


# Java Interview Puzzles and Coding Exercises

 Lokesh Gupta

 September 10, 2023

 Interview Questions

 Interview Questions, Java Puzzle

These Java puzzles and coding exercises and their answers will help you in the next [Java Interview](#). Learn and practice these solutions in advance, to make a better impression in the next interview.

Remember, the key to solving these puzzles is to understand the problem, come up with an efficient algorithm, and write clean and well-structured code.

## 1. FizzBuzz

Write a program that prints the numbers from 1 to 100. But for multiples of 5, print "Fizz" instead of the number, and for multiples of 7, print "Buzz". For numbers that are multiples of both 5 and 7, print "FizzBuzz."

To solve the above puzzle, we need to iterate over the numbers from 1 to 10 and check each number such that:

```
if number is divisible by 5.  
    if number is divisible by 7 the print FizzBuzz.  
    else print Fizz.
```



```
if number is divisible by 7 the print Buzz.  
else print the number.
```

The following Java program demonstrates the usage of the above pseudo steps to solve the *FizzBuzz* problem using the [Java 8 Stream API](#). It uses `rangeClosed(1, 100)` method to iterate over the stream of numbers from 1 to 100. Then we use `mapToObj()` method to check each number against our algorithm and finally print the output.

```
IntStream.rangeClosed(1, 100)  
    .mapToObj(i -> i % 5 == 0 ? (i % 7 == 0 ? "FizzBuzz" : "Fizz") : (i % 7 == 0 ? "Buzz'  
    .forEach(System.out::println);
```

The program output:

```
1  
2  
3  
4  
Fizz  
6  
Buzz  
...  
34  
FizzBuzz  
36  
...
```

## 2. Reverse a string

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

To solve this puzzle, we first convert the string into a character array, then uses two

pointers (start and end) to swap characters from the start and end indices iteratively until they meet at the center. It reverses the whole array.

Finally, we convert the reversed character array back into a string and return it as a string.

```
private static String reverse(String str) {  
  
    char[] chars = str.toCharArray();  
    int start = 0;  
    int end = chars.length - 1;  
  
    while (start < end) {  
        // Swap characters at start and end indices  
        char temp = chars[start];  
        chars[start] = chars[end];  
        chars[end] = temp;  
  
        // Move the indices towards the center  
        start++;  
        end--;  
    }  
  
    return new String(chars);  
}
```



Let us test this function with a simple string.

```
String string = "howtodoinjava";  
  
String reverseString = reverse(string);  
  
System.out.println(reverseString); //avajniidotwoh
```



### 3. Palindrome Program

Write a program to check if a given string is a **palindrome** (reads the same forward and backward).

To check if a string is a palindrome, we need to reverse the string and compare it with the reversed string. If both string matches then the string is palindrome.

We already saw a Java program to reverse the string in the previous question. Either we can use the above reverse() function or we can use the built-in Java APIs for a quick example.

```
String string = "naman";  
String reversed = new StringBuilder(originalString).reverse().toString();  
boolean isPalindrome = string.equalsIgnoreCase(reversed); //prints 'true'
```



Note that we do not care about the case sensitivity in the string comparison, we can use the `equals()` method in place of `equalsIgnoreCase()`.

## 4. Anagram Program

Write a program to check if two given strings are anagrams of each other. An anagram is a word or phrase formed by rearranging the letters of another word or phrase.

There can be different ways to solve this puzzle. For example:

- We iterate over characters of the first string and remove each character from the second array. At the last, if the second string is empty then it means both strings contained the same characters so they are anagrams.
- We sort both strings and compare them. If both strings are equal, it means that the strings are anagrams.

Note that it is a good idea to check the string's length. If the length of both strings is not equal we can safely conclude that strings are not anagrams.

