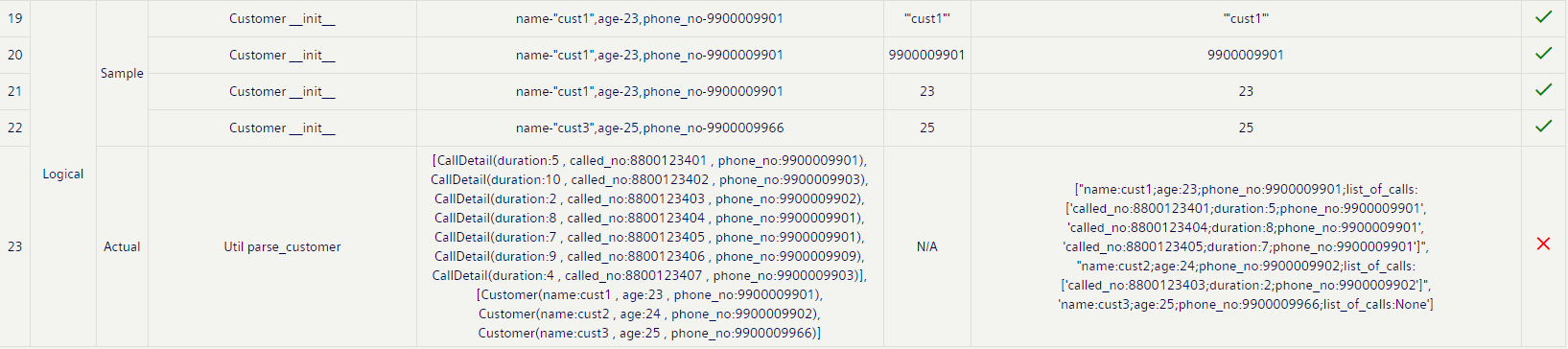
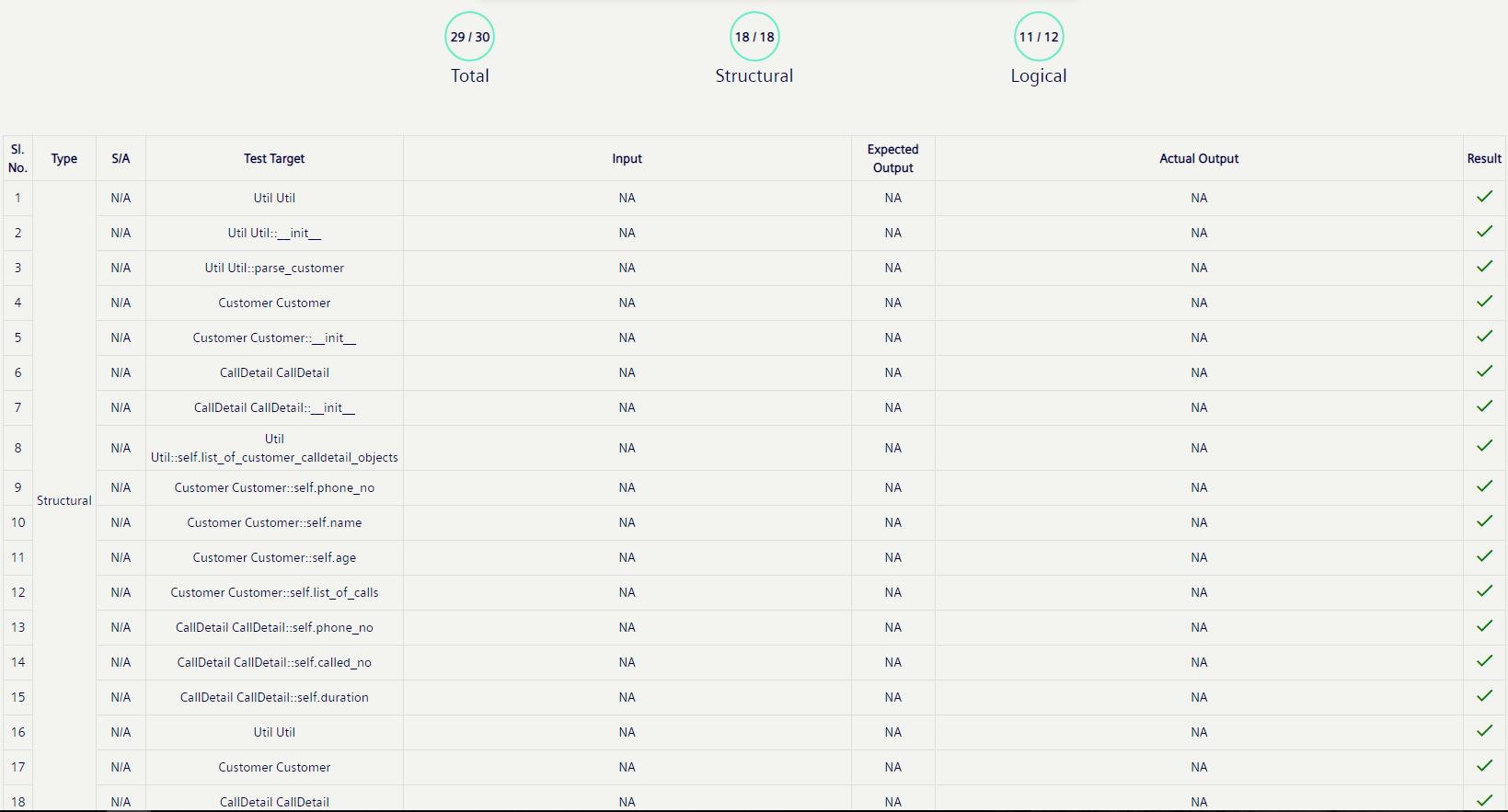
# Assignment Set 3

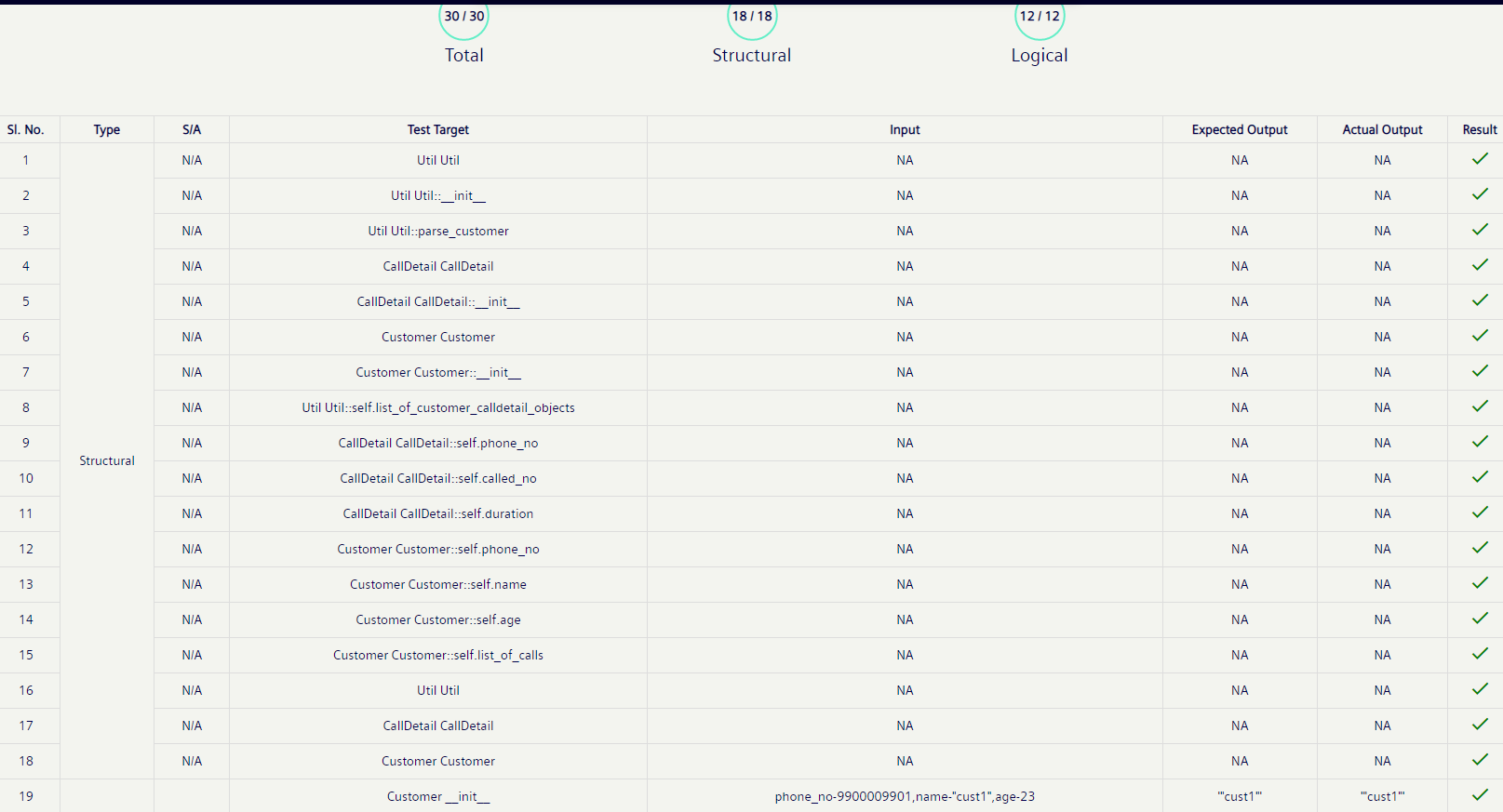
## Assignment on Dependency - Level 2

A telecom company wants to generate reports on the call details of the customers. Each customer can make multiple phone calls.  
  
**Problem Statement:**The parse\_customer method takes a list of Customer objects and a list of CallDetail objects. For every customer, identify all the corresponding Call Detail objects ( the customer phone number and the phone number of Call detail object have to match ), add them to the list\_of\_calls of corresponding customer object and add the updated customer object to list\_of\_customer\_calldetail\_objects of Util class.

