**JAVA Questions**

**1. Reverse a string in Java**

public class StringPrograms {

public static void main(String[] args) {

String str = "123";

System.out.println(reverse(str));

}

public static String reverse(String in) {

if (in == null)

throw new IllegalArgumentException("Null is not valid input");

StringBuilder out = new StringBuilder();

char[] chars = in.toCharArray();

for (int i = chars.length - 1; i >= 0; i--)

out.append(chars[i]);

return out.toString();

}

}

## 2. Java program to check if a vowel is present in a string.

public class StringContainsVowels {

public static void main(String[] args) {

System.out.println(stringContainsVowels("Hello")); // true

System.out.println(stringContainsVowels("TV")); // false

}

public static boolean stringContainsVowels(String input) {

return input.toLowerCase().matches(".\*[aeiou].\*");

}

}

## 3. Check whether a string is a palindrome in Java?

boolean checkPalindromeString(String input) {

boolean result = true;

int length = input.length();

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

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

result = false;

break;

}

}

return result;

}

## 4.  Create a deadlock scenario programmatically in Java?

public class ThreadDeadlock {

public static void main(String[] args) throws InterruptedException {

Object obj1 = new Object();

Object obj2 = new Object();

Object obj3 = new Object();

Thread t1 = new Thread(new SyncThread(obj1, obj2), "t1");

Thread t2 = new Thread(new SyncThread(obj2, obj3), "t2");

Thread t3 = new Thread(new SyncThread(obj3, obj1), "t3");

t1.start();

Thread.sleep(5000);

t2.start();

Thread.sleep(5000);

t3.start();

}

}

class SyncThread implements Runnable {

private Object obj1;

private Object obj2;

public SyncThread(Object o1, Object o2) {

this.obj1 = o1;

this.obj2 = o2;

}

@Override

public void run() {

String name = Thread.currentThread().getName();

System.out.println(name + " acquiring lock on " + obj1);

synchronized (obj1) {

System.out.println(name + " acquired lock on " + obj1);

work();

System.out.println(name + " acquiring lock on " + obj2);

synchronized (obj2) {

System.out.println(name + " acquired lock on " + obj2);

work();

}

System.out.println(name + " released lock on " + obj2);

}

System.out.println(name + " released lock on " + obj1);

System.out.println(name + " finished execution.");

}

private void work() {

try {

Thread.sleep(30000);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

## 5. Implement a binary search in Java?

public static int binarySearch(int arr[], int low, int high, int key) {

int mid = (low + high) / 2;

while (low <= high) {

if (arr[mid] < key) {

low = mid + 1;

} else if (arr[mid] == key) {

return mid;

} else {

high = mid - 1;

}

mid = (low + high) / 2;

}

if (low > high) {

return -1;

}

return -1;

}

## 6.  Java program that checks if two arrays contain the same elements.

import java.util.Arrays;

import java.util.HashSet;

import java.util.Set;

public class ArraySameElements {

public static void main(String[] args) {

Integer[] a1 = {1,2,3,2,1};

Integer[] a2 = {1,2,3};

Integer[] a3 = {1,2,3,4};

System.out.println(sameElements(a1, a2));

System.out.println(sameElements(a1, a3));

}

static boolean sameElements(Object[] array1, Object[] array2) {

Set<Object> uniqueElements1 = new HashSet<>(Arrays.asList(array1));

Set<Object> uniqueElements2 = new HashSet<>(Arrays.asList(array2));

// if size is different, means there will be a mismatch

if (uniqueElements1.size() != uniqueElements2.size()) return false;

for (Object obj : uniqueElements1) {

// element not present in both?

if (!uniqueElements2.contains(obj)) return false;

}

return true;

}

}

## 7. Find the second largest number in an array in Java?

private static int findSecondHighest(int[] array) {

int highest = Integer.MIN\_VALUE;

int secondHighest = Integer.MIN\_VALUE;

for (int i : array) {

if (i > highest) {

secondHighest = highest;

highest = i;

} else if (i > secondHighest) {

secondHighest = i;

}

}

return secondHighest;

}

## 8. Find a string in a text file in Java?

boolean findStringInFile(String filePath, String str) throws FileNotFoundException {

File file = new File(filePath);

Scanner scanner = new Scanner(file);

// read the file line by line

while (scanner.hasNextLine()) {

String line = scanner.nextLine();

if (line.contains(str)) {

scanner.close();

return true;

}

}

scanner.close();

return false;

}

## 9. Java program that sorts HashMap by value.

import java.util.ArrayList;

import java.util.HashMap;

import java.util.LinkedHashMap;

import java.util.List;

import java.util.Map;

import java.util.Map.Entry;

import java.util.Set;

public class SortHashMapByValue {

public static void main(String[] args) {

Map<String, Integer> scores = new HashMap<>();

scores.put("David", 95);

scores.put("Jane", 80);

scores.put("Mary", 97);

scores.put("Lisa", 78);

scores.put("Dino", 65);

System.out.println(scores);

scores = sortByValue(scores);

System.out.println(scores);

