
# BARLA ANILKUNAR

## Java Full Stack_Coding Assessment_14's report

Submitted on Dec 05 2023 20:58:02 IST

**113.2 (75.5%)**
scored out of 150

**Completed**
in the assignment

**3**
problems attempted out of 3

**2.0 / 5**
avg. code quality score

**Severe Violation**
flagged by DoSelect proctoring engine

## Test time analysis

**1h 22m 50s**
time taken for completion

**Dec 05 2023 19:30:53 IST**
test invite time

**Dec 05 2023 19:35:11 IST**
test start time

**Dec 05 2023 20:58:02 IST**
test end time

## Performance summary

**3**
solutions partially accepted

## Proctor analysis

**0**
browser used

**0**
navigation violation

**2**
webcam violations

**0 min**
no test window violation

**Webcam Violation** - flagged by DoSelect Proctoring Engine due to below reasons

## Total Frames Captured : 0

| | |
|---|---|
| Frames with Matching Faces | **0** |

| | |
|---|---|
| Frames with Multiple Faces | **0** |
| Frames with Different Face | **0** |
| Frames with No Face | **0** |

## Total Frames Missing : 4970

| | |
|---|---|
| Webcam not detected | **0** |
| Test-taker closing the tab | **0** |
| Network Issues | **0** |
| Other factors* | **4970** |

## Total Webcam Violations : 2

| | |
|---|---|
| Set of 10 back-to-back Suspicious Frames** | **0** |
| Set of 10 back-to-back Missing frames | **1** |
| Suspicious Frames**/Missing Frames detected in more than 10% of test duration | **1** |

* Missing frames due to other factors such as test-taker's system issues etc

** Suspicious frames includes Multiple Faces, Different Faces and No Face

## Identity Image

## Solutions

| Problem Name | Problem Type | Status | Score |
|---|---|---|---|
| **Count Characters [Lab 3 Ex-4]** | Coding | PARTIALLY ACCEPTED | **25.0** / 50 |
| **Shop Online** | Coding | PARTIALLY ACCEPTED | **43.8** / 50 |
| **The classroom** | Coding | PARTIALLY ACCEPTED | **44.4** / 50 |

## Technology used


**Java**

## Additional Information

| Question | Response |
|---|---|
| **Enrollment Number** | EBEON0923842377 |
| **Batch Code (Eg : 2022-XXXX)** | 2023-10433 |

# Detailed Report

## Problem 1 : Count Characters [Lab 3 Ex-4]

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:
   countChars(char[] arr): method that accepts a character array and count the
number of times each character is present in the array.
    return type: Map<Character, Integer>
    visibility: public
```

**Task:**

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

- **countChars(char[] arr):** accept a character array and count the number of times each character is present in the array.

**Sample Input**

```
'a', 'f', 'c', 'd', 'a', 'c'
```

**Sample Output**

```
{a=2, c=2, d=1, f=1}
```

**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

PARTIALLY ACCEPTED | SCORE: **25.0** / 50

## Code Quality Analysis

### Minor quality violations

Quality score: 3.0

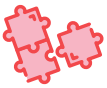
## Deep Code Analysis Results

### Straightforward approach

No cyclomatic constructs detected.

### Very low modularity

No 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.*;

// Class name should be "Source",
// otherwise solution won't be accepted
public class Source {
    static Map<Character,Integer> countChars(char[] arr){
        Map<Character,Integer> charCount = new HashMap<>();

        for(char c:arr){
            if(charCount.containsKey(c)){
                int count =charCount.get(c);
                charCount.put(c,count+1);
            }else{
                charCount.put(c,1);
            }
            }
        return charCount;
    }
        public static void main(String args[] ) throws Exception{
           BufferedReader br=new  BufferedReader(new InputStreamReader(System.in));

            String input=br.readLine();
            Map<Character,Integer> result=countChars(input.toCharArray());
            StringBuilder output=new StringBuilder("{");
            for(Map.Entry<Character,Integer> entry : result.entrySet()){

output.append(entry.getKey()).append("=").append(entry.getValue()).append(",");
            }
            output.setLength(output.length()-2);
            output.append("}");
            System.out.println(output);
        }
}
```