1Fail: \**was failing due to my ERROR CHECK - if(len(temp)!=0:*

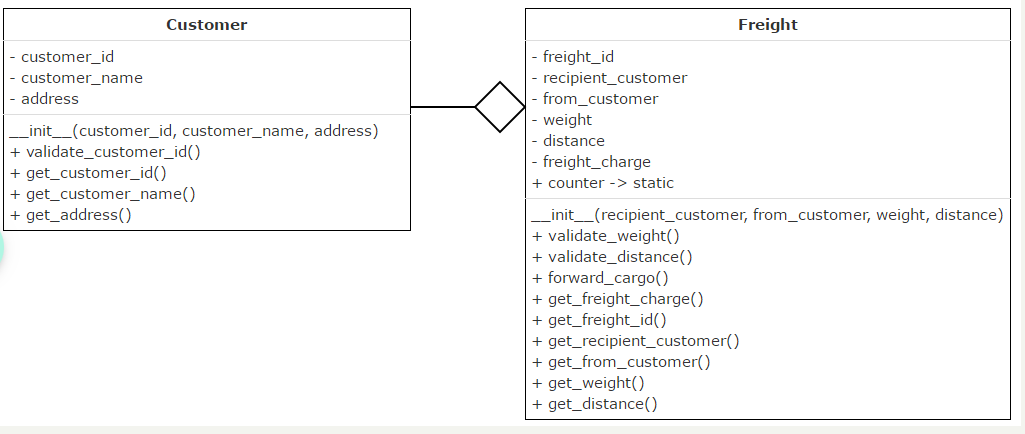


All Pass:



## Assignment on Aggregation - Level 2

Freight Pvt. Ltd, a cargo company, forwards cargos/freights between its customers.  
Freight charges are applied based on weight and distance of the shipment.  
Write a python program to implement the class diagram given below.



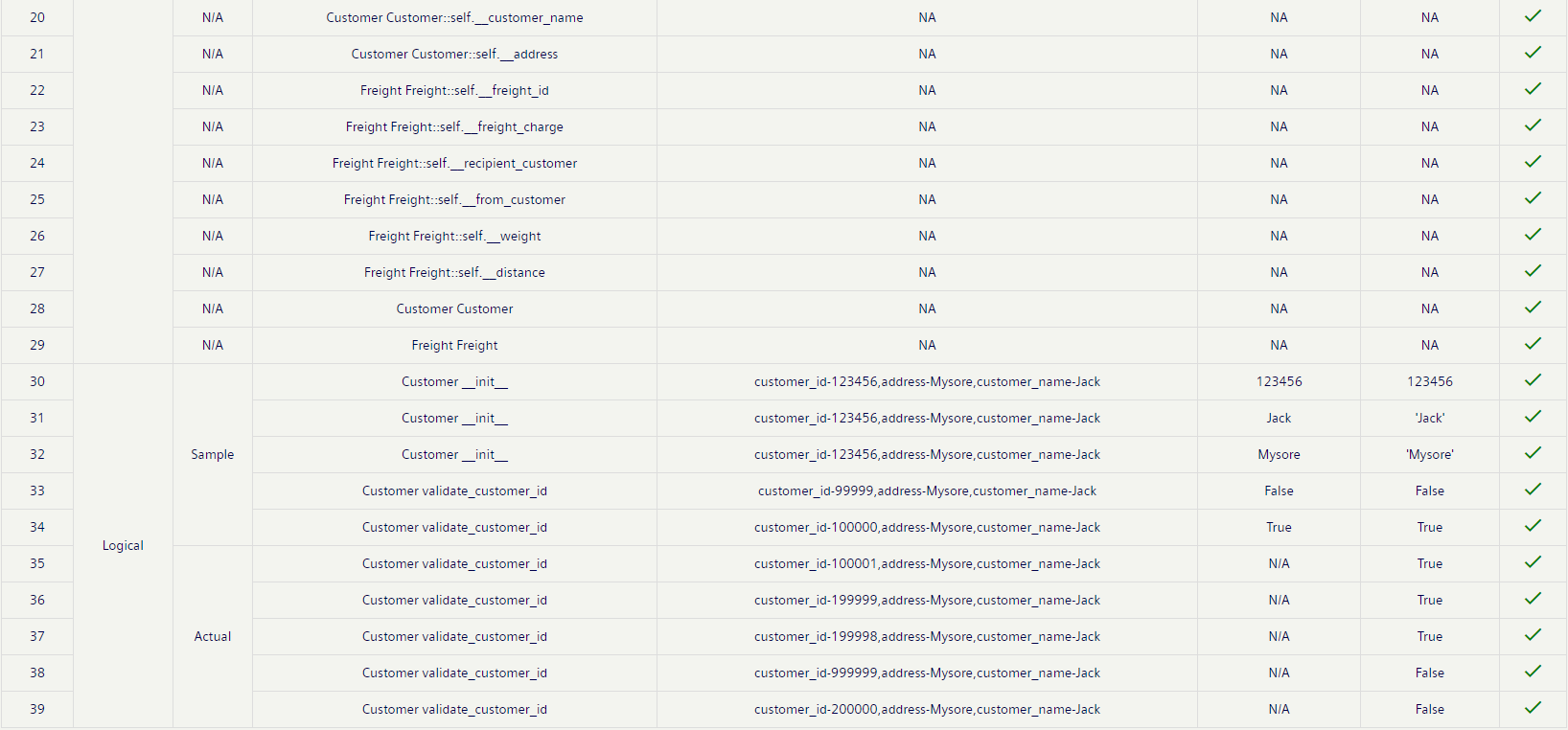
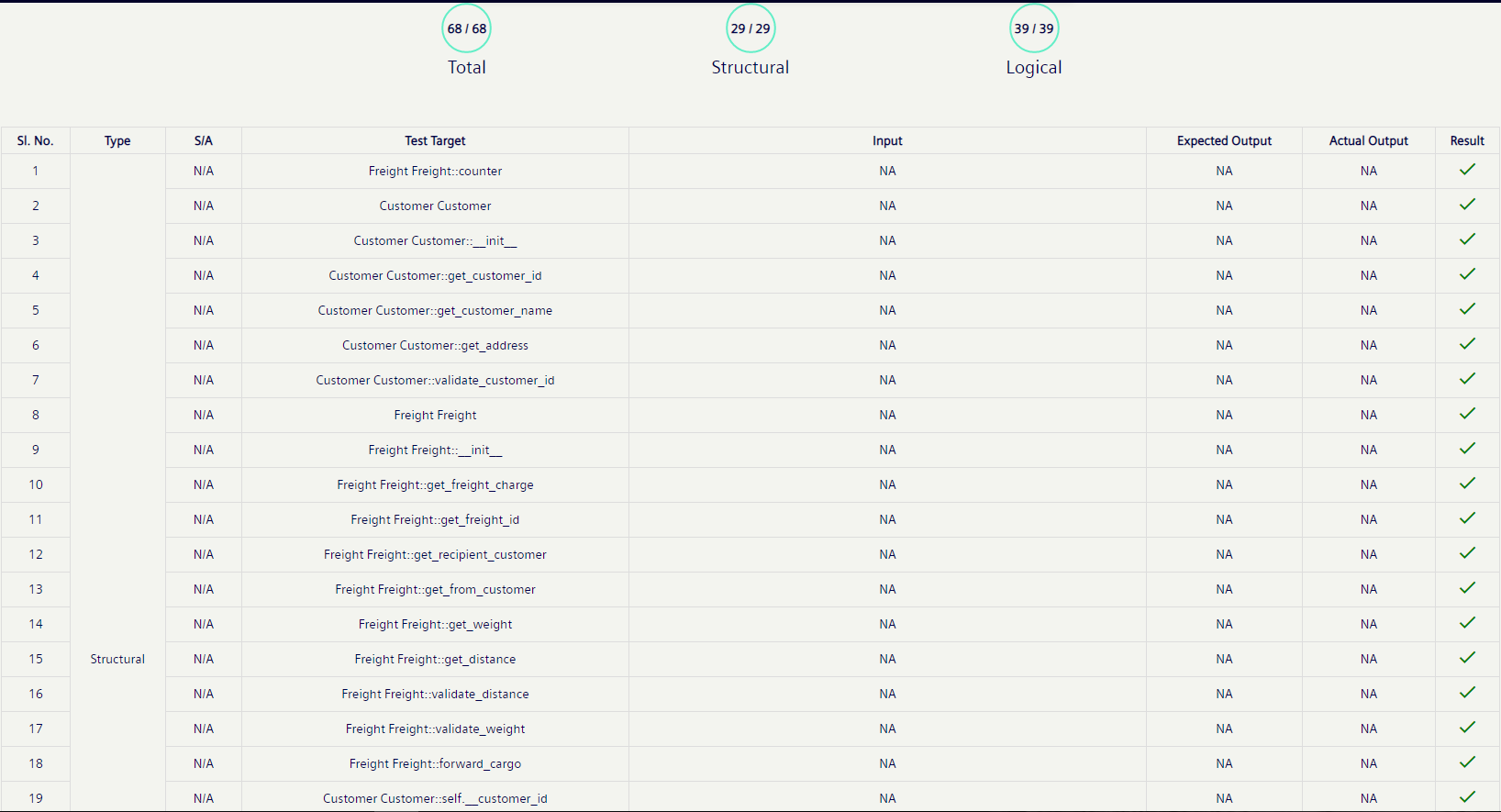
**Method description:**

1. Initialize counter variable to 198 in Freight class
2. All validate methods should return true, if validation succeeds. Else it should return false
3. **validate\_customer\_id():** Customer id should be 6 digits and should begin with digit 1
4. **validate\_weight():** Weight should be a multiple of 5
5. **validate\_distance():**Distance should be between 500kms and 5000kms (both inclusive)
6. **forward\_cargo():**
   * Validate from\_customer.customer\_id, recipient\_customer.customer\_id, distance and weight of the freight
   * If valid,
     + auto-generate freight\_id starting from 200 and initialize it. freight\_id should be even
     + calculate freight\_charge based on weight (Rs.150/kg) and distance (Rs.60/km)
   * Else, set freight\_charge to 0

**For testing:**

* Create objects of Customer and Freight class
* Invoke forward\_cargo() method on Freight object
* Display freight id and freight charge
* In case of error/invalid data, display appropriate error message

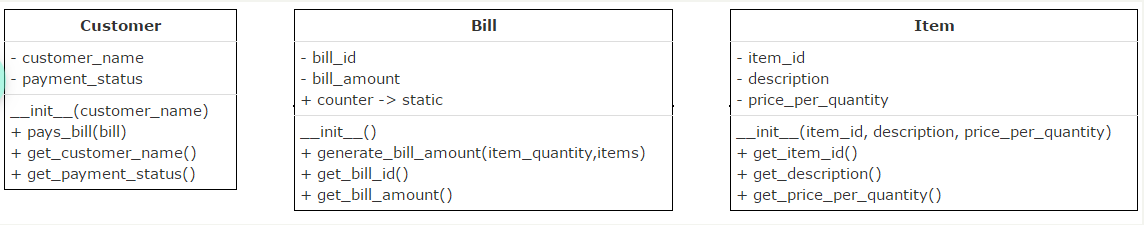
All Pass:



## Assignment on Dependency & List Objects - Level 2

In the retail store scenario, let's look at the portion of customer purchasing items from the retail store.

