PITTALA ARUN KUMAR

Java Full Stack - Coding Assessment 36's report

Submitted on Jun 03 2023 11:56:58 IST

222.2 (74%)
scored out of 300

Completed
in the assignment

3
problems attempted out of 3

2.4 / 5
avg. code quality score

Severe Violation
flagged by DoSelect proctoring engine

Test time analysis

34m 45s
time taken for completion

Jun 03 2023 11:20:51 IST
test invite time

Jun 03 2023 11:22:12 IST
test start time

Jun 03 2023 11:56:58 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
Distance Traveled	Coding	ACCEPTED	**100.0** / 100
The classroom	Coding	PARTIALLY ACCEPTED	**22.2** / 100
Exception in Name [5.4]	Coding	ACCEPTED	**100.0** / 100

Technology used

Java

Additional Information

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

Detailed Report

Problem 1 : Distance Traveled

CODING | SCORE: **100**

Problem Statement

Write a program to calculate the distance travelled by car at different time intervals. The initial velocity of the car is 36 km/hr and the constant acceleration is 5 m/s2.

The formula to calculate distance is:

> *Distance Travelled = u*t+((a*t*t)/2) where,*

> *u = initial velocity of the car (36 km/hr)*

> *a = acceleration of the car (5 m/s^2)*

> *t = time duration in seconds*

The program should accept 2-time intervals as the input (one-time interval per line) and print the distance travelled in meters by car (one output per line).

Definitions:

> *1 kilometer = 1000 meters*

> *1 hour = 3600 seconds*

Sample Input

```
10
8
```

Sample Output

```
350
240
```

Solution

ACCEPTED | SCORE: **100.0** / 100

Code Quality Analysis

Many quality violations

Quality score: 2.2

Deep Code Analysis Results

Straightforward approach

No cyclomatic constructs detected.

Low modularity

Some reusable components found.

Low extensibility

Some extensible features detected.

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 name should be "Source",
8  // otherwise solution won't be accepted
9  public class Source {
10     public static void main(String[] args) {
11
12         Scanner in = new Scanner(System.in);
13
14         // Declare the variable
15         int a;
16         int b;
17
18         // Read the variable from STDIN
19         a = in.nextInt();
20         b = in.nextInt();
21
22         // Output the variable to STDOUT
23         calcSpeed(a);
24         calcSpeed(b);
25     }
26     public static void calcSpeed(int t){
27         int u=10;
28         int a=5;
29         int distance_travelled=u*t+((a*t*t)/2);
30         System.out.println(distance_travelled);
31
32    }
33 }
```

Evaluation Details

Testcase #1 *(sample)*

Status	Passed
Execution time	0.46s
CPU	0s
Memory	22MB
Description	Testcase passed! The solution's output matches the expected output.

Input

```
10
8
```

Solution output

```
350
240
```

Expected output

```
350
240
```

Testcase #2 *(weight:1)*

Status	Passed
Execution time	0.47s
CPU	0s
Memory	22MB
Description	Testcase passed! The solution's output matches the expected output.

Problem 2 : The classroom

CODING | SCORE: **100**

Your task here is to implement **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.

Specifications

```
class definitions:
 class Student:
  data members:
   String name
   int score

  Student(String name, int score): constructor with public visibility

 class Classroom:
  method definition:
   registerStudent(Student student):
    return : String
    visibility:  public

   studentCard(String card):
    return : String
    visibility : public
```

class Student

- define data members according to the above specifications

class Classroom

- define data members according to the above specifications

-Implement the below methods for this class:

-String registerStudent(Student student):

- Write a code to validate the names and marks of the students according to the below specifications.
- The name must be in uppercase, if not return "**Block letters needed**".
- If the score is not between 0 to 100 then return **"Invalid score".**
- **If the above conditions are satisfied** then return "**Registered**".

-String studentCard(String card):

- The given string contains **only numbers**(no alphabets and special symbols allowed).
- If it has any other character return "**Invalid card" else return "Valid card".**

Sample Input

```
Student s1=new Student("A",6);
Classroom cm=new Classroom();
cm.registerStudent(s1);
```

Sample Output

```
Registered
```

NOTE:

- You can make suitable function calls and use the **RUN CODE** button to check your **main()** method output.
- Make sure that all the strings in the return statement are case-sensitive.

Solution

PARTIALLY ACCEPTED | SCORE: **22.2** / 100

Code Quality Analysis

Minor quality violations

Quality score: 2.6

Deep Code Analysis Results

Straightforward approach

No cyclomatic constructs detected.

Modular code

Sufficient reusable components found.

Low extensibility

Some extensible features detected.

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 Student{
8  //Code Here..
9  String name;
10 int score;
11 public Student (String name,int score){
12     this.name=name;
13     this.score=score;
14 }
15 }
16
17 class Classroom{
18 //Code Here..
```

```
public String registerStudent(Student student){
    if(!student.name.equals(student.name.toUpperCase())){
        return "Block letters needed";
    }
    if(student.score<0||student.score>100){
        return "invalid score";
    }
    return "Valid card";
}
}

public class Source {
        public static void main(String args[] ) throws Exception {
                /* Enter your code here. Read input from STDIN. Print output to STDOUT */
                Student s1=new Student("A",6);
                Classroom cm=new Classroom();
                String output=cm.registerStudent(s1);
                System.out.println(output);
        }
}
```

Evaluation Details

Testcase #clsClassroom *(weight:1)*

Status	Failed
Execution time	2.33s
CPU	0s
Memory	1MB
Description	Testcase failed.

