass;

import org.testng.annotations.Test;

public class WebsiteTitleTest {

// Set the path to your ChromeDriver executable

private static final String CHROMEDRIVER\_PATH = "path/to/chromedriver";

private WebDriver driver;

@BeforeClass

public void setUp() {

// Set the system property for ChromeDriver

System.setProperty("webdriver.chrome.driver", CHROMEDRIVER\_PATH);

// Initialize the WebDriver

driver = new ChromeDriver();

}

@Test

public void testWebsiteTitle() {

// Navigate to the website

String websiteURL = "https://www.example.com";

driver.get(websiteURL);

// Get the title of the page

String pageTitle = driver.getTitle();

// Verify the title of the page

Assert.assertEquals(pageTitle, "Expected Title", "Page title doesn't match expected title.");

}

@AfterClass

public void tearDown() {

// Close the WebDriver

if (driver != null) {

driver.quit();

}

}

}

Write a TestNG test to fill out a login form on a website and verify successful login with valid credentials.

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

import org.testng.Assert;

import org.testng.annotations.AfterClass;

import org.testng.annotations.BeforeClass;

import org.testng.annotations.Test;

public class LoginFormTest {

// Set the path to your ChromeDriver executable

private static final String CHROMEDRIVER\_PATH = "path/to/chromedriver";

private WebDriver driver;

@BeforeClass

public void setUp() {

// Set the system property for ChromeDriver

System.setProperty("webdriver.chrome.driver", CHROMEDRIVER\_PATH);

// Initialize the WebDriver

driver = new ChromeDriver();

}

@Test

public void testLoginWithValidCredentials() {

// Navigate to the login page

String loginPageURL = "https://www.example.com/login";

driver.get(loginPageURL);

// Find the username and password fields and the login button

WebElement usernameInput = driver.findElement(By.id("username"));

WebElement passwordInput = driver.findElement(By.id("password"));

WebElement loginButton = driver.findElement(By.id("loginButton"));

// Enter valid credentials

String validUsername = "your\_username";

String validPassword = "your\_password";

usernameInput.sendKeys(validUsername);

passwordInput.sendKeys(validPassword);

// Click the login button

loginButton.click();

// Verify successful login by checking if a welcome message is displayed

WebElement welcomeMessage = driver.findElement(By.id("welcomeMessage"));

Assert.assertTrue(welcomeMessage.isDisplayed(), "Login was not successful.");

}

@AfterClass

public void tearDown() {

// Close the WebDriver

if (driver != null) {

driver.quit();

}

}

}

Write a TestNG test to navigate to a website, click on a link to navigate to a different page, and verify the presence of certain elements on the new page.

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

import org.testng.Assert;

import org.testng.annotations.AfterClass;

import org.testng.annotations.BeforeClass;

import org.testng.annotations.Test;

public class NavigateAndVerifyTest {

// Set the path to your ChromeDriver executable

private static final String CHROMEDRIVER\_PATH = "path/to/chromedriver";

private WebDriver driver;

@BeforeClass

public void setUp() {

// Set the system property for ChromeDriver

System.setProperty("webdriver.chrome.driver", CHROMEDRIVER\_PATH);

// Initialize the WebDriver

driver = new ChromeDriver();

}

@Test

public void testNavigateAndVerifyElements() {

// Navigate to the main page

String mainPageURL = "https://www.example.com";

driver.get(mainPageURL);

// Find and click on the link to navigate to a different page

WebElement linkToNewPage = driver.findElement(By.linkText("Click Here"));

linkToNewPage.click();

// Verify the presence of certain elements on the new page

WebElement element1 = driver.findElement(By.id("element1"));

WebElement element2 = driver.findElement(By.cssSelector(".element2"));

Assert.assertTrue(element1.isDisplayed(), "Element 1 is not displayed on the new page.");

Assert.assertTrue(element2.isDisplayed(), "Element 2 is not displayed on the new page.");

}

@AfterClass

public void tearDown() {

// Close the WebDriver

if (driver != null) {

driver.quit();

}

}

}