Write a python program to implement the class diagram given below.



**Class Description:   
Bill class:**

1. Initialize static variable counter to 1000
2. **generate\_bill\_amount(item\_quantity,items):**Calculate bill amount based on the items purchased by the customer
   1. Accept a dictionary, item\_quantity which contains the item id (key) of the items purchased along with the quantity (value)
   2. Accept a list, items which contains the list of Item objects available in the store
   3. Generate bill id starting from 1001 prefixed by "B" and initialize attribute, bill\_id. Ex. "B1001", "B1002" etc.
   4. Calculate bill amount based on the quantity and price of the items purchased by the customer
   5. Set attribute, bill\_amount with the calculated bill amount

Assume that values in item\_quantity and items are always valid.   
**Customer class:**

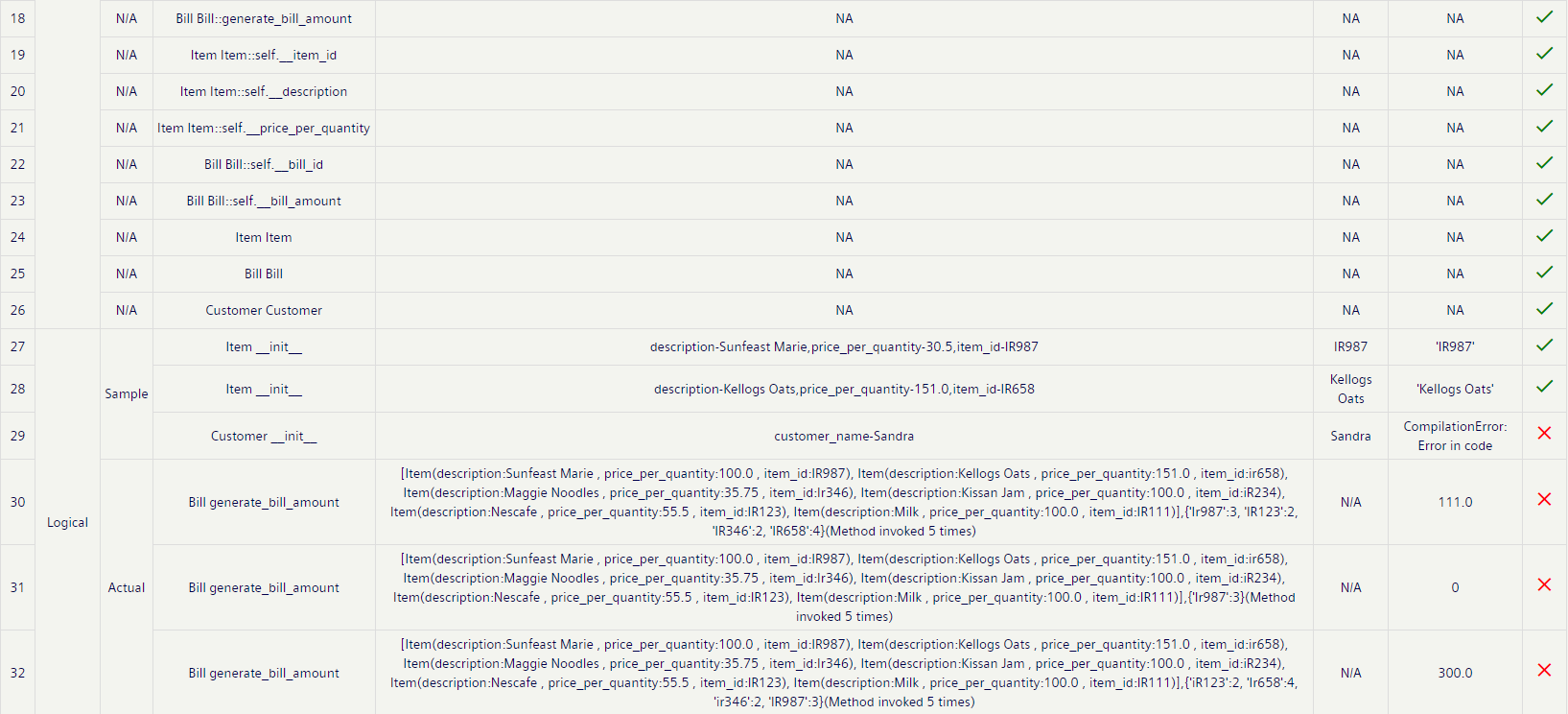
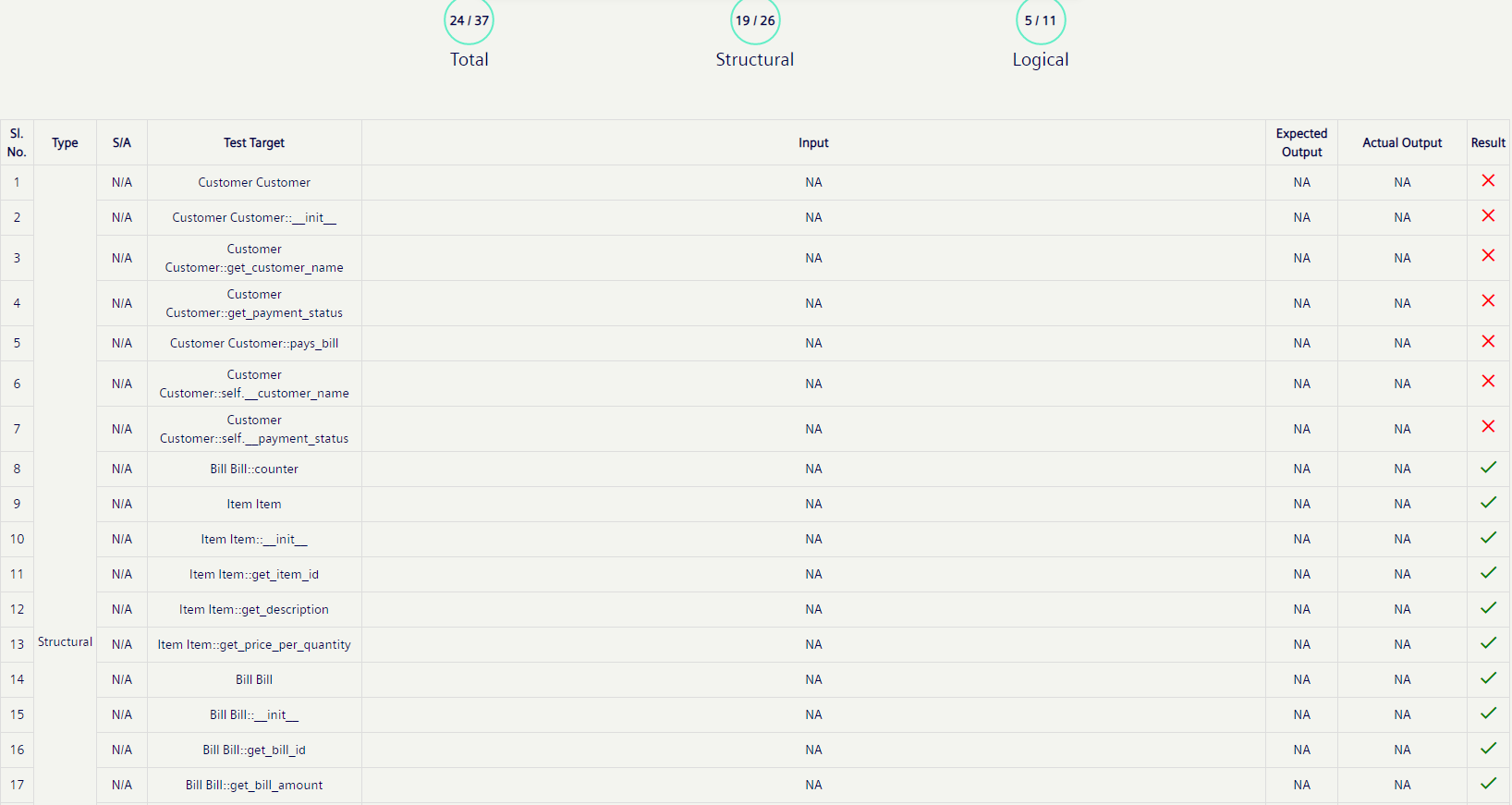
**pays\_bill(bill):**Pay bill based on the bill amount

1. Accept Bill object which contains the details of the bill to be paid by the customer
2. Update attribute, payment\_status to "Paid"
3. Display customer name, bill id and bill amount

