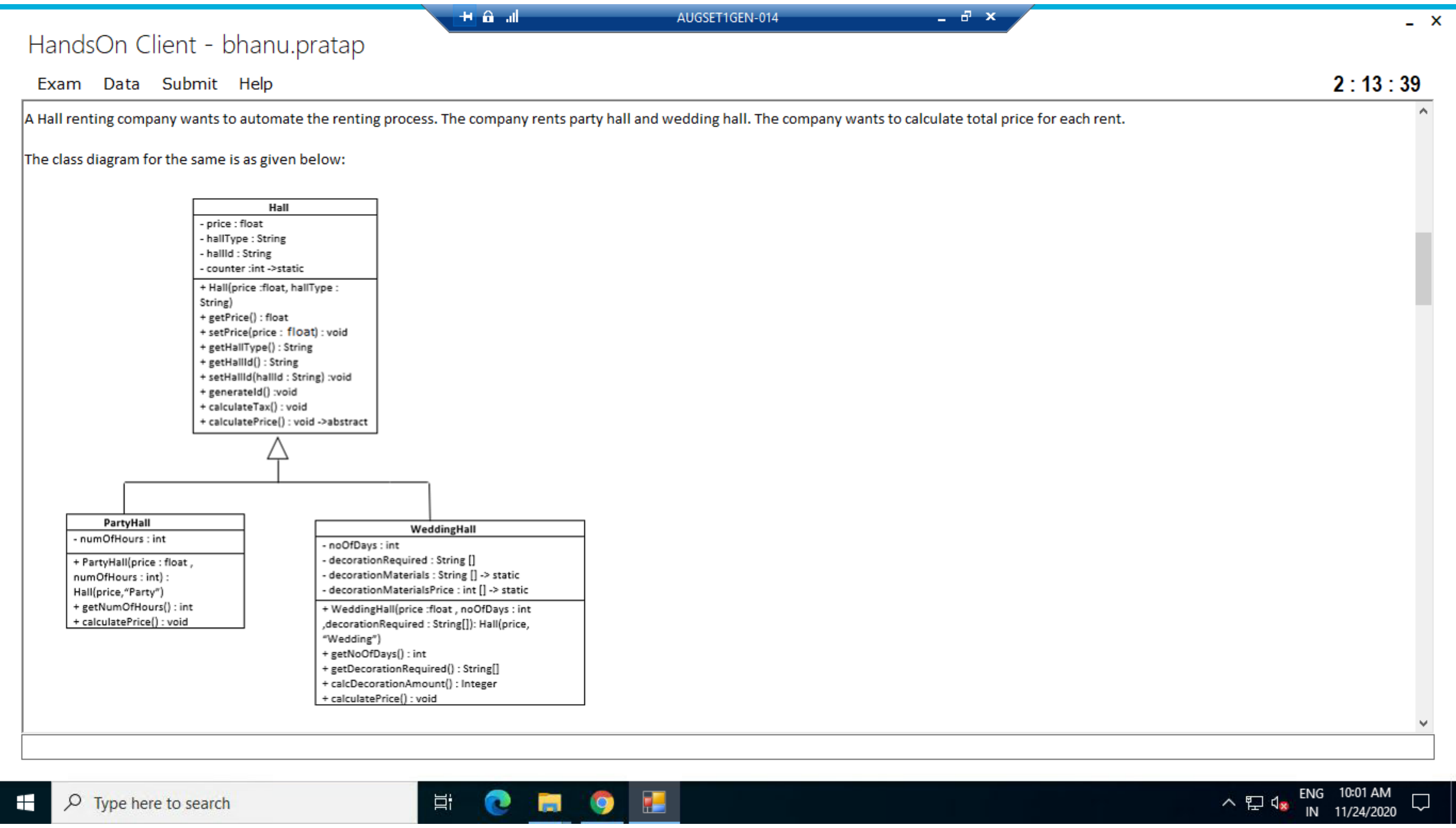


A Hall renting company wants to automate the renting process. The company rents party hall and wedding hall. The company wants to calculate total price for each rent.

The class diagram for the same is as given below:

Hall

- price : float
- hallType : String
- hallId : String
- counter :int ->static

+ Hall(price :float, hallType : String)
+ getPrice() : float
+ setPrice(price : float) : void
+ getHallType() : String
+ getHallId() : String
+ setHallId(hallId : String) :void
+ generateId() :void
+ calculateTax() : void
+ calculatePrice() : void ->abstract

PartyHall

