PITTALA ARUN KUMAR

Java Full Stack - Coding Assessment 35's report

Submitted on Jun 03 2023 09:33:44 IST

133.3 (89%)
scored out of 150

Completed
in the assignment

3
problems attempted out of 3

2.3 / 5
avg. code quality score

Severe Violation
flagged by DoSelect proctoring engine

Test time analysis

2h 0m 0s
time taken for completion

Jun 03 2023 07:31:32 IST
test invite time

Jun 03 2023 07:33:44 IST
test start time

Jun 03 2023 09:33:44 IST
test end time

Performance summary

2
solutions accepted

1
solution partially 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
Job Seekers	Coding	ACCEPTED	**50.0** / 50
Paper Wasp	Coding	PARTIALLY ACCEPTED	**33.3** / 50
Exception in Age	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 : Job Seekers

CODING | SCORE: **50**

You are asked to create an application for registering the details of a job-seeker. The requirement is:

- Username should always end with _job and there should be at least minimum of 8 characters to the left of _job. Write a function to validate the same.

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 definitons:
       validate(String s): Return true in case the validation is passed. In cas
e of validation failure return false.
          return type: boolean
          visibility: public
```

Task

Create a class **Source** according to above specifications and implement the below given method:

- **boolean validate(String s):** Return true in case the validation is passed. In case of validation failure return false.

Sample Input 1

```
capgemini
```

Sample Input 2

```
capgemini_job
```

Sample Output 1

```
false
```

Sample Output 2

```
true
```

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

Many quality violations

Quality score: 2.4

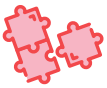
Deep Code Analysis Results

Straightforward approach

No cyclomatic constructs detected.

Low modularity

Some reusable components found.



Very low extensibility

The code is difficult to extend.

Java 8

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

// Class name should be "Source",
// otherwise solution won't be accepted
public class Source {
        public static void main(String args[] ) {
            Scanner scanner =new Scanner(System.in);
            String s=scanner.nextLine();
            boolean result=validate(s);
            System.out.println(result);
        }
            public static boolean validate(String s){
                if(s.endsWith("_job")&&s.length()>=8){
                    return true;
                }
                else
                {
                    return false;
                }
            }

                /* Enter your code here. Read input from STDIN. Print output to STDOUT */
}
```

Evaluation Details

Test_Methods *(weight:1)*

Status Passed

Execution time	2.59s
CPU	0s
Memory	1MB
Description	Testcase passed!

Sample_TC *(sample)*

Status	Passed
Execution time	2.81s
CPU	0s
Memory	1MB
Description	Testcase passed!

Test_Job *(weight:1)*

Status	Passed
Execution time	2.60s
CPU	0s
Memory	1MB
Description	Testcase passed!

Problem 2 : Paper Wasp

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 Insect:
  data members:
    String insectName;
    int insectWeight;
     visibility: private

   Insect(String insectName, int insectWeight): constructor with public
visibility
    Define getters and setters with public visibility
    toString(): has been implemented for you

 class Insecticides:
  method definition:
   mapInsectstName(List<Insect> list):
    return type: List<String>
    visibility: public

   getWeight(List<Insect> list):
    return type: List<Insect>
    visibility: public
```

Task:

class Insect:

- define class **Student** according to the above specifications

class **Insecticides:**

Implement the below method for this class:

- **List<String>mapInsectsName(List<Insect> list):** fetch and return the Insect name from the list
- **List<Insect>getWeight(List<Insect> list):** filter the weight from the list greater than **40** and less than equal to **100**, put it into a list and return the desired list

> *Refer sample output for clarity*

Sample Input

```
 Insecticides i = new Insecticides();
  List<Insect> list = new ArrayList<Insect>();
         list.add(new Insect("Ant", 45));
         list.add(new Insect("Cockroach", 65));
         list.add(new Insect("bee", 99));
         list.add(new Insect("paper wasp", 11));
-----------------------------------------------------------
```

```
  i.mapInsectstName(list)
  i.getWeight(list)
```

Sample Output

```
[Ant, Cockroach, bee, paper wasp]
-----------------------------------------------------------
[Insect{insectName='Ant', insectWeight=45}, Insect{insectName='Cockroach',
insectWeight=65}, Insect{insectName='bee', insectWeight=99}]
```

NOTE

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

Solution

PARTIALLY ACCEPTED | SCORE: **33.3** / 50

Code Quality Analysis

Many quality violations

Quality score: 1.7

Deep Code Analysis Results

Straightforward approach

No cyclomatic constructs detected.

Low modularity

Some reusable components found.

Extensible implementation

The code is easy to extend.

Java 8

```
1  import java.io.*;
2  import java.util.*;
3  import java.text.*;
4  import java.math.*;
5  import java.util.regex.*;
6
7  class Insect {
8    //Code Here..
9    private String insectName;
10   private int insectWeight;
11   public Insect(String insectName,int insectWeight){
12       this.insectName=insectName;
13       this.insectWeight=insectWeight;
14   }
15
16       public String getInsectName(){
17           return insectName;
18
19       }
20       public void setInsectName(String insectName){
```