**Note:** Perform case insensitive string comparison  
  
**For testing:**

* Create objects of Customer class, Item class and Bill class
* Invoke generate\_bill\_amount(item\_quantity,items) on Bill object by passing the dictionary containing item\_id and quantity of items purchased by the Customer and list of Item objects
* Invoke pays\_bill() on Customer object by passing the Bill object

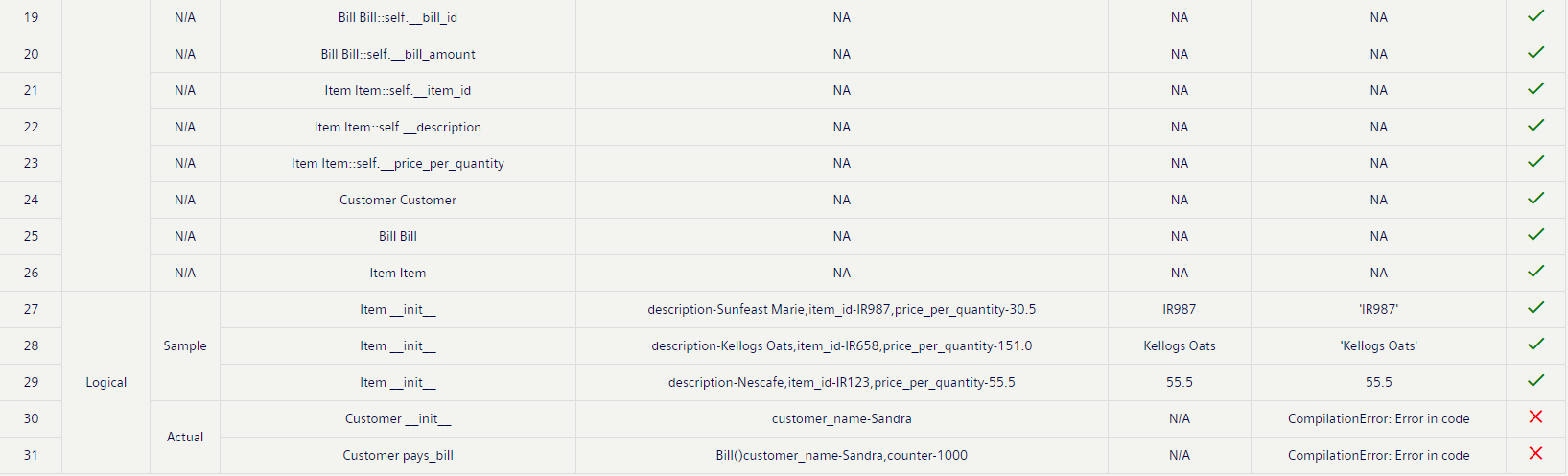
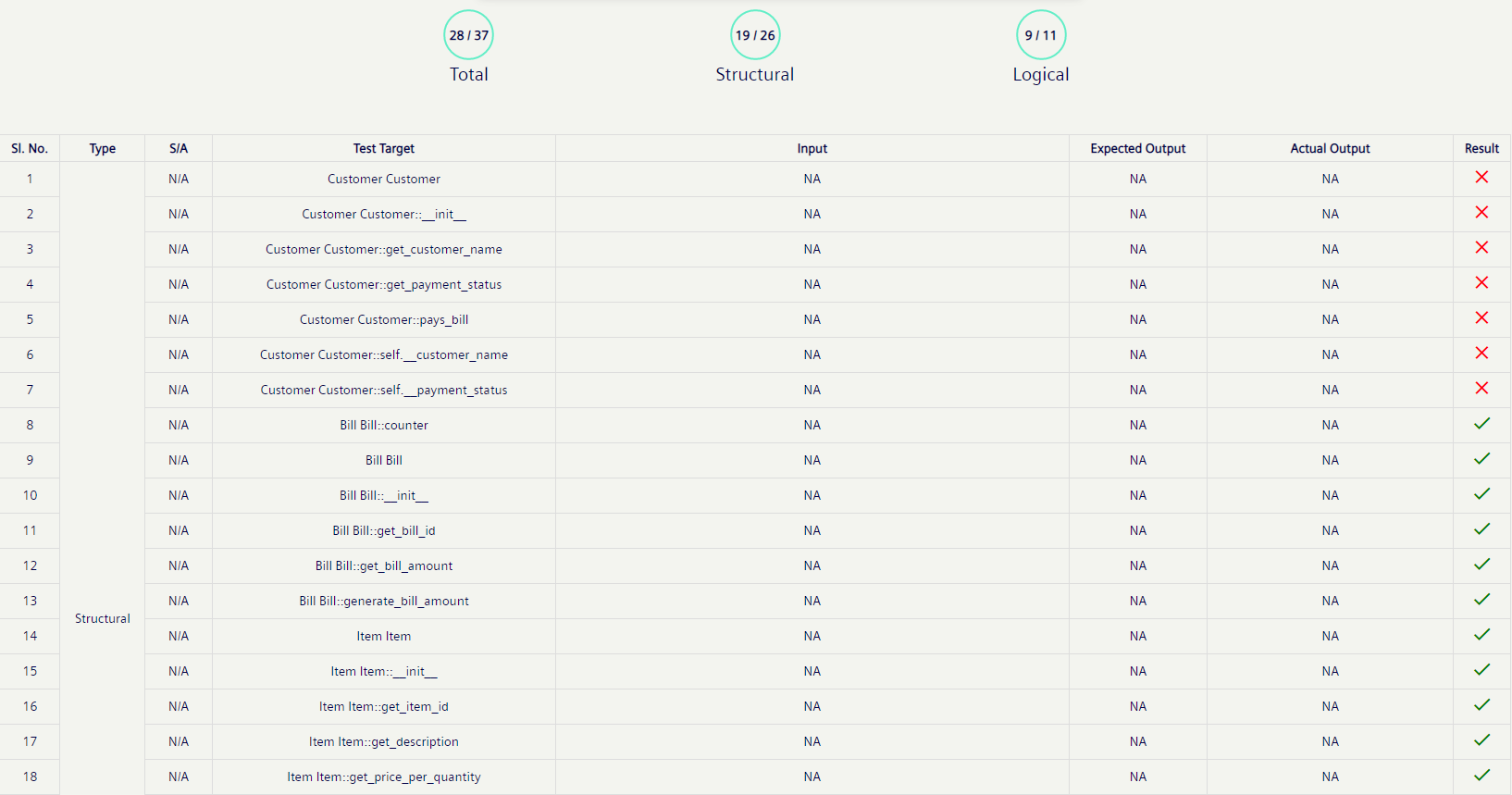
Initial Verification:



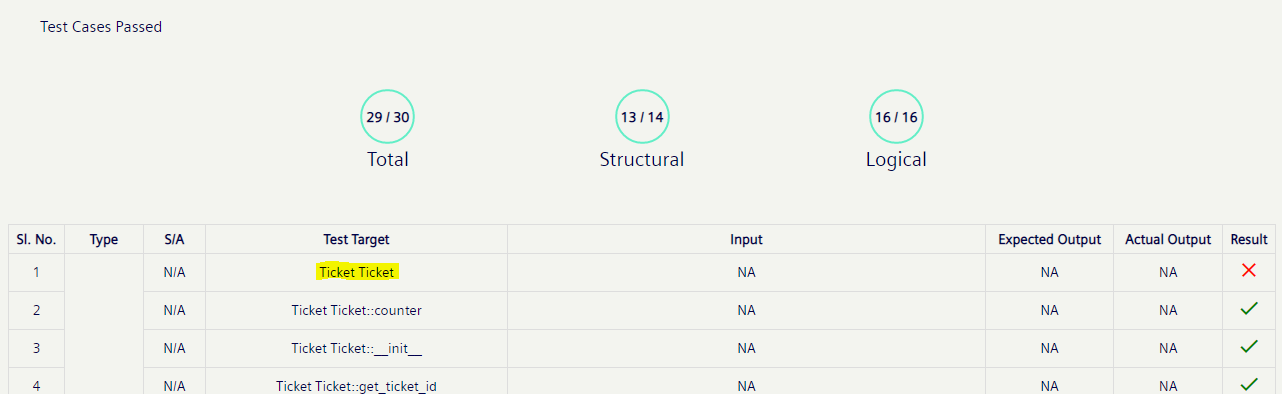
Debugging:

* Bill generate\_bill\_amount test case passed after adding “if(key.upper()==item.get\_item\_id().upper()):” since ***case insensitive*** string comparison

Still Below test cases failing:



## Assignment on Static Counter - Level 2 [Repeated]

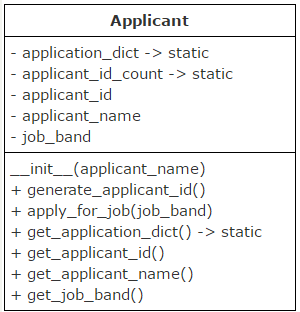


*What is the above Testcase checking for?*

## Assignment on Static - Level 2

"Infinity" IT solution wants to automate their recruitment process. They have decided to accept 5 applications for each of the three job bands ("A", "B" and "C") in the company.

Write a Python program to implement the class diagram given below.