## Evaluation Details

**Test_Method** *(weight:1)*

| | |
|---|---|
| **Status** | Failed |
| **Execution time** | 2.71s |
| **CPU** | 0s |
| **Memory** | 1MB |
| **Description** | Testcase failed. |

**Evaluation logs**

```
Exception in thread "main" java.lang.AssertionError: expected:<10> but was:<11>
at org.junit.Assert.fail(Assert.java:88)
at org.junit.Assert.failNotEquals(Assert.java:834)
at org.junit.Assert.assertEquals(Assert.java:645)
at org.junit.Assert.assertEquals(Assert.java:631)
at eval.main(eval.java:13)
```

**Test_Count** *(weight:1)*

| | |
|---|---|
| **Status** | Passed |
| **Execution time** | 2.64s |
| **CPU** | 0s |
| **Memory** | 1MB |
| **Description** | Testcase passed! |

**Sample_TC** *(sample)*

| | |
|---|---|
| **Status** | Passed |
| **Execution time** | 2.84s |
| **CPU** | 0s |
| **Memory** | 1MB |
| **Description** | Testcase passed! |

## Problem 2 : Shop Online

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 class unless mentioned otherwise.

**Specifications:**

```
class definitions:
 class Customer:
       data fields:
          int  id
          String name
          double walletBalance
          String address
    method definitions:
        Define a parameterized constructor with public visibility

 class Item:
    data fields:
       int id
       String name
       String companyName
       double price
       boolean isInStock
  method definitions:
    Define a parameterized constructor with public visibility

  class ShoppingWebsite:
   method definition:
       purchaseItem(Item i, Customer c) throws ItemOutOfStockException,
InsufficientBalanceException:
          return type: String
          visibility: public

 class InsufficientBalanceException extends Exception:
   method definition:
      InsufficientBalanceException(String message):
         visibility: public

 class ItemOutOfStockException extends Exception:
  method definition:
     ItemOutOfStockException(String message):
        visibility: public
```

Task:

-Implement class **Customer** according to the above specifications

-Implement class **Item** according to the above specifications

-Class **ShoppingWebsite**

**String purchaseItem(Item i, Customer c) throws ItemOutOfStockException, InsufficientBalanceException:**

- Throw an **ItemOutOfStockException** when the item is out of stock with the message "item is out of stock".

- Throw an **InsufficientBalanceException** when customer wallet balance is not sufficient(Item price is greater than the wallet balance) with the message "customer wallet balance is not sufficient"**.**
- If no exception found then return "Order Successful".

-class **InsufficientBalanceException**

- define custom exception class **InsufficientBalanceException** by **extending** the **Exception** class.
- define a parameterized constructor with a String argument to pass the message to the super class.

-class **ItemOutOfStockException**

- define custom exception class **ItemOutOfStockException** by **extending** the **Exception** class.
- define a parameterized constructor with a String argument to pass the message to the super class.

**Sample Testcase**

**Input**

```
Customer cusDet = new Customer(927392, "Steve" ,5000.0, "USA");
Item itemDet = new Item(27392, "T-Shirt", "US polo", 800, true);
ShoppingWebsite obj = new ShoppingWebsite();
String out = obj.purchaseItem(itemDet, cusDet);
```