Let us use the second approach to write the solution.

```
private static boolean checkAnagrams(String str1, String str2) {  
  
    if (str1.length() != str2.length()) {  
        return false;  
    }  
  
    char[] charArray1 = str1.toCharArray();  
    char[] charArray2 = str2.toCharArray();  
    Arrays.sort(charArray1);  
    Arrays.sort(charArray2);  
  
    return Arrays.equals(charArray1, charArray2);  
}
```

Let us test this function with an example.

```
String str1 = "listen";  
String str2 = "silent";  
  
System.out.println(checkAnagrams(str1, str2)); //true
```

## 5. HiLo Program

In HiLo game, there are two simple rules:

- Guess the secret number in a maximum of 6 tries.
- The secret number is an integer between 1 and 100, Inclusive.

Everytime, you will guess a number below the secret number (only JRE knows it), "LO" will be printed. Similarly, when you guess a number higher than the secret number, "HI" will be printed. You have to adjust your next guess such that you are able to guess the right

number within six attempts.

We have discussed the solution to this puzzle in a [HiLo Guessing Game](#) article, you can check out.

## 6. Check for balanced parentheses

Write a Java program to check if a given string of parentheses is balanced. For example, "([])" is balanced, but "(())" is not.

This puzzle can be solved using the [Stack](#) data structure. The Stack stores the elements in LIFO order. We can use *LIFO* feature to match the parentheses.

Theoretically, we iterate over all the characters of the given string and everytime an opening parentheses is found, we push it into stack. When the close parentheses is found, we check the last pushed element in the stack and they should match for the same opening/closing parentheses set. For example, '{' will match to '}'.

At last, there should not be any parentheses left in the stack when the program finishes.

```
public static boolean checkBalancedParentheses(String expression) {
    Stack<Character> stack = new Stack<>();

    for (char ch : expression.toCharArray()) {
        if (ch == '(' || ch == '{' || ch == '[') {
            stack.push(ch);
        } else if (ch == ')' || ch == '}' || ch == ']') {

            if (stack.isEmpty()) {
                return false;
            }

            char top = stack.pop();
            if ((ch == ')' && top != '(') ||
                (ch == '}' && top != '{') ||
                (ch == ']' && top != '[')) {
                return false;
            }
        }
    }

    return stack.isEmpty();
}
```



```
        }  
    }  
}  
return stack.isEmpty();  
}
```

Let us test the above function with a correct and an incorrectly balanced expression.

```
String expression1 = "{[ a; ( b; ) c; ]}";  
String expression2 = "{[ a; ( b; ) c; ]}"; //incorrectly matched in last two characters  
  
checkBalancedParentheses(expression1); //true  
checkBalancedParentheses(expression2); //false
```

## 7. Find the missing number

Given an array containing n distinct numbers taken from 0, 1, 2, ..., n, find the missing number. Assume the array is unsorted.

To solve this puzzle, we can use simple mathematics. We add all the numbers from the array and subtract it from the number which is the sum of the numbers if the ano number was missing. The result will be the missing number.

```
int[] numbers = {1, 2, 3, 4, 5, 6, 7, 8, 9, 11, 12};  
  
int N = numbers[numbers.length-1]; //The last element in the array  
int expectedSum = (N * (N + 1)) / 2;  
int actualSum = Arrays.stream(numbers).sum();  
  
int missingNumber = expectedSum - actualSum;  
System.out.println(missingNumber); //Prints '10'
```

## 8. Find duplicates in an array

Given an array of integers, write a Java program to find all the duplicate elements present in the array.

To solve the above puzzle, the simplest way is to create a *MultiMap* like data structure. When we put a key-value pair in a multimap, and the key already exists, it overwrites the previous value with the new value. Rather the new value is added in a list against the same key, so that multimap contains both values for the same key in a list.

Then we check the multimap for each key, and if there is more than 1 value for a key, then the corresponding key or number is a duplicate.

```
Integer[] array = {1, 2, 3, 2, 4, 3, 5, 6, 5, 7, 8, 8, 9};

MultiMap multiMap = new MultiHashMap();
for (int num : array) {
    multiMap.put(num, num);
}

List duplicates = multiMap.keySet().stream()
    .filter(i -> ((ArrayList) multiMap.get(i)).size() > 1)
    .toList();

System.out.println(duplicates); //Prints [2, 3, 5, 8]
```



## 9. Two Sum

Given an array of integers and a target sum, find two numbers that add up to the target sum.