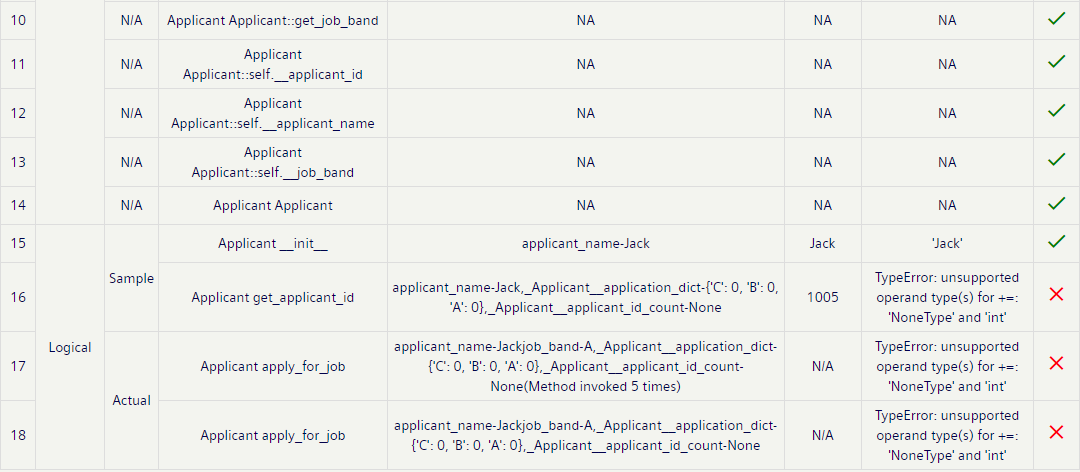
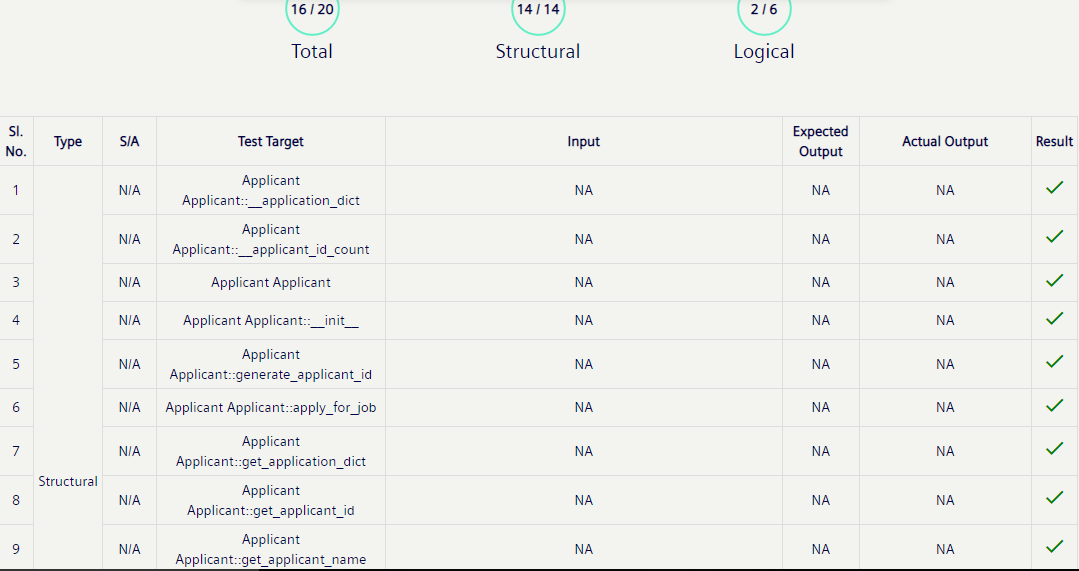
**Method/Attribute description:**

1. Initialize static variable, applicant\_id\_count to 1000
2. **application\_dict:** Dictionary which store application count (value) for each job band (key)
3. **generate\_applicant\_id():** Auto-generate applicant id starting from 1001 and initialize attribute, applicant\_id
4. **apply\_for\_job(job\_band):** Accept the job band for which the applicant is applying.
   1. Check if application count for the applied job band has reached the maximum limit, 5. If so, return -1.
   2. Else,
      * increment application count for the applied job band by 1 in the dictionary
      * generate applicant id and
      * initialize attribute, job\_band with the applied job\_band

**For testing:**

* Create objects of Applicant class
* Invoke apply\_for\_job(job\_band) method on Applicant object by passing the job band for which applicant is applying
* If application is accepted, display applicant id, name and job band
* Else, display appropriate error message

Verification:

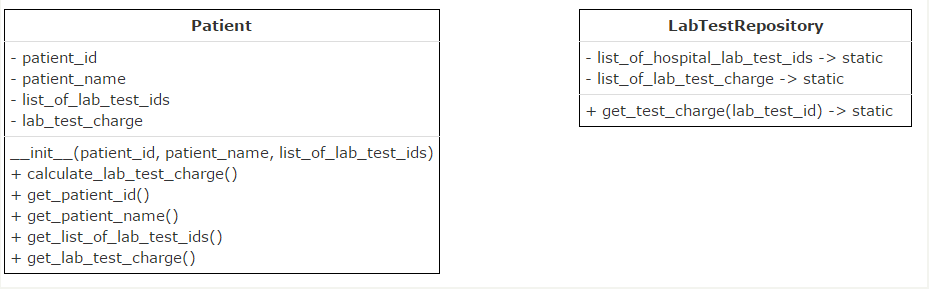


\*\*\* All test cases were failing earlier due to the print statement in else: of the method – “def **apply\_for\_job**(*self*,job\_band):”

Still some test cases ; check later.

## Assignment on Dependency & Static - Level 2

Care hospital management wants to calculate the charge of lab tests done by its patients.  
Write a python program to implement the class diagram given below.



**Class Description:  
LabTestRepository class:**

1. **list\_of\_hospital\_lab\_test\_ids:** Static list which contains the list of test ids of lab tests available in the hospital
2. **list\_of\_lab\_test\_charge:** Static list which contains the charge of the lab tests available in the hospital
3. The above two lists have one-to-one correspondence
4. **get\_test\_charge(lab\_test\_id):** Accept a lab test id and return the corresponding lab test charge. If lab test id is invalid, return -1

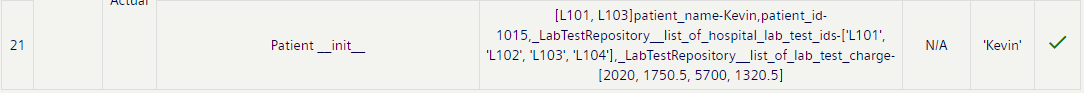
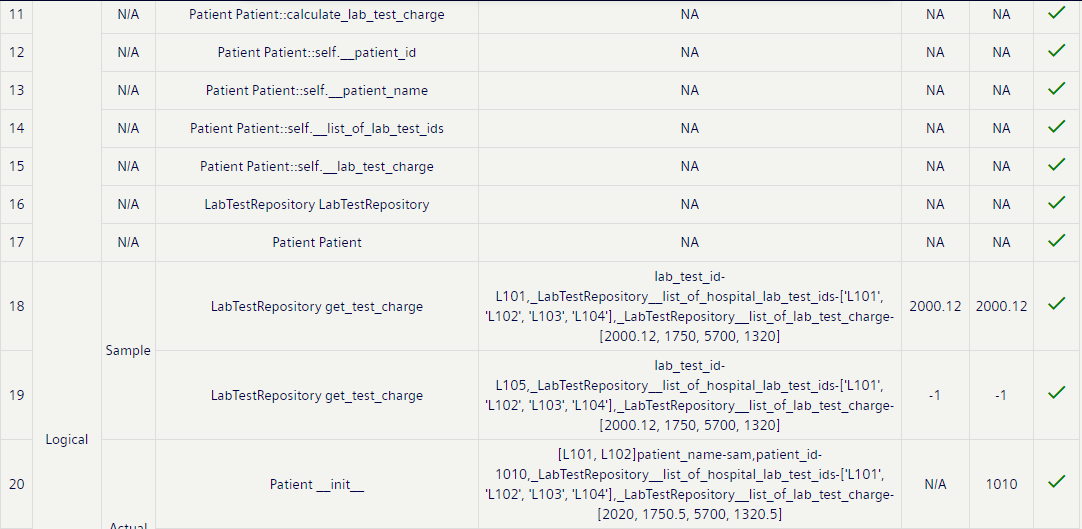
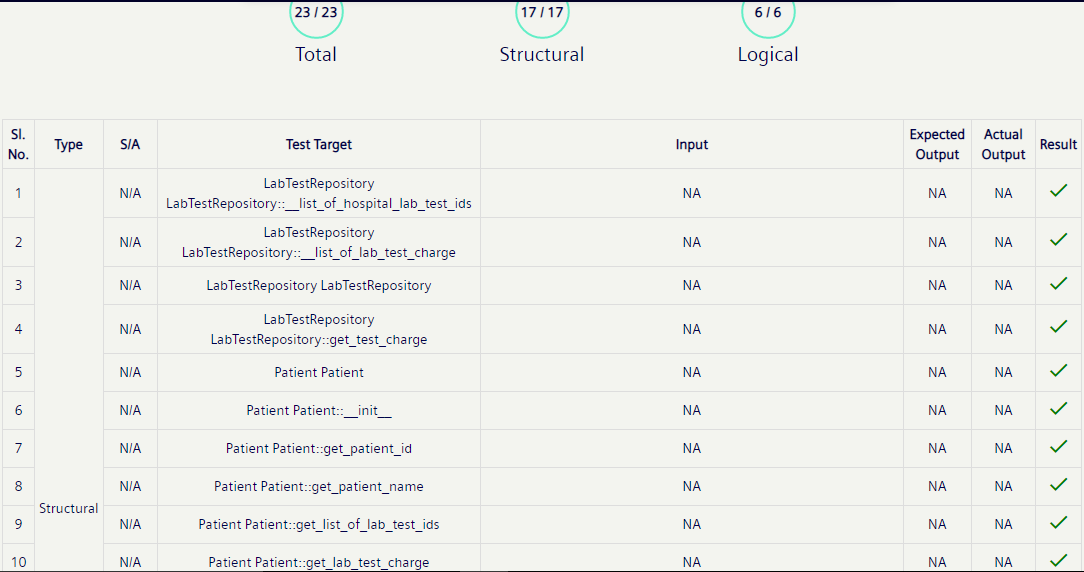
**Patient class:**

1. **list\_of\_lab\_test\_ids:** Instance variable which contains the list of test ids of lab tests done by the patient
2. **calculate\_lab\_test\_charge():** Calculate total charge of the lab tests done by the patient
   1. Calculate total lab test charge based on test charge of each lab test done by the patient
   2. If any lab test id provided by the patient is invalid, consider its charge to be 0
   3. Initialize attribute, lab\_test\_charge with the total lab test charge

**Note:**Perform case sensitive string comparison    
  
**For testing:**

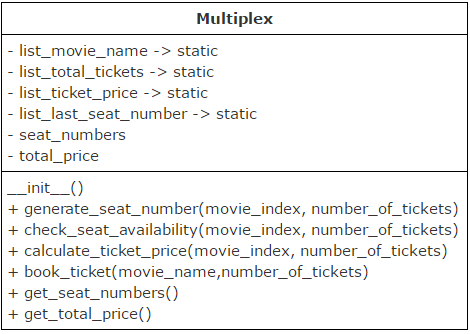
* Create objects of Patient class
* Invoke calculate\_lab\_test\_charge() on Patient object
* Display patient name, patient id, test ids of lab tests done by the patient and total lab test charge

Verification: [All Pass]



## Assignment on Static List - Level 3

"FairyLand Multiplex" wants to automate ticket booking and seat allocation process.  
Write a python program to implement the class diagram given below.