Evaluation logs

```
Exception in thread "main" java.lang.AssertionError
at eval.main(eval.java:8)
```

Testcase #studentCard2 *(weight:1)*

Status	Failed
Execution time	2.59s
CPU	0s
Memory	432kB
Description	Testcase failed.

Evaluation logs

```
eval.java:6: error: cannot find symbol
assert cm.studentCard("23dwq").equals("Invalid card");
^
symbol: method studentCard(String)
location: variable cm of type Classroom
1 error
```

Testcase #Sample *(sample)*

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

Testcase #clsStudent *(weight:1)*

Status	Passed
Execution time	2.27s
CPU	0s
Memory	1MB
Description	Testcase passed!

Testcase #studentCard3 *(weight:1)*

Status	Failed
Execution time	2.43s
CPU	0s
Memory	436kB
Description	Testcase failed.

Evaluation logs

```
eval.java:6: error: cannot find symbol
assert cm.studentCard("23234").equals("Valid card");
^
symbol: method studentCard(String)
location: variable cm of type Classroom
1 error
```

Testcase #registerStudent1 *(weight:1)*

Status	Failed
Execution time	2.53s
CPU	0s
Memory	1MB
Description	Testcase failed.

Evaluation logs

```
Exception in thread "main" java.lang.AssertionError
at eval.main(eval.java:6)
```

Testcase #registerStudent4 *(weight:1)*

Status	Failed
Execution time	2.41s
CPU	0s
Memory	1MB
Description	Testcase failed.

Evaluation logs

```
Exception in thread "main" java.lang.AssertionError
at eval.main(eval.java:9)
```

Testcase #registerStudent2 *(weight:1)*

Status	Passed
Execution time	2.34s
CPU	0s
Memory	1MB
Description	Testcase passed!

Testcase #registerStudent3 *(weight:1)*

Status	Failed
Execution time	2.24s
CPU	0s
Memory	1MB
Description	Testcase failed.

Evaluation logs

```
Exception in thread "main" java.lang.AssertionError
at eval.main(eval.java:6)
```

Testcase #studentCard *(weight:1)*

Status	Failed

Execution time	2.21s
CPU	0s
Memory	436kB
Description	Testcase failed.

Evaluation logs

```
eval.java:6: error: cannot find symbol
assert cm.studentCard("23d#$@$").equals("Invalid card");
^
symbol: method studentCard(String)
location: variable cm of type Classroom
1 error
```

Problem 3 : Exception in Name [5.4]

CODING SCORE: **100**

Write a Java Program to validate the full name of an employee. Create and throw a user defined exception if **firstName** and **lastName** is blank.

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:
       checkName(String firstName,String lastName)throw a user defined
exception if firstName and lastName is blank else return name
         return type: String
         visibility: public
```

Task

- Define **MyException** class
- Create a class **Source** and implement the below given method
- **checkName(String firstName,String lastName):** throw a user defined exception if firstName and lastName is blank else return name

Sample Input

```
Alan Hasley
```

Sample Output

```
AlanHasley
```

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: **100.0** / 100

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.

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 name should be "Source",
// otherwise solution won't be accepted
public class Source {
        public static void main(String args[] ) throws Exception {
                /* Enter your code here. Read input from STDIN. Print output to STDOUT */
                Scanner sc=new Scanner(System.in);
                String firstName=sc.next();
                String lastName=sc.next();
                System.out.println(checkName(firstName,lastName));
        }
        public static String checkName(String firstName,String lastName) throws MyException{
            if(firstName.equals(" ") || lastName.equals(" "))
            throw new MyException("Invalid name");
            return (firstName+lastName);
        }
}
        class MyException extends Exception{
            public MyException(String message){
                super(message);
            }
        }
```

Evaluation Details

Test_Methods_MyEXception *(weight:1)*

Status	Passed
Execution time	2.46s
CPU	0s
Memory	1MB
Description	Testcase passed!

Test_Name *(weight:1)*

Status	Passed
Execution time	2.57s
CPU	0s
Memory	1MB
Description	Testcase passed!

Sample_TC *(sample)*

Status	Passed
Execution time	2.85s
CPU	0s
Memory	1MB
Description	Testcase passed!

Test_Methods_Source *(weight:1)*

Status	Passed
Execution time	2.41s
CPU	0s
Memory	1MB
Description	Testcase passed!