Java Coding Interview Questions and Answers for 2-5 Years Experience

Author: Ramesh Fadatare

CODING INTERVIEW JAVA

In this blog post, we will discuss frequently asked Java _coding interview questions along with complete programs and their outputs.

1. What happens if you put the return statement in the 'try' or 'catch' block? Will the 'finally' block execute?

If a *return* statement is encountered inside the try or catch block, the control immediately exits the try or catch block, and the *finally* block will execute before the method returns the value.

The *finally* block ensures that any cleanup or resource-releasing code is executed regardless of whether an exception occurred or not.

The below program and its output demonstrate the same:

```
public class FinallyBlockExample {
    public static int divide(int a, int b) {
        try {
            return a / b;
        } catch (ArithmeticException e) {
            System.out.println("Exception caught: " + e.getMessage());
            return -1; // This will be executed before finally block
        } finally {
            System.out.println("Finally block is executed.");
        }
    }

    public static void main(String[] args) {
        int result = divide(10, 0);
        System.out.println("Result: " + result);
    }
```

```
}
```

Output:

```
Exception caught: / by zero
Finally block is executed.
Result: -1
```

To summarize, if a *return* statement is used inside the try or catch block, the *finally* block will execute.

2. What happens if you put the System.exit() inside the try or catch block? Will the 'finally' block execute?

If *System.exit()* is called inside the try or catch block, the Java Virtual Machine (JVM) terminates the program immediately, and the *finally* block will not execute.

The *finally* block is skipped because *System.exit()* causes the program to terminate abruptly.

The below program and its output demonstrate the same:

```
public class FinallyBlockExample {
    public static void main(String[] args) {
        try {
            System.out.println("Inside try block");
            System.exit(0); // JVM will terminate here, finally block won't execute
        } catch (Exception e) {
            System.out.println("Exception caught: " + e.getMessage());
        } finally {
            System.out.println("Finally block is executed."); // This won't be
executed
        }
    }
}
```

Output:

```
Inside try block
```

To summarize, if *System.exit()* is used inside the try or catch block, the *finally* block will be skipped, and the program will terminate immediately.

3. What will happen if you try to store a key that is already present in HashMap?

If you try to store a key that is already present in a **HashMap**, the existing value associated with that key will be replaced by the new value that you are trying to put. The **HashMap** in Java does not allow duplicate keys; each key in the map must be unique.

Here's how it works:

```
import java.util.HashMap;

public class HashMapExample {
    public static void main(String[] args) {
        // Create a HashMap
        HashMap<String, Integer> hashMap = new HashMap<>();

        // Put key-value pairs in the HashMap
        hashMap.put("one", 1);
        hashMap.put("two", 2);
        hashMap.put("three", 3);

        // Print the original HashMap
        System.out.println("Original HashMap: " + hashMap);

        // Put a key that is already present
        hashMap.put("two", 22);

        // Print the updated HashMap
        System.out.println("Updated HashMap: " + hashMap);
    }
}
```

Output:

```
Original HashMap: {one=1, two=2, three=3}
Updated HashMap: {one=1, two=22, three=3}
```

As you can see, when we tried to put the key "two" with the value 22, which is already present in the **HashMap**, it updated the value associated with the key "two" to 22. The previous value 2 was replaced by the new value.

So, in summary, if you attempt to store a key that is already present in a **HashMap**, the existing key-value pair will be updated with the new key-value pair, effectively replacing the previous value with the new value.

4. If a method throws NullPointerException in the superclass, can we override it with a method that throws RuntimeException?

Yes, you can override a method in a subclass with a method that throws a different type of exception, even if the superclass method throws a more specific exception.

In Java, when you override a method, you are allowed to throw a subclass of the exception thrown by the superclass method, or you can choose not to throw any checked exception at all. However, you are not allowed to throw a checked exception that is higher up in the exception hierarchy than the one thrown by the superclass method. In other words, you can only make the exception more specific or leave it unchecked.

Here's an example to illustrate this:

```
class Superclass {
    void foo() throws NullPointerException {
        throw new NullPointerException("NullPointerException in Superclass");
    }
}

class Subclass extends Superclass {
    @Override
    void foo() throws RuntimeException {
        throw new RuntimeException("RuntimeException in Subclass");
    }
}

public class Main {
    public static void main(String[] args) {
        Superclass obj = new Subclass();
        obj.foo();
    }
}
```

Output:

```
Exception in thread "main" java.lang.RuntimeException: RuntimeException in Subclass
        at Subclass.foo(Main.java:10)
        at Main.main(Main.java:17)
```

In this example, the *`Superclass`* has a method *`foo()`* that throws a *`NullPointerException`*. The Subclass overrides the method *`foo()`* and throws a *`RuntimeException`*. This is valid because *`RuntimeException`* is an unchecked exception and does not need to be declared in the method signature.

