PITTALA ARUN KUMAR

Java Full Stack - Coding Assessment 34's report

Submitted on May 11 2023 21:48:51 IST

150.0 (100%)
scored out of 150

Completed
in the assignment

3
problems attempted out of 3

3.4 / 5
avg. code quality score

Severe Violation
flagged by DoSelect proctoring engine

Test time analysis

1h 0m 20s
time taken for completion

May 09 2023 15:25:59 IST
test invite time

May 11 2023 20:48:31 IST
test start time

May 11 2023 21:48:51 IST
test end time

Performance summary

3
solutions accepted

Proctor analysis

0
browser used

0
navigation violation

2
webcam violations

0 min
no test window violation

Solutions

Problem Name	Problem Type	Status	Score
Insert Space [Lab 10 Ex-2]	Coding	ACCEPTED	**50.0** / 50
Infinite String	Coding	ACCEPTED	**50.0** / 50
Sarah in Amsterdam	Coding	ACCEPTED	**50.0** / 50

Technology used

Java

Additional Information

Question	Response
Enrollment Number	EBEON0223750433
Batch Code (Eg : 2022-XXXX)	2022-8938

Detailed Report

Problem 1 : Insert Space [Lab 10 Ex-2]

CODING | SCORE: **50**

Your task here is to implement a **Java** code based on the following specifications. Note that your code should match the specifications in a precise manner. Consider default visibility of classes, data fields and methods unless mentioned otherwise.

Specifications:

```
class definitions:class Source:
   visibility: public
   method definition:
    insertSpace(String s): method that uses lambda expression to format a
given string, where a space is inserted between each character of string.
     return type: String
```

Task

Create a **Source** class and implement below given method:

- **insertSpace(String s):** Use lambda expression to format a given string, where a space is inserted between each character of string

Implement using **Lambda expressions.**

NOTE

- Do not use any **for** loops or other control structures.
- Use the **Stream API** methods for your implementations, else the test-cases might fail.

Sample Input

```
capgemini
```

Sample Output

```
c a p g e m i n i
```

NOTE:

- The above **Sample Input** and **Sample Output** are only for demonstration purposes and will be obtained if you implement the **main()** method with all method calls accordingly.
- Upon implementation of **main()** method, you can use the **RUN CODE** button to pass the **Sample Input** as input data in the method calls and arrive at the **Sample Output**.

Solution

ACCEPTED | SCORE: **50.0** / 50

Code Quality Analysis

Maintainable code

Quality score: 4.1

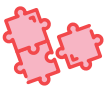
Deep Code Analysis Results

Straightforward approach

No cyclomatic constructs detected.

Modular code

Sufficient reusable components found.



Very low extensibility

The code is difficult to extend.

Java 8

```java
@FunctionalInterface
interface StringSpace {
    String addSpace(String s);
}


public class Source {

    public static void main(String[] args) {
        String input="capgemini";
        System.out.println(Source.insertSpace(input));
    }
    static String insertSpace(String s)
    {
        StringSpace space=str->str.chars()
        .mapToObj(c->String.valueOf((char) c))
        .reduce("",(str1,str2)->str1+" "+str2);
        String result=space.addSpace(s).trim();
        return result;
    }
}
```

Evaluation Details

Test_Method *(weight:1)*

Status	Passed
Execution time	2.88s
CPU	0s
Memory	1MB
Description	Testcase passed!
Annotation	Correct code

Test_Space *(weight:1)*

Status	Passed
Execution time	3.62s
CPU	0s
Memory	1MB
Description	Testcase passed!
Annotation	Correct code

Sample_TC *(sample)*

Status	Passed
Execution time	3.20s
CPU	0s
Memory	1MB
Description	Testcase passed!

Problem 2 : Infinite String

CODING | SCORE: **50**

Problem Statement

In this problem you will be given a string ***S*** , consisting of lowercase alphabets *(a-z)*, in which each character is unique. Another string ***INF*** is formed by repeating the string *S* infinitely many times.

Example: If ***S*** = *"abcde"* then the string ***INF*** is *...abcdeabcdeabcde...* Here the dots ('.') indicate that there are infinitely many characters before and after the string.

Now you will be given another string ***A*** and asked to find whether there is any sub-string in ***INF*** which is identical to ***A***.

Input Format

- The first line contains the number of test-cases **T**.
- The next **T** lines will contain a space-separated string, made up of two parameters:
- The first parameter will be **infStr**, representing **S** from the above example
- The second parameter will be **toFind**, representing **A** from the above example.

Output Format

- The function should print **YES** if *A* can be found in *S*, otherwise it should print **NO**.

e.g., If the second line of input contains:

```
abcd abce
```

- The function should print **NO**, because, if we repeat **"***abcd***"** infinitely many times we will get, **"***...abcdabcdabcdabcd...***"** . We will never get an **"***e***".**

Evaluation Parameters

Sample Input

```
ghijk ghijkghi
```

Sample Output

```
YES
```

Explanation

- The infinite string of '*ghijk*' contains '*ghijkghi*' as it's sub-string, hence you print **YES**.

Solution

ACCEPTED | SCORE: **50.0** / 50

Code Quality Analysis

Minor quality violations

Quality score: 3.2

Deep Code Analysis Results

Straightforward approach

No cyclomatic constructs detected.

Very low modularity

No reusable components found.

Low extensibility

Some extensible features detected.