**Assumptions:**

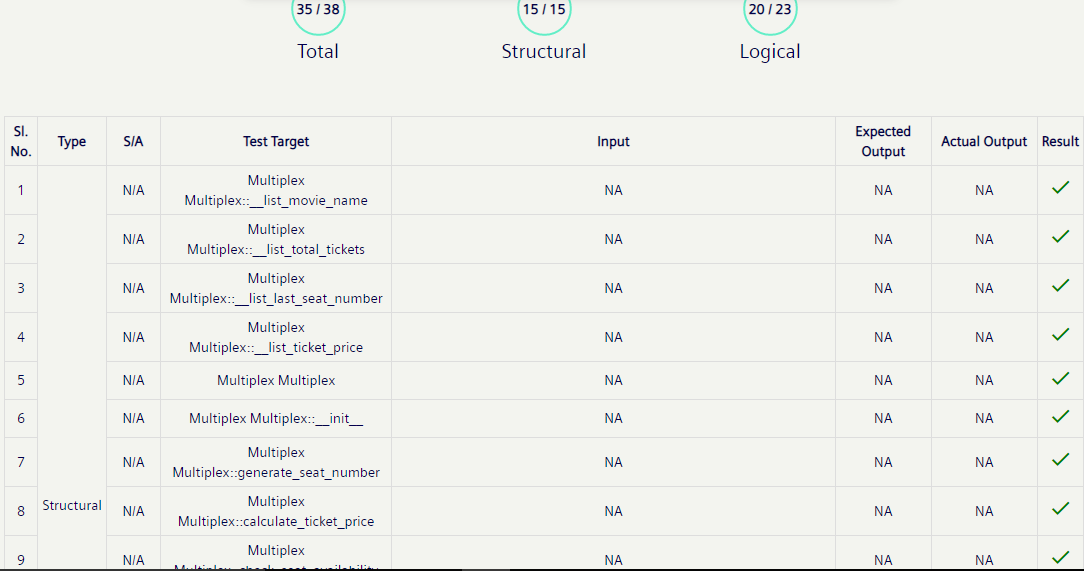
1. Multiplex has two screens having different seating capacity
2. Two movies will be screened everyday (one show/movie)
3. Booking will be opened every day morning for that day’s shows
4. Movie name, total tickets available, ticket price and last seat number allocated for both movies are stored in lists having one to one correspondence. Details of first movie will be available at the 0th index and second movie at the 1st index of these lists

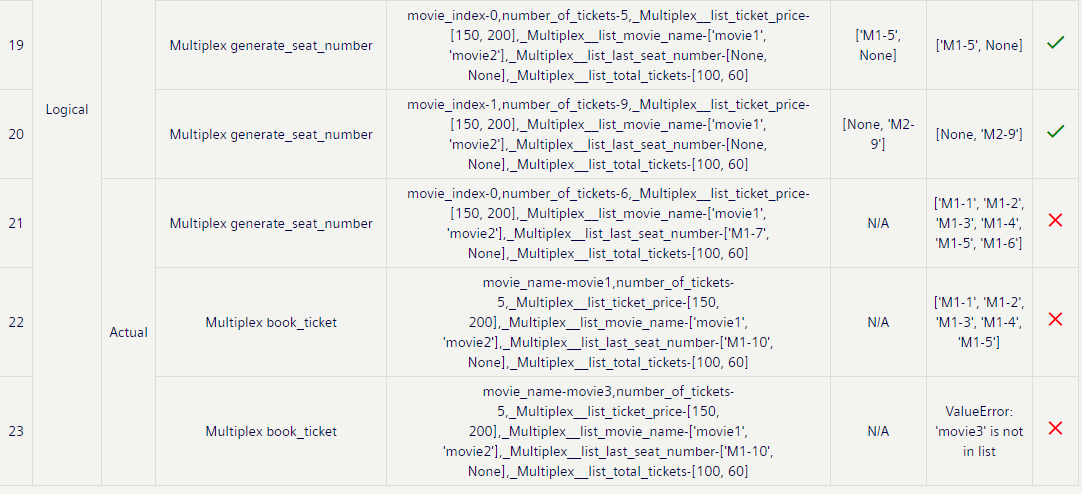
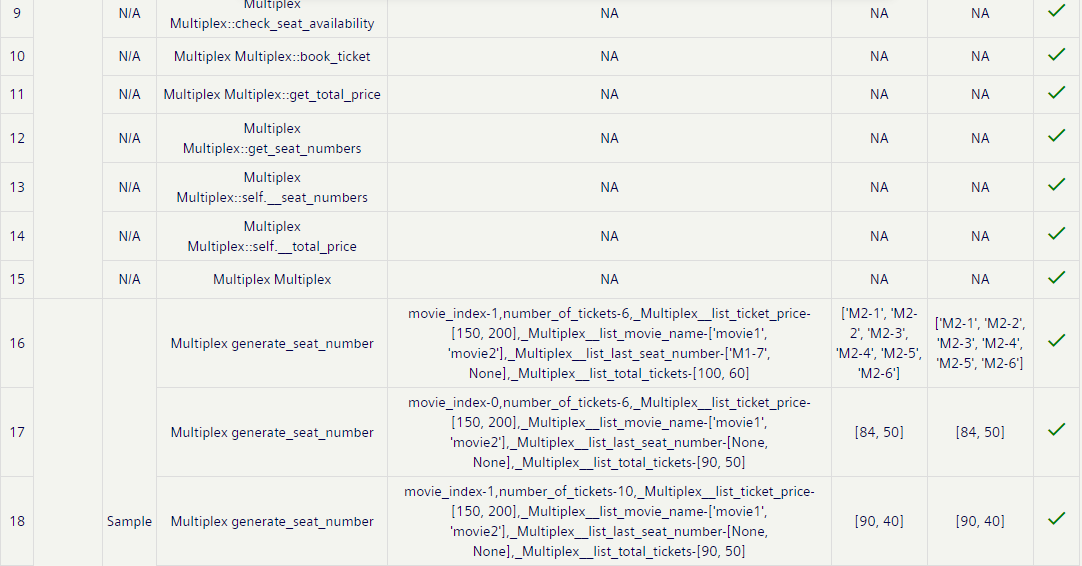
**Method description:**

1. **check\_seat\_availability(movie\_index,number\_of\_tickets):** Checks seat availability for the given movie. Refer the code given in starter code
2. **calculate\_ticket\_price(movie\_index,number\_of\_tickets):** Calculates total ticket price for the given movie. Refer the code given in starter code
3. **generate\_seat\_number(movie\_index,number\_of\_tickets):**Allocate required number of seats for the given movie.
   1. Seat numbers should be auto-generated as mentioned below:
      1. Seat numbers should be generated starting from 1, prefixed by "M1-" for movie-1 and "M2-" for movie 2
      2. Examples movie-1: M1-1, M1-2, M1-3 etc, movie-2: M2-1,M2-2 etc
4. Update total number of tickets available for the given movie in list\_total\_tickets
5. Update last seat number allocated for the given movie in list\_last\_seat\_number
6. Return the list of generated seat numbers
7. **book\_ticket(movie\_name,number\_of\_tickets):** Book tickets for the given movie.
8. Return 0, if movie name is invalid
9. Return -1, if enough tickets are not available for the given movie
10. Else,
    1. Generate seat numbers
    2. Initialize attribute, seat\_numbers with the list of generated seat numbers
    3. Calculate total ticket price

Perform case sensitive string comparison.

Verification: [3 test cases failed]





Debugging:

* Removing **movie\_index=Multiplex.\_\_list\_movie\_name.index(movie\_name)** passed the last test case-23 as it was Throwing *ValueError: 'movie3' is not in list during runtime* & also bypassing the movie validation if statement.
* Modified code in def **generate\_seat\_number**(*self*,movie\_index, number\_of\_tickets): to generate tickets from 0 as well as from the last seat number. This cleared 2 more test cases.
* Last test case (i.e, test case-22) was failing as I didn’t account for the “last seat number” to be anything other than a single digit number.

**\*\*All passed after the above debugging.**

# Inheritance

## Exercise on Inheritance - Level 1

The circus has hired two riders – a bike rider and a cycle rider. Both of them ride vehicles but bike rider rides the bike in a dome whereas the cycle rider rides the cycle blind folded. The circus manger has also ensured that both of them are trained and have enough experience of performing these stunts in circus. Apart from this, bike rider also has a race license.

Create the class diagram for representing the above scenario by choosing the class names, attributes, methods and relationships from the list given below.