**output**

```
out = "Order Successful"
```

**NOTE**

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

## Solution

PARTIALLY ACCEPTED | SCORE: **43.8** / 50

## Code Quality Analysis

**Many quality violations**

Quality score: 0.6

## 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

```java
import java.io.*;
import java.util.*;
import java.text.*;
import java.math.*;
import java.util.regex.*;
class Customer {
  // Write code from here..
  int id;
  String name;
  double walletBalance;
  String address;
  public Customer (int id, String name, double walletBalance,String address){
      this.id = id;
      this.name = name;
      this.walletBalance = walletBalance;
      this.address = address;
  }
  public int getId(){
      return id;
  }
  public void setId(int id){
      this.id = id;

  }
  public String getName(){
      return name;
  }
  public void setName(String name){
      this.name = name;
  }
  public double WalletBalance(){
      return walletBalance;
  }
  public void setWalletBalance(double walletBalance){
      this.walletBalance = walletBalance;
  }
  public String getAddress(){
      return address;
  }
  public void setAddress(String address){
      this.address = address;
  }

}
class Item {
  // Write code from here..
  int id;
  String name;
  String companyName;
  double price;
  boolean isInStock;
  public Item(int id,String name,String companyName,double price,boolean isInStock){
      this.id = id;
      this.name = name;
      this.companyName = companyName;
```

```
 56        this.price = price;
 57        this.isInStock = isInStock;
 58    }
 59
 60  }
 61
 62  class ShoppingWebsite {
 63    // Write code from here.
 64    public String purchaseItem(Item i,Customer c) throws
ItemOutOfStockException,InsufficientBalanceException{
 65        if(!i.isInStock){
 66            throw new ItemOutOfStockException("item is out of stock");
 67        }else if(i.price >c.walletBalance){
 68            throw new InsufficientBalanceException("customer wallet balance is not
sufficient");
 69        }else{
 70            c.walletBalance -= i.price;
 71            return "Order Successful";
 72        }
 73        }
 74
 75        }
 76
 77
 78
 79
 80  class InsufficientBalanceException extends Exception {
 81    // Write code from here..
 82    public  InsufficientBalanceException(String message){
 83        super(message);
 84    }
 85  }
 86  class ItemOutOfStockException extends Exception{
 87    // Write code from here..
 88    public ItemOutOfStockException (String message){
 89        super(message);
 90    }
 91
 92  }
 93  public class Source {
 94          public static void main(String args[] ) throws Exception {
 95                  /* Enter your code here. Read input from STDIN. Print output to STDOUT */
 96                  Scanner in = new Scanner(System.in);
 97                  int cid = in.nextInt();
 98                  String cName = in.next();
 99                  double cBalance = in.nextDouble();
100                  String cAddress = in.next();
101                  Customer c = new Customer(cid,cName,cBalance,cAddress);
102
103                  int iid = in.nextInt();
104                  String iname=in.next();
105                  String icompanyName=in.next();
106                  double iprice=in.nextDouble();
107                  boolean iisInStock=in.nextBoolean();
108                  Item i= new Item(iid,iname,icompanyName,iprice,iisInStock);
109
110                  ShoppingWebsite s = new ShoppingWebsite();
111                  try{
112                      String output = s.purchaseItem(i,c);
113                      System.out.println(output);
114
115                  }
116                  catch(ItemOutOfStockException | InsufficientBalanceException e){
117                      System.out.println(e.getMessage());
118                  }
119          }
120  }
```

## Evaluation Details

**ValidData_TC** *(weight:1)*

| | |
|---|---|
| **Status** | Passed |
| **Execution time** | 2.52s |
| **CPU** | 0s |
| **Memory** | 1MB |
| **Description** | Testcase passed! |

**InvalidData_ItemOutOfStockException** *(weight:1)*

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

**InvalidData_InsufficientBalanceException** *(weight:1)*

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

**Sample Testcase** *(sample)*

| | |
|---|---|
| **Status** | Passed |
| **Execution time** | 2.79s |
| **CPU** | 0s |
| **Memory** | 1MB |
| **Description** | Testcase passed! |

**ShoppingWebsite_TC** *(weight:1)*

| | |
|---|---|
| **Status** | Passed |
| **Execution time** | 2.44s |

| | |
|---|---|
| **CPU** | 0s |
| **Memory** | 1MB |
| **Description** | Testcase passed! |

**InsufficientBalanceException_TC** *(weight:1)*

| | |
|---|---|
| **Status** | Passed |
| **Execution time** | 3.02s |
| **CPU** | 0s |
| **Memory** | 1MB |
| **Description** | Testcase passed! |

**Customer_TC** *(weight:1)*

| | |
|---|---|
| **Status** | Failed |
| **Execution time** | 2.50s |
| **CPU** | 0s |
| **Memory** | 1MB |
| **Description** | Testcase failed. |

**Evaluation logs**

```
Exception in thread "main" java.lang.AssertionError: expected:<17> but was:<9>
at org.junit.Assert.fail(Assert.java:88)
at org.junit.Assert.failNotEquals(Assert.java:834)
at org.junit.Assert.assertEquals(Assert.java:645)
at org.junit.Assert.assertEquals(Assert.java:631)
at eval.main(eval.java:15)
```