Java 8

```java
import java.io.*;

 class Source
{
        public static int inf_string(String a, String b)
        {
                // return 1 if the string 'a' can be contained in 'b'.
                StringBuilder strbldr=new StringBuilder();
                while(strbldr.length()<(b.length()*2))
                {
                    strbldr.append(a);
                }
                return strbldr.toString().contains(b)?1:0;
        }

        public static void main (String[] args) throws
IOException,java.lang.NumberFormatException
        {
            int t;
                BufferedReader input = new BufferedReader (new InputStreamReader (System.in));
                t =Integer.parseInt(input.readLine());


                while(t > 0)
                {
                        String ab = input.readLine();
                        //String b = input.readLine();
                        String a=ab.split(" ")[0];
                        String b=ab.split(" ")[1];
                        int ans = inf_string(a, b);
                        if(ans == 1)
                        {
                                System.out.println("YES");
                        }
                        else
                        {
                                System.out.println("NO");
                        }
                        t--;
                }
        }
}
```

Evaluation Details

Testcase #1 *(weight:1)*

Status	Passed
Execution time	0.30s
CPU	0s
Memory	2MB
Description	Testcase passed! The solution's output matches the expected output.
Annotation	Solution implements the correct logic.

Testcase #5 *(weight:1)*

Status	Passed
Execution time	0.93s
CPU	0s
Memory	28MB
Description	Testcase passed! The solution's output matches the expected output.
Annotation	Solution passes for the larger test case.

Testcase #2 *(weight:1)*

Status	Passed
Execution time	0.32s
CPU	0s
Memory	2MB
Description	Testcase passed! The solution's output matches the expected output.
Annotation	Solution implements the correct logic.

Testcase #4 *(weight:1)*

Status	Passed
Execution time	1.09s
CPU	0s
Memory	29MB
Description	Testcase passed! The solution's output matches the expected output.
Annotation	Solution passes for the larger test case.

Testcase #6 *(sample)*

Status	Passed
Execution time	0.32s
CPU	0s
Memory	2MB
Description	Testcase passed! The solution's output matches the expected output.
Annotation	Solution implements the correct logic.

Input

```
2
abcd abce
abcde eabcdeab
```

Solution output

```
NO
YES
```

Expected output

```
NO
YES
```

Testcase #3 *(weight:1)*

Status	Passed
Execution time	0.74s
CPU	0s
Memory	27MB
Description	Testcase passed! The solution's output matches the expected output.
Annotation	Solution implements the correct logic.

Problem 3 : Sarah in Amsterdam

CODING | SCORE: **50**

Sarah is planning to spend a week at her friend's summer house in Amsterdam. Sarah is not sure if her father will allow her to go with the friends. However, Sarah went to her father and ask for his permission. Sarah's father works at DoSelect as a Programmer. He came to Sarah with one condition, he wants Sarah to code something related to "AMSTERDAM" and if she does it correctly, she will be allowed to go.

Sarah's father gave a string and wants her to write a function that returns the number of times "am" appears in the String ignoring the case

Sarah is not so good at programming and needs your help.

Your task here is to implement a **Java** code based on the following specifications. Note that your code should match the specifications in a precise manner. Consider **default visibility** of classes, data fields and methods are public unless mentioned otherwise.

Specifications

```
class definitions:
 class InAmsterdam:
 countAm(String str):
  return type: int
  visibility: public
```

Task:

class InAmsterdam

Implement the below method for this class:

- **int countAm(String str):** return the **number of times "am" appears** in the String ignoring the case

Sample Input

```
I Am in Amsterdam am I?
```

Sample Output

```
2
```

NOTE

- You can make suitable function calls and use **the RUN CODE** button to check your **main()** method output.

Solution

ACCEPTED | SCORE: **50.0** / 50

Code Quality Analysis

Minor quality violations

Quality score: 3.0

Deep Code Analysis Results

Straightforward approach

No cyclomatic constructs detected.

Low modularity

Some reusable components found.

Low extensibility

Some extensible features detected.

Java 8

```java
import java.io.*;
import java.util.*;
import java.text.*;
import java.math.*;
import java.util.regex.*;

class InAmsterdam {
  //Write Your Code Here..
  public int countAm(String str){
      String lowercaseStr=str.toLowerCase();
      String [] words=lowercaseStr.split("\\s+");
      int count=0;
      for(String wd:words)
      {
          if(wd.equals("am"))
          {
              count++;
          }
      }
      return count;
  }
}

public class Source {
        public static void main(String args[] ) throws Exception {
                /* Enter your code here. Read input from STDIN. Print output to STDOUT */
                InAmsterdam inAm=new InAmsterdam();
                String str="I Am in Amsterdam am I?";
                int count=inAm.countAm(str);
                System.out.println(count);
        }
}
```

Evaluation Details

Test_countAm_3 *(weight:1)*

Status	Passed
Execution time	2.69s
CPU	0s
Memory	1MB

Description	Testcase passed!

Test_countAm_2 *(weight:1)*

Status	Passed
Execution time	2.69s
CPU	0s
Memory	1MB
Description	Testcase passed!

Test_InAmsterdam *(weight:1)*

Status	Passed
Execution time	2.63s
CPU	0s
Memory	1MB
Description	Testcase passed!

Test_countAm_4 *(weight:1)*

Status	Passed
Execution time	2.58s
CPU	0s
Memory	1MB
Description	Testcase passed!

Sample_TC *(sample)*

Status	Passed
Execution time	3.73s
CPU	0s
Memory	1MB
Description	Testcase passed!

Test_countAm_1 *(weight:1)*

Status	Passed

Execution time	2.74s
CPU	0s
Memory	1MB
Description	Testcase passed!