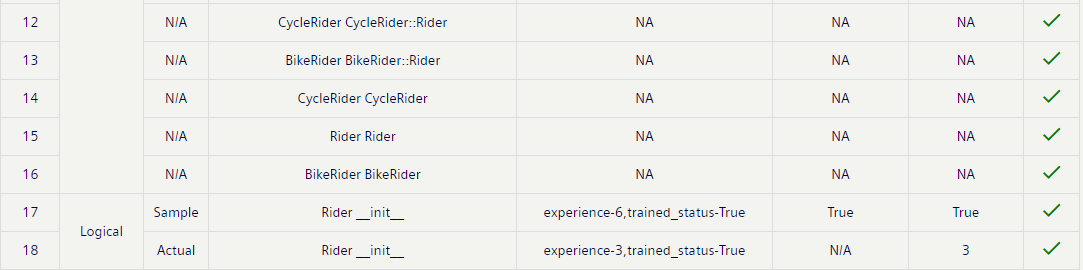
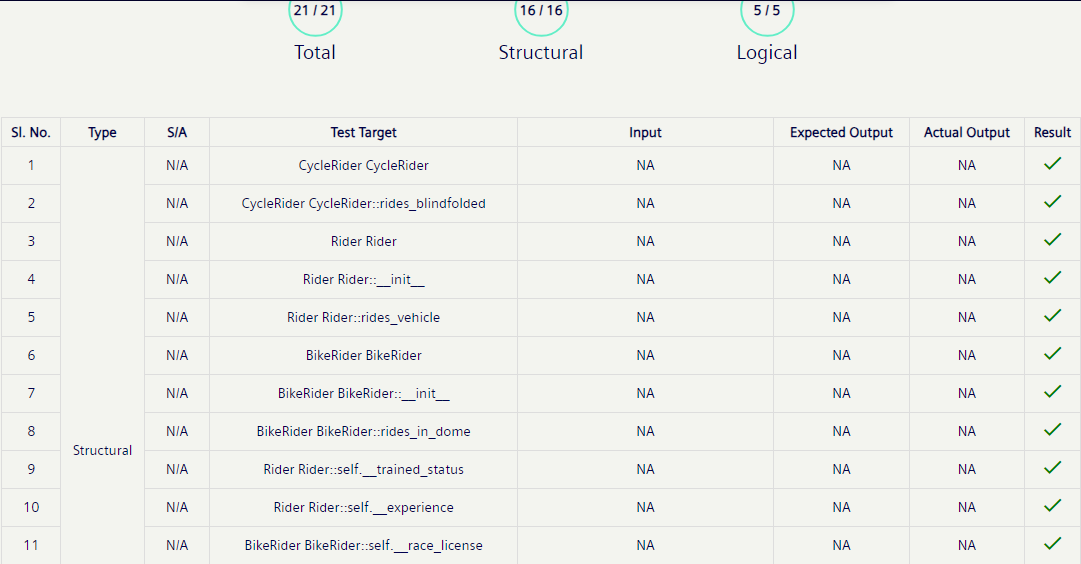
* experience
* performs\_tricks()
* \_\_init\_\_(trained\_status,experience,race\_license)
* rider
* rides\_blindfolded()
* trained\_status
* \_\_init\_\_(trained\_status,experience)
* rides\_vehicle()
* bikeRider
* cycleRider
* race\_license
* rides\_in\_dome()

Assume that none of the instance variables can be accessed outside the class whereas methods can be accessed.

Write a python program to implement the created class diagram. Represent bike rider and cycle rider, make them ride the respective vehicles.

**Note:** rides\_vehicle(), rides\_in\_dome(), rides\_blindfolded() methods should display appropriate messages. Assume that trained\_status and race\_license are boolean variables and experience is an integer.

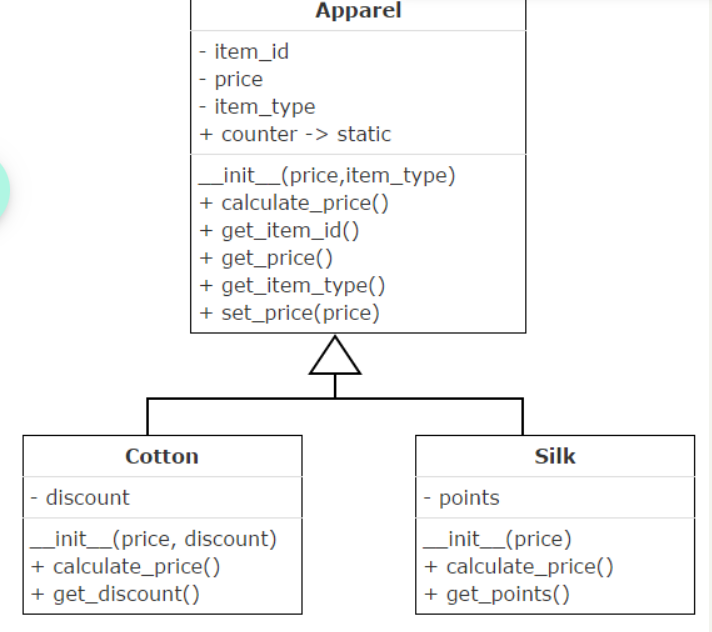
**Verification:** [Initially 1 test case failed as i used def \_\_init\_\_(trained\_status,experience,race\_license) in BikeRider Class instead of inherting it from **Rider Class** by using super().\_\_init\_\_(trained\_status, experience)



# Assignment Set 4

## Assignment on Inheritance - Level 2

An apparel shop wants to manage the items which it sells.   
Write a python program to implement the class diagram given below.



**Class Description:  
Apparel class:**

1. Initialize static variable counter to 100
2. In the constructor, auto-generate item\_id starting from 101 prefixed by "C" for cotton apparels and "S" for silk apparels. Example – C101, S102, S103, C104 etc.
3. **calculate\_price():** Add 5% service tax on the price of the apparel and update attribute, price with the new value

**Cotton class:**

1. While invoking parent constructor from child constructor, pass "Cotton" as item\_type
2. **calculate\_price():**Update attribute, price of Apparel class based on rules given below
   1. Add service tax on price by invoking appropriate method of Apparel class
   2. Apply discount on price
   3. Add 5% VAT on final price

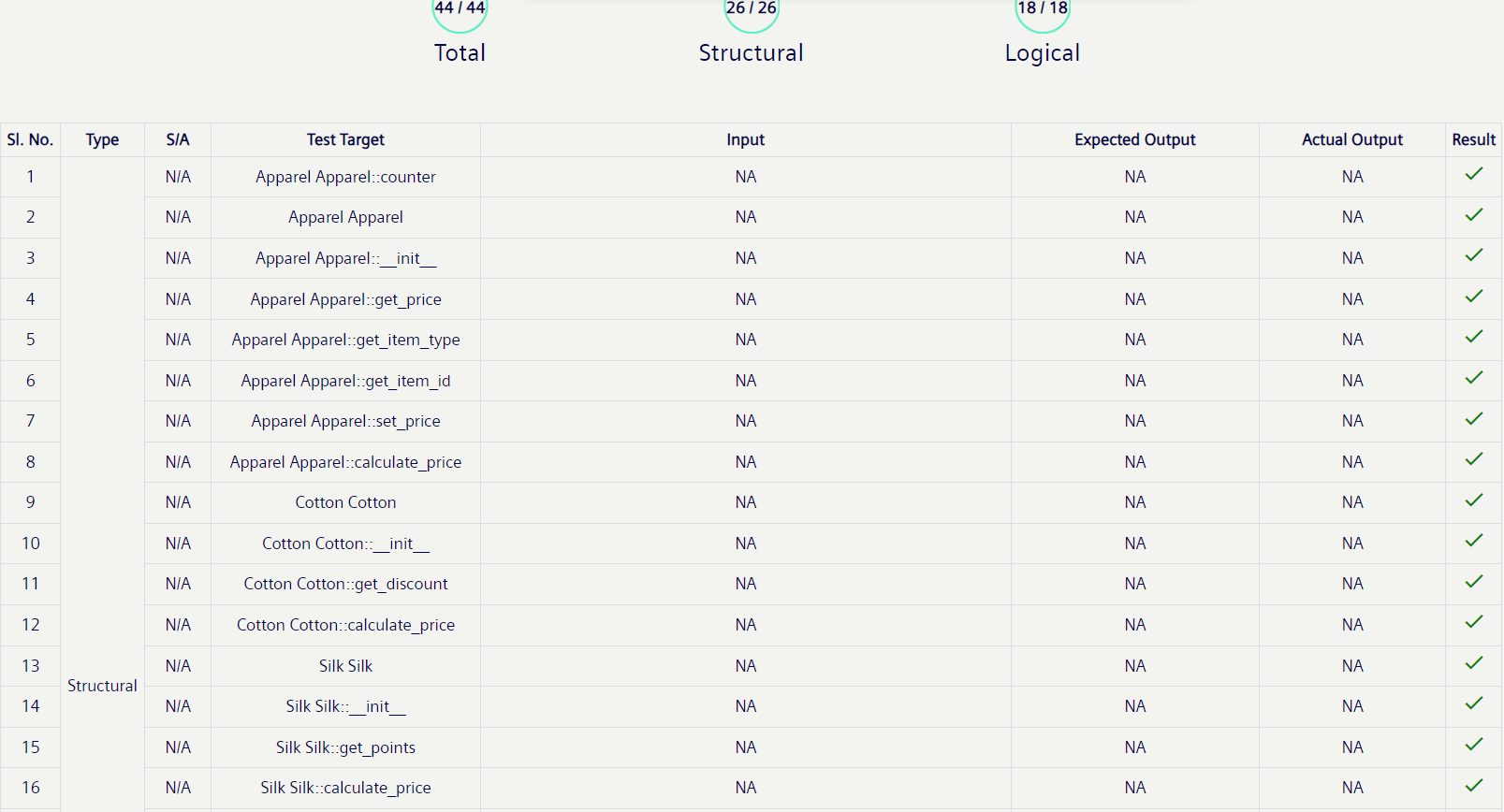
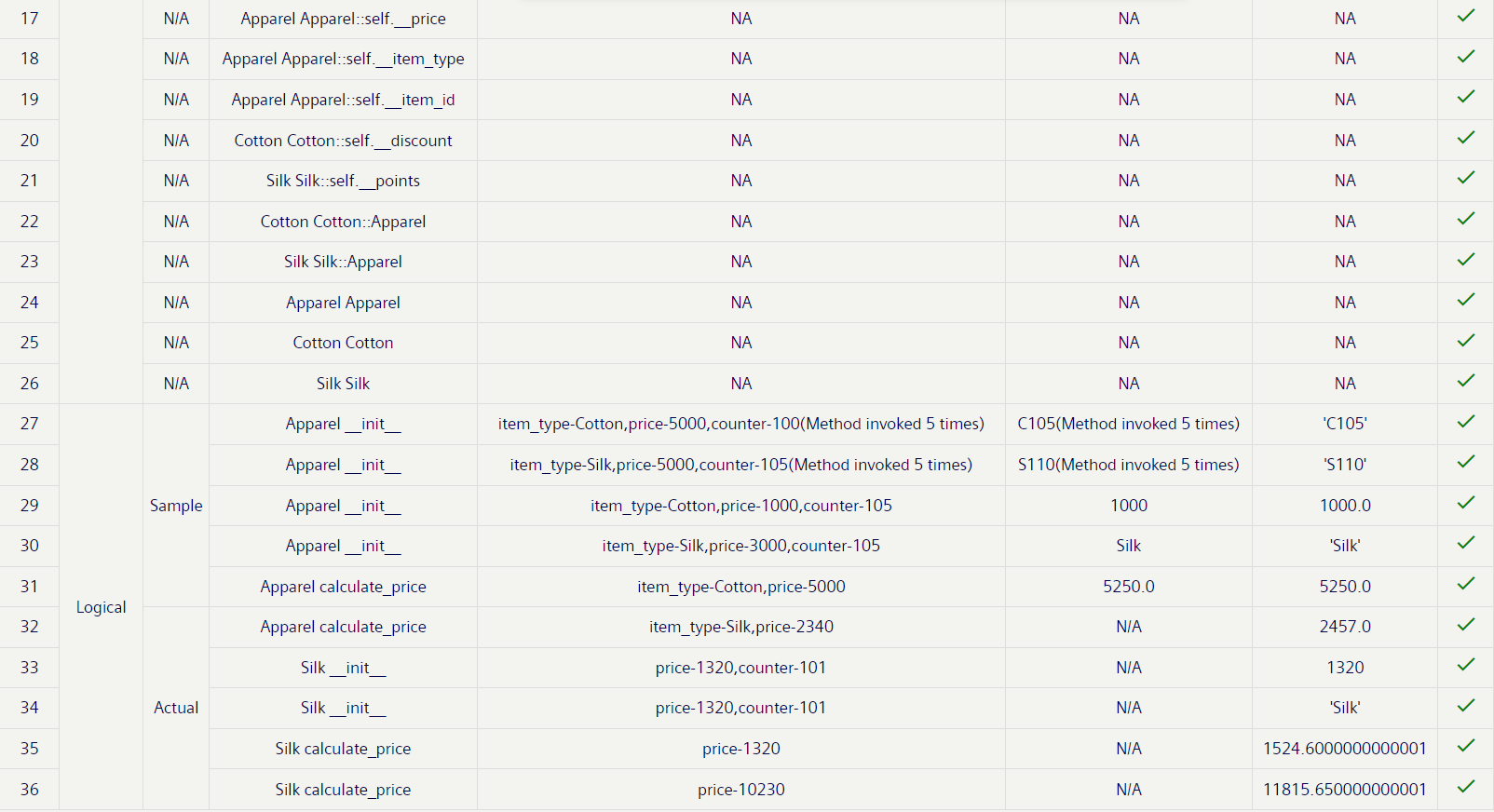
**Silk class:**