To solve this puzzle, we iterate over the numbers in the array. And for each number:

- subtract the number from the expected *sum* to find the *complement* number.
- check if the array contains the complement number.

Let us write a simple Java program to solve the two-sum problem.





```
private static int[] twoSum(int[] nums, int sum) {  
  
    for (int i = 0; i < nums.length; i++) {  
        int complement = sum - nums[i];  
        int foundAtIndex = Arrays.binarySearch(nums, complement);  
  
        if(foundAtIndex > 0) {  
            return new int[] {nums[i], nums[foundAtIndex]};  
        }  
    }  
    return null;  
}
```

The above function returns an array containing the two numbers which add to the specified sum. Let us test that.



```
int[] nums = {2, 7, 11, 18};  
int sum = 9;  
  
int[] result = twoSum(nums, sum);  
  
System.out.println(Arrays.toString(result));    //[2, 7]
```

## 10. Find the largest element in an array

Write a program to find the largest element in an array of integers.

To find the largest number in an array, we create a variable and initialize it with the first value in the array. Next, we iterate over the array elements and compare each number with the variable. If the number is larger than the variable, we assign the number to the variable.

When the loop finishes, the variable contains the largest value from the array.



```
public static int findLargestElement(int[] array) {
```

```
if (array == null || array.length == 0) {
    throw new IllegalArgumentException("Array is empty or null.");
}

int largest = array[0];

for (int i = 1; i < array.length; i++) {
    if (array[i] > largest) {
        largest = array[i];
    }
}

return largest;
}
```

Let us test the above function with a simple Java program.

```
int[] array = {10, 20, 30, 40, 90, 23, 12, 60};

int largest = findLargestElement(array);
System.out.println("The largest element in the array is: " + largest); //Prints 90
```

## 11. Find the longest substring without repeating characters

Given a string, write a Java program to find the length of the longest substring without repeating characters.

We can solve this puzzle using the sliding window approach. It uses two variables *left* and *right* to track the start and end index of the substring in the specified argument string. Initially, they can be 0 and 0.

A *Set* represents the current substring under test. Initially, the set can be empty.

We iterate through the string while expanding the *right* boundary of the substring. If the current character is already present in the set, it means we have a repeating character and we remove it from the left side of the sliding window and move the left boundary to the

right. This process continues until there are no repeating characters in the current substring.

Finally, we return the size of the Set that represents the largest substring.

```
public static int findLongestSubstringLength(String str) {  
  
    int maxLength = 0;  
    int left = 0;  
    int right = 0;  
    Set<Character> slidingWindow = new HashSet<>();  
  
    while (right < str.length()) {  
  
        char currentChar = str.charAt(right);  
  
        if (slidingWindow.contains(currentChar)) {  
            slidingWindow.remove(str.charAt(left));  
            left++;  
        } else {  
            slidingWindow.add(currentChar);  
            maxLength = Math.max(maxLength, right - left + 1);  
            right++;  
        }  
    }  
  
    return maxLength;  
}
```

Let us test the program.

```
String input = "abcabcbb";  
int length = findLongestSubstringLength(input);  
  
System.out.println(length); //3
```

## 12. Find the common elements in multiple sorted arrays

Given multiple sorted arrays, find the common elements present in all the arrays.

The above puzzle can be solved using different techniques. For example, we can iterate over elements from the first array and check each element in the remaining arrays. If the element is present in all other arrays, we add it to the list of common elements. However, this approach requires a lot of iterations and is not a suitable solution.