**Item_TC** *(weight:1)*

| | |
|---|---|
| **Status** | Passed |
| **Execution time** | 2.68s |
| **CPU** | 0s |
| **Memory** | 1MB |
| **Description** | Testcase passed! |

**ItemOutOfStockException_TC** *(weight:1)*

| | |
|---|---|
| **Status** | Passed |

| | |
|---|---|
| **Execution time** | 2.77s |
| **CPU** | 0s |
| **Memory** | 1MB |
| **Description** | Testcase passed! |

### Problem 3 : The classroom

CODING | SCORE: **50**

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: **44.4** / 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.

### 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 Student(String name, int score){
12     this.name=name;
13     this.score=score;
14 }
15 }
16
17 class Classroom{
18 //Code Here..
19 public String registerStudent(Student student){
20     if(!student.name.equals(student.name.toUpperCase())){
```

```
21          return "Block letters needed";
22
23      }
24      if(student.score <0 || student.score >100){
25          return "Invalid Score";
26
27      }
28      return "Registered";
29  }
30  public String studentCard(String card){
31      if(!card.matches("[0-9]+")){
32          return "Invalid card";
33      }
34      return "Valid card";
35  }
36  }
37
38  public class Source {
39          public static void main(String args[] ) throws Exception {
40                  /* Enter your code here. Read input from STDIN. Print output to STDOUT */
41          BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
42           String[] studentDetails = br.readLine().trim().split(" ");
43           Student student = new Student(studentDetails[0],
Integer.parseInt(studentDetails[1]));
44           Classroom classroom = new Classroom();
45           System.out.println (classroom.registerStudent(student));
46          }
47  }
```

## Evaluation Details

**Testcase #clsClassroom** *(weight:1)*

| | |
|---|---|
| **Status** | Passed |
| **Execution time** | 2.56s |
| **CPU** | 0s |
| **Memory** | 1MB |
| **Description** | Testcase passed! |

**Testcase #studentCard2** *(weight:1)*

| | |
|---|---|
| **Status** | Passed |
| **Execution time** | 2.68s |
| **CPU** | 0s |
| **Memory** | 1MB |
| **Description** | Testcase passed! |

**Testcase #Sample** *(sample)*

| | |
|---|---|
| **Status** | Passed |
| **Execution time** | 2.51s |
| **CPU** | 0s |

| | |
|---|---|
| **Memory** | 1MB |
| **Description** | Testcase passed! |

**Testcase #clsStudent** *(weight:1)*

| | |
|---|---|
| **Status** | Passed |
| **Execution time** | 2.48s |
| **CPU** | 0s |
| **Memory** | 1MB |
| **Description** | Testcase passed! |

**Testcase #studentCard3** *(weight:1)*

| | |
|---|---|
| **Status** | Passed |
| **Execution time** | 2.53s |
| **CPU** | 0s |
| **Memory** | 1MB |
| **Description** | Testcase passed! |

**Testcase #registerStudent1** *(weight:1)*

| | |
|---|---|
| **Status** | Passed |
| **Execution time** | 2.79s |
| **CPU** | 0s |
| **Memory** | 1MB |
| **Description** | Testcase passed! |

**Testcase #registerStudent4** *(weight:1)*

| | |
|---|---|
| **Status** | Passed |
| **Execution time** | 2.64s |
| **CPU** | 0s |
| **Memory** | 1MB |
| **Description** | Testcase passed! |

**Testcase #registerStudent2** *(weight:1)*

| | |
|---|---|
| **Status** | Passed |
| **Execution time** | 2.62s |
| **CPU** | 0s |
| **Memory** | 1MB |
| **Description** | Testcase passed! |

**Testcase #registerStudent3** *(weight:1)*

| | |
|---|---|
| **Status** | Failed |
| **Execution time** | 2.73s |
| **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** | Passed |
| **Execution time** | 2.57s |
| **CPU** | 0s |
| **Memory** | 1MB |
| **Description** | Testcase passed! |