1. While invoking parent constructor from child constructor, pass "Silk" as item\_type
2. **calculate\_price():** Update attribute, price of Apparel class based on rules given below
   1. Add service tax on price by invoking appropriate method of Apparel class
   2. Identify points earned based on rules given below:
      * Silk apparels with price more than Rs. 10000, earn 10 points and anything less than or equal to that earn 3 points
   3. Initialize attribute, points with the identified points
   4. Add 10% VAT on price

**Note:** Perform case sensitive string comparison    
  
**For testing:**

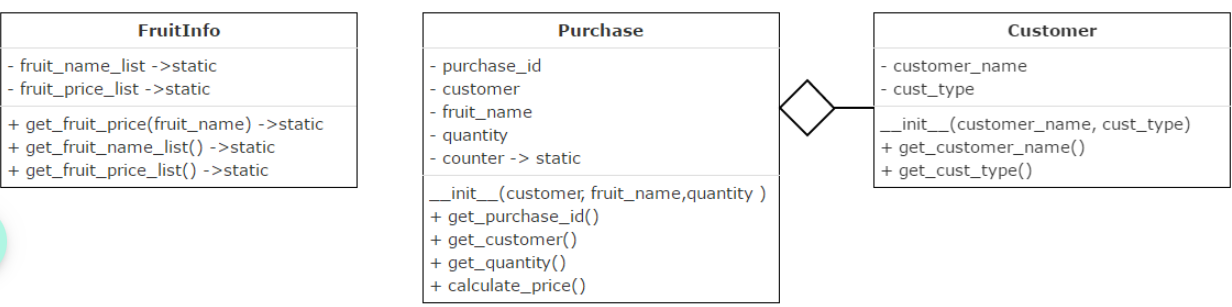
* Create objects of Cotton class and Silk class
* Invoke calculate\_price() on Cotton objects and Silk objects
* Display their details

**Verification:**[All Pass]

****

## Assignment on Dependency & Aggregation - Level 3

**Coorg Fruit Farm** is a retail chain which sells fruits grown in their orchards in Coorg, India.  
They want to keep track of customers who buy fruits from them and also the billing process. Write a python program to implement the class diagram given below.



**Class Description:  
Fruit Info class:**

1. **fruit\_name\_list:** Static list which contains the list of fruits available
2. **fruit\_price\_list:** Static list which contains the price/kg of fruits
3. The above two lists have one-to-one correspondence, initialize it with the data given in the table
4. **get\_fruit\_price(fruit\_name):** Accept a fruit name and return its price/kg. If fruit is not available, return -1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Fruit Name** | **Apple** | **Guava** | **Orange** | **Grape** | **Sweet Lime** |
| **Price per Kg** | 200 | 80 | 70 | 110 | 60 |

**Purchase class:**

1. Initialize static variable counter to 101
2. **calculate\_price()**: Calculate and return total fruit price based on rules given below
   1. For valid fruit name (hint: invoke get\_fruit\_price(fruit\_name)),
      * Calculate price based on price/kg and quantity of the fruit purchased by the customer
      * If price/kg of the fruit is maximum among the fruits in fruit lists and quantity purchased is more than 1kg, apply 2% discount on calculated price
      * If price/kg of the fruit is minimum among the fruits in fruit lists and quantity purchased is 5kg or more, apply 5% discount on calculated price
      * If the customer is a "wholesale" customer, provide an additional 10% discount. Apply this discount on already discounted price, if any one of the above two points are applicable. Else apply it on calculated price
      * Auto-generate purchase id starting from 101 prefixed by “P”. Example – P101,P102 P103 etc.
      * Return final fruit price
   2. Else, return -1.

**Note:**

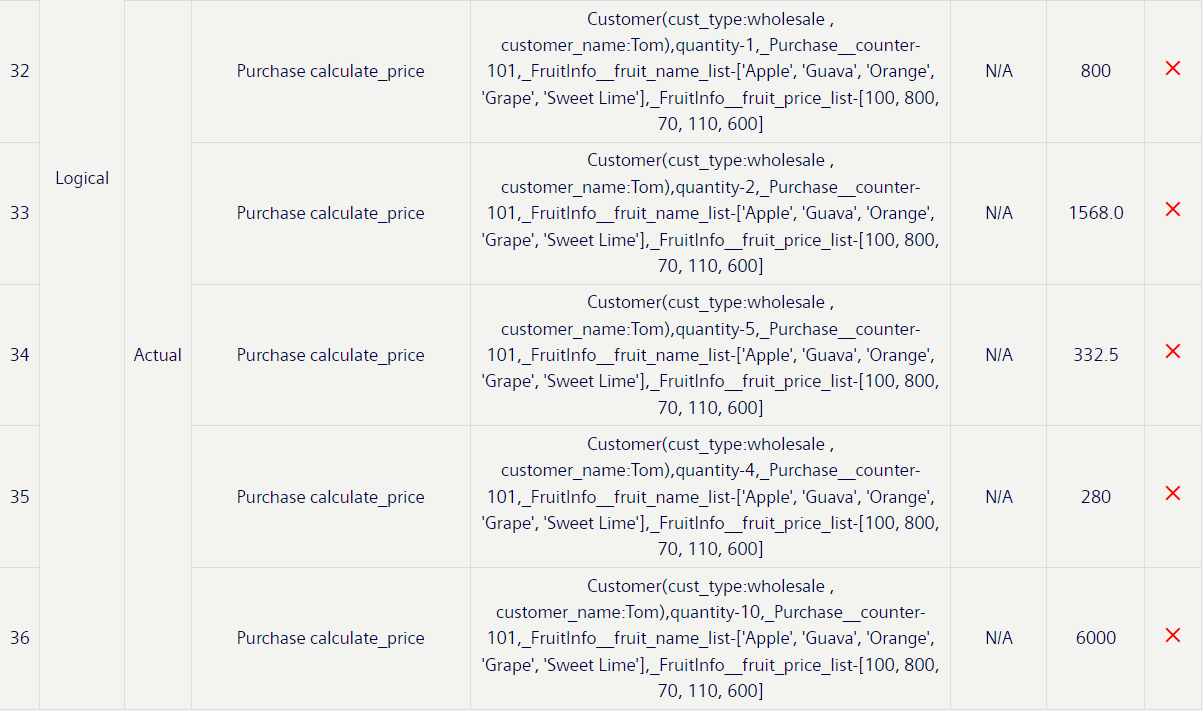
* Perform case sensitive string comparison
* There will be only one fruit with maximum price and one with minimum price

**For testing:**

* Create objects of Customer and Purchase class
* Invoke calculate\_price() on Purchase object
* Display the details

**Verification:**

*\*Below testcases failing*



**Debugging:**