}

private static Map<String, Integer> sortByValue(Map<String, Integer> scores) {

Map<String, Integer> sortedByValue = new LinkedHashMap<>();

// get the entry set

Set<Entry<String, Integer>> entrySet = scores.entrySet();

System.out.println(entrySet);

// create a list since the set is unordered

List<Entry<String, Integer>> entryList = new ArrayList<>(entrySet);

System.out.println(entryList);

// sort the list by value

entryList.sort((x, y) -> x.getValue().compareTo(y.getValue()));

System.out.println(entryList);

// populate the new hash map

for (Entry<String, Integer> e : entryList)

sortedByValue.put(e.getKey(), e.getValue());

return sortedByValue;

}

}

## 10. Get distinct characters and their count in a string in Java?

String str1 = "abcdABCDabcd";

char[] chars = str1.toCharArray();

Map<Character, Integer> charsCount = new HashMap<>();

for (char c : chars) {

if (charsCount.containsKey(c)) {

charsCount.put(c, charsCount.get(c) + 1);

} else

charsCount.put(c, 1);

}

System.out.println(charsCount); // {a=2, A=1, b=2, B=1, c=2, C=1, d=2, D=1}

## 11. Implement Queue using Stack

|  |
| --- |
| // Java program to implement Queue using  // two stacks with costly enQueue()  **import** java.util.\*;    **class** GFG  {  **static** **class** Queue  {  **static** Stack<Integer> s1 = **new** Stack<Integer>();  **static** Stack<Integer> s2 = **new** Stack<Integer>();    **static** **void** enQueue(**int** x)      {          // Move all elements from s1 to s2  **while** (!s1.isEmpty())          {              s2.push(s1.pop());              //s1.pop();          }            // Push item into s1          s1.push(x);            // Push everything back to s1  **while** (!s2.isEmpty())          {              s1.push(s2.pop());              //s2.pop();          }      }        // Dequeue an item from the queue  **static** **int** deQueue()      {          // if first stack is empty  **if** (s1.isEmpty())          {              System.out.println("Q is Empty");              System.exit(0);          }            // Return top of s1  **int** x = s1.peek();          s1.pop();  **return** x;      }  };    // Driver code  **public** **static** **void** main(String[] args)  {      Queue q = **new** Queue();      q.enQueue(1);      q.enQueue(2);      q.enQueue(3);        System.out.println(q.deQueue());      System.out.println(q.deQueue());      System.out.println(q.deQueue());  }  } |

## 12. Implement stack using linked list

// Java program to Implement a stack

// using singly linked list

// import package

**import** **static** java.lang.System.exit;

// Driver code

**class** GFG {

**public** **static** **void** main(String[] args)

    {

        // create Object of Implementing class

        StackUsingLinkedlist obj

            = **new** StackUsingLinkedlist();

        // insert Stack value

        obj.push(11);

        obj.push(22);

        obj.push(33);

        obj.push(44);

        // print Stack elements

        obj.display();

        // print Top element of Stack

        System.out.printf("\nTop element is %d\n",

                          obj.peek());

        // Delete top element of Stack

        obj.pop();

        obj.pop();

        // print Stack elements

        obj.display();

        // print Top element of Stack

        System.out.printf("\nTop element is %d\n",

                          obj.peek());

    }

}

// Create Stack Using Linked list

**class** StackUsingLinkedlist {

    // A linked list node

**private** **class** Node {

**int** data; // integer data

        Node link; // reference variable Node type

    }

    // create global top reference variable global

    Node top;

    // Constructor

    StackUsingLinkedlist() { **this**.top = **null**; }

    // Utility function to add an element x in the stack

**public** **void** push(**int** x) // insert at the beginning

    {

        // create new node temp and allocate memory

        Node temp = **new** Node();

        // check if stack (heap) is full. Then inserting an

        //  element would lead to stack overflow

**if** (temp == **null**) {

            System.out.print("\nHeap Overflow");

**return**;

        }

        // initialize data into temp data field

        temp.data = x;

        // put top reference into temp link

        temp.link = top;

        // update top reference

        top = temp;

    }

    // Utility function to check if the stack is empty or

    // not

**public** **boolean** isEmpty() { **return** top == **null**; }

    // Utility function to return top element in a stack

**public** **int** peek()

    {

        // check for empty stack

**if** (!isEmpty()) {

**return** top.data;

        }

**else** {

            System.out.println("Stack is empty");

**return** -1;

        }

    }

    // Utility function to pop top element from the stack

**public** **void** pop() // remove at the beginning

    {

        // check for stack underflow

**if** (top == **null**) {

            System.out.print("\nStack Underflow");

**return**;

        }

        // update the top pointer to point to the next node

        top = (top).link;

    }

**public** **void** display()

    {

        // check for stack underflow

**if** (top == **null**) {

            System.out.printf("\nStack Underflow");

            exit(1);

        }

**else** {

            Node temp = top;

**while** (temp != **null**) {

                // print node data

                System.out.print(temp.data);

                // assign temp link to temp

                temp = temp.link;

**if**(temp != **null**)

                    System.out.print(" -> ");

            }

        }

    }

}