Another solution is to find the common elements between two arrays incrementally, and then check the common elements in the third array, and so on.

```
Create a list commonElements and initialize it with 1st array.  
Iterate over the elements of commonElements  
    Search each element in the 2nd array; if element is found, add to the  
temp array;  
    Assign temp array to commonElements  
Repeat
```

Let us write a Java program for the above pseudo-code.

```
public static List<Integer> findCommonElements(Integer[][] arrays) {  
    if (arrays == null || arrays.length == 0) {  
        return List.of(); //empty arguments  
    }  
  
    List<Integer> commonElements = Arrays.asList(arrays[0]);  
  
    for (int i = 1; i < arrays.length; i++) {  
        List<Integer> temp = new ArrayList<>();  
        for (int num : arrays[i]) {  
            if (commonElements.contains(num)) {  
                temp.add(num);  
            }  
        }  
        commonElements = temp;  
    }  
    return commonElements;  
}
```



Now test the above program with a few arrays.

```
Integer[][] arrays = {
    {1, 2, 3, 4, 5},
    {2, 4, 6, 8},
    {2, 3, 4, 7},
    {4, 5, 8, 9}
};

List<Integer> commonElements = findCommonElements(arrays);
System.out.println("Common elements in the arrays: " + commonElements); //Prints [4]
```

### 13. Rotate an array

Write a Java program to rotate an array of size n by k positions to the right.

To solve this puzzle, we can use the following pseudo-code. We will also use an example array to understand it better.

Suppose, we have the initial array [1, 2, 3, 4, 5] and we want to rotate it by 2 positions.

```
Intialize the variables n = array.length (5); and k = 2;
Rotate the complete array -> [5, 4, 3, 2, 1]
Reverse the array from positions 0 to k-1 -> [4, 5, 3, 2, 1]
Reverse the array from positions k to n-1 -> [4, 5, 1, 2, 3]
```

After the program finishes, the array will be reversed by k positions.

Let us write a Java program for the above pseudo-code.

```
public static void rotateArray(int[] array, int k) {
    if (array == null || array.length == 0) {
        return;
    }
}
```

```
int n = array.length;
k = k % n; // Adjust k if it is greater than n

reverseArray(array, 0, n - 1); // Reverse the entire array
reverseArray(array, 0, k - 1); // Reverse the first k elements
reverseArray(array, k, n - 1); // Reverse the remaining n-k elements
}

public static void reverseArray(int[] array, int start, int end) {

    while (start < end) {
        int temp = array[start];
        array[start] = array[end];
        array[end] = temp;
        start++;
        end--;
    }
}
```

We can test the logic with the following example:

```
int[] array = {1, 2, 3, 4, 5};
int k = 2;

System.out.println("Original Array: " + Arrays.toString(array));

rotateArray(array, k);

System.out.println("Rotated Array: " + Arrays.toString(array));
```

Program output:

```
Original Array: [1, 2, 3, 4, 5]
Rotated Array: [4, 5, 1, 2, 3]
```

## 14. More Puzzles

- [Dead code and unreachable code in Java](#)

- [How to create an instance of any class without using the \*new\* keyword](#)
- [How to Detect infinite loop in LinkedList](#)
- [TreeMap put operation puzzle](#)
- [Good String – Bad String](#)
- [Check if the string is complete \(contains all alphabets\)](#)
- [Return all the strings with the Nth longest length](#)

Do not forget to share more such puzzles if you have been asked – and you think they may help others.

Happy Learning !!

[Source Code on Github](#)

## Further Reading:

- [Python Interview Questions and Answers](#)
- [Java Concurrency Interview Questions](#)
- [Python Numpy 101: A Beginners Guide](#)
- [Reverse an Array in Java](#)
- [Java NIO Buffer Tutorial](#)
- [Merge Sort – Algorithm, Implementation and Performance](#)

## Comments

Subscribe

---



Be the First to Comment!

**0 COMMENTS**

---

Search ...

### Weekly Newsletter

Stay Up-to-Date with Our  
Weekly Updates. Right into  
Your Inbox.

Email Address

Subscribe









## About Us

*HowToDoInJava* provides tutorials and how-to guides on Java and related technologies.

It also shares the best practices, algorithms & solutions and frequently asked interview questions.

## Tutorial Series

[OOP](#)

[Regex](#)

[Maven](#)

[Logging](#)

[TypeScript](#)

[Python](#)

## Meta Links

[About Us](#)

[Advertise](#)

[Contact Us](#)

[Privacy Policy](#)

## Our Blogs

## REST API Tutorial

**Follow On:**

**Dark Mode**



Copyright © 2024 | [Sitemap](#)