```
21            this.insectName=insectName;
22        }
23        public int getInsectWeight(){
24            return insectWeight;
25        }
26        public void setInsectWeight(int insectWeight){
27            this.insectWeight=insectWeight;
28        }
29
30
31
32        @Override
33      public String toString() {
34          return "Insect{" +
35                  "insectName='" + insectName + '\'' +
36                  ", insectWeight=" + insectWeight +
37                  '}';
38      }
39  }
40
41  class Insecticides {
42   //Code Here..
43   public List<String>mapInsectsName(List<Insect>list){
44       List<String>result=new ArrayList<>();
45       for(Insect i:list){
46           result.add(i.getInsectName());
47       }
48       return result;
49   }
50
51
52   public List<Insect>getWeight(List<Insect>list){
53       List<Insect>result=new ArrayList<>();
54       for(Insect i:list) {
55           if(i.getInsectWeight()>40 && i.getInsectWeight()<=100){
56           result.add(i);
57       }
58   }
59   return result;
60   }
61   }
62  public class Source {
63          public static void main(String args[] ) throws Exception {
64                  /* Enter your code here. Read input from STDIN. Print output to STDOUT */
65                  Insecticides i=new Insecticides();
66                  List<Insect> list=new ArrayList<Insect>();
67                  List<String>result=new ArrayList<String>();
68                  list.add(new Insect("Ant",45));
69                  list.add(new Insect("Cockroach",65));
70                  list.add(new Insect("bee",99));
71                  list.add(new Insect("paper wasp",11));
72                  System.out.println(i.mapInsectsName(list));
73                  System.out.println(i.getWeight(list));
74                  for(Insect in:list){
75                      result.add(in.toString());
76                  }
77                  System.out.println(result);
78
79          }
80  }
```

Evaluation Details

Test_Insecticides *(weight:1)*

Status	Passed
Execution time	2.82s

CPU	0s
Memory	1MB
Description	Testcase passed!

Test_getWeight2 *(weight:1)*

Status	Passed
Execution time	2.77s
CPU	0s
Memory	1MB
Description	Testcase passed!

Test_Insect *(weight:1)*

Status	Passed
Execution time	3.01s
CPU	0s
Memory	1MB
Description	Testcase passed!

Test_getWeight1 *(weight:1)*

Status	Passed
Execution time	2.90s
CPU	0s
Memory	1MB
Description	Testcase passed!

Test_mapInsectstName2 *(weight:1)*

Status	Failed
Execution time	3.08s
CPU	0s
Memory	432kB
Description	Testcase failed.

Evaluation logs

```
eval.java:13: error: cannot find symbol
assertEquals("[, ]",String.valueOf( i.mapInsectstName(list)));
^
symbol: method mapInsectstName(List<Insect>)
location: variable i of type Insecticides
1 error
```

Test_mapInsectstName1 *(weight:1)*

Status	Failed
Execution time	3.47s
CPU	0s
Memory	432kB
Description	Testcase failed.

Evaluation logs

```
eval.java:13: error: cannot find symbol
assertEquals("[Hopper, cvvfd]",String.valueOf( i.mapInsectstName(list)));
^
symbol: method mapInsectstName(List<Insect>)
location: variable i of type Insecticides
1 error
```

Sample_TC *(sample)*

Status	Failed
Execution time	4.72s
CPU	0s
Memory	432kB
Description	Testcase failed.

Evaluation logs

```
eval.java:15: error: cannot find symbol
assertEquals("[Ant, Cockroach, bee, paper wasp]",String.valueOf(
i.mapInsectstName(list)));
^
symbol: method mapInsectstName(List<Insect>)
location: variable i of type Insecticides
1 error
```

Problem 3 : Exception in Age

CODING | SCORE: **50**

Write a java program to validate the age of a person and display proper message by using user defined exception. Age of a person should be above 15.

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 MyException: Define exception
  class Source:
    method definitons:
      checkAge(int age): throw a user defined exception if age is smaller than
15
        visibility: public
```

Task

- Define **MyException** class
- Create a class **Source** and implement the below given method
- **String checkAge(int age)**: throw a user defined exception if age is smaller than 15

Sample Input

```
22
```

Sample Output

```
valid
```

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

Minor quality violations

Quality score: 2.8

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.util.*;
public class Source{
    public static void main(String args[]){
        Scanner sc=new Scanner(System.in);
        int age=sc.nextInt();
        try
        {
            checkAge(age);
        }
        catch(MyException e){
            e.getMessage();
        }
    }
    public static String checkAge(int age)throws MyException{
        if(age<15)
        throw new MyException("age below 15");
        else
        return "valid";
    }
}
class MyException extends Exception{
    MyException(String s){
        super(s);
    }
}
```

Evaluation Details

Test_Methods_Source *(weight:1)*

Status	Passed
Execution time	2.54s
CPU	0s
Memory	1MB
Description	Testcase passed!

Sample_TC *(sample)*

Status	Passed
Execution time	2.74s
CPU	0s
Memory	1MB

Description	Testcase passed!

Test_Valid *(weight:1)*

Status	Passed
Execution time	2.95s
CPU	0s
Memory	1MB
Description	Testcase passed!

Test_Methods_MyException *(weight:1)*

Status	Passed
Execution time	2.73s
CPU	0s
Memory	1MB
Description	Testcase passed!