However, keep in mind that throwing a broader exception in the subclass can lead to a loss of information about the specific exception that occurred in the superclass. So, it's generally considered good practice to maintain the same or more specific exception in the overriding method, and only use unchecked exceptions when necessary.

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

```
public class ReverseString {
    public static String reverseString(String str) {
        StringBuilder sb = new StringBuilder();
        for (int i = str.length() - 1; i >= 0; i--) {
            sb.append(str.charAt(i));
        }
        return sb.toString();
    }

    public static void main(String[] args) {
        String input = "Java Guides";
        String reversed = reverseString(input);
        System.out.println("Reversed String: " + reversed);
    }
}
```

Output:

```
Reversed String: sediuG avaJ
```

6. Write a Java program to find the missing number in an array of integers from 1 to n

```
public class FindMissingNumber {
    public static int findMissingNumber(int[] nums, int n) {
        int totalSum = n * (n + 1) / 2;
        int actualSum = 0;
        for (int num : nums) {
            actualSum += num;
        }
        return totalSum - actualSum;
    }
```

```
    public static void main(String[] args) {
        int[] nums = {1, 2, 4, 6, 3, 7, 8};
        int n = 8;
        int missingNumber = findMissingNumber(nums, n);
        System.out.println("Missing Number: " + missingNumber);
    }
}
```

Output:

```
Missing Number: 5
```

7. Write a Java program to check if a given string is a palindrome

```
public class PalindromeCheck {
    public static boolean isPalindrome(String str) {
        int left = 0;
        int right = str.length() - 1;
        while (left < right) {
            if (str.charAt(left) != str.charAt(right)) {
                return false;
            }
            left++;
            right--;
        }
        return true;
    }

    public static void main(String[] args) {
        String input = "level";
        boolean isPalindrome = isPalindrome(input);
        System.out.println("Is Palindrome: " + isPalindrome);
    }
}
```

Output:

```
Is Palindrome: true
```

8. Write a Java program to merge two sorted arrays into a single sorted array

```
import java.util.Arrays;

public class MergeArrayProgram
{
    private static int[] mergeArray(int[] arrayA, int[] arrayB)
    {
        int[] mergedArray = new int[arrayA.length + arrayB.length];

        int i=0, j=0, k=0;

        while (i < arrayA.length && j < arrayB.length)
        {
            if (arrayA[i] < arrayB[j])
            {
                mergedArray[k] = arrayA[i];
                i++;
                k++;
            }
            else
            {
                mergedArray[k] = arrayB[j];
                j++;
                k++;
            }
        }

        while (i < arrayA.length)
        {
            mergedArray[k] = arrayA[i];
            i++;
            k++;
        }

        while (j < arrayB.length)
        {
            mergedArray[k] = arrayB[j];
            j++;
            k++;
        }

        return mergedArray;
    }

    public static void main(String[] args)
    {
        int[] arrayA = new int[] {1, 2, 3, 4, 5};
```

```
        int[] arrayB = new int[] {6, 7, 8, 9, 10};

        int[] mergedArray = mergeArray(arrayA, arrayB);

        System.out.println("Array A : "+Arrays.toString(arrayA));

        System.out.println("Array B : "+Arrays.toString(arrayB));

        System.out.println("Merged Array : "+Arrays.toString(mergedArray));
    }
}
```

Output:

```
Array A : [1, 2, 3, 4, 5]
Array B : [6, 7, 8, 9, 10]
Merged Array : [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

9. Write a Java program to find the length of the longest substring without repeating characters

```
import java.util.*;

public class LongestSubstringWithoutRepeatingChars {
    public static int findLongestSubstringLength(String s) {
        if (s == null || s.length() == 0) {
            return 0;
        }

        int maxLength = 0;
        Map<Character, Integer> charIndexMap = new HashMap<>();

        for (int i = 0, j = 0; i < s.length(); i++) {
            char currentChar = s.charAt(i);

            if (charIndexMap.containsKey(currentChar)) {
                j = Math.max(j, charIndexMap.get(currentChar) + 1);
            }

            charIndexMap.put(currentChar, i);
            maxLength = Math.max(maxLength, i - j + 1);
        }

        return maxLength;
```

```
    }

    public static void main(String[] args) {
        String input = "abcabcbb";
        int length = findLongestSubstringLength(input);
        System.out.println("Length of the longest substring without repeating
characters: " + length);
    }
}
```

Output:

```
Length of the longest substring without repeating characters: 3
```

10. Write a Java program to check if a given number is prime

```
public class PrimeCheck {
    public 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;
    }

    public static void main(String[] args) {
        int number = 17;
        boolean isPrime = isPrime(number);
        System.out.println(number + " is Prime: " + isPrime);
    }
}
```

Output:

```
17 is Prime: true
```

11. Write a Java program to reverse a singly linked list

```
class Node {
    int data;
    Node next;

    Node(int data) {
        this.data = data;
        this.next = null;
    }
}

public class ReverseLinkedList {
    static Node head;

    public static Node reverseLinkedList(Node current) {
        Node prev = null;
        Node next = null;

        while (current != null) {
            next = current.next;
            current.next = prev;
            prev = current;
            current = next;
        }

        return prev;
    }

    public static void printLinkedList(Node node) {
        while (node != null) {
            System.out.print(node.data + " -> ");
            node = node.next;
        }
        System.out.println("null");
    }

    public static void main(String[] args) {
        head = new Node(1);
        head.next = new Node(2);
        head.next.next = new Node(3);
        head.next.next.next = new Node(4);
        head.next.next.next.next = new Node(5);

        System.out.println("Original Linked List:");
        printLinkedList(head);

        head = reverseLinkedList(head);
```

```
        System.out.println("Reversed Linked List:");
        printLinkedList(head);
    }
}
```

Output:

```
Original Linked List:
1 -> 2 -> 3 -> 4 -> 5 -> null
Reversed Linked List:
5 -> 4 -> 3 -> 2 -> 1 -> null
```

12. Write a Java program to find all the start indices of anagrams of a given string in another string

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class AnagramsStartIndices {
    public static List<Integer> findAnagramStartIndices(String s, String p) {
        List<Integer> result = new ArrayList<>();
        if (s == null || p == null || s.length() < p.length()) {
            return result;
        }

        Map<Character, Integer> pFrequencyMap = new HashMap<>();
        for (char c : p.toCharArray()) {
            pFrequencyMap.put(c, pFrequencyMap.getOrDefault(c, 0) + 1);
        }

        int left = 0;
        int right = 0;
        int count = pFrequencyMap.size();

        while (right < s.length()) {
            char currentChar = s.charAt(right);
            if (pFrequencyMap.containsKey(currentChar)) {
                pFrequencyMap.put(currentChar, pFrequencyMap.get(currentChar) - 1);
                if (pFrequencyMap.get(currentChar) == 0) {
                    count--;
                }
```

```
            }

            while (count == 0) {
                if (right - left + 1 == p.length()) {
                    result.add(left);
                }

                char leftChar = s.charAt(left);
                if (pFrequencyMap.containsKey(leftChar)) {
                    pFrequencyMap.put(leftChar, pFrequencyMap.get(leftChar) + 1);
                    if (pFrequencyMap.get(leftChar) > 0) {
                        count++;
                    }
                }
                left++;
            }

            right++;
        }

        return result;
    }

    public static void main(String[] args) {
        String s = "cbaebabacd";
        String p = "abc";
        List<Integer> indices = findAnagramStartIndices(s, p);
        System.out.println("Start indices of anagrams of '" + p + "' in string '" + s
+ "': " + indices);
    }
}
```

Output:

```
Start indices of anagrams of 'abc' in string 'cbaebabacd': [0, 6]
```

More Java Programs for Interviews

Also, check out below frequently asked Java programs in the interviews:

1. **Java Program to Count Number of Duplicate Words in String**
2. **Java Program to Count Number of Words in Given String**
3. **Java Program to Count the Number of Occurrences of Substring in a String**
4. **Java Program to Count the Occurrences of Each Character in String**

5. Java Program to Merge Two String Arrays
6. Java Program to Remove Duplicate Words from String
7. Java Program to Reverse a String(5 ways)
8. Java Program to Reverse Each Word of a String
9. Java Program to Swap Two Strings
10. How to Check if the String Contains Only Digits
11. How to Check if the String Contains Only Letters
12. How to Check If the String Contains Only Letters or Digits
13. Java Program to Check if Input String is Palindrome
14. Java Program to Find all Permutations of String
15. How to Remove or Trim All White Spaces from a String in Java
16. How to Remove Leading and Trailing White Space From a String in Java
17. Java Program to Count Duplicate Characters in a String
18. Remove Character from String in Java (Java 8)
19. Java Program to Count Vowels and Consonants in a String (Java 8)
20. 4 Ways to Find First Non-Repeated Character in String in Java
21. Java Program to Remove Duplicate Elements in an Array
22. Java Program to Find Largest Element in an Array
23. Java Program to Reverse an Array Without Using Another Array
24. Java Program to Check the Equality of Two Arrays
25. Java Program to Check Armstrong Number
26. Java program to check prime number
27. Java Program to Swap Two Numbers
28. Java Program to Find Factorial of a Number
29. Java Program to Reverse a Number
30. Java Program to Swap Two Strings Without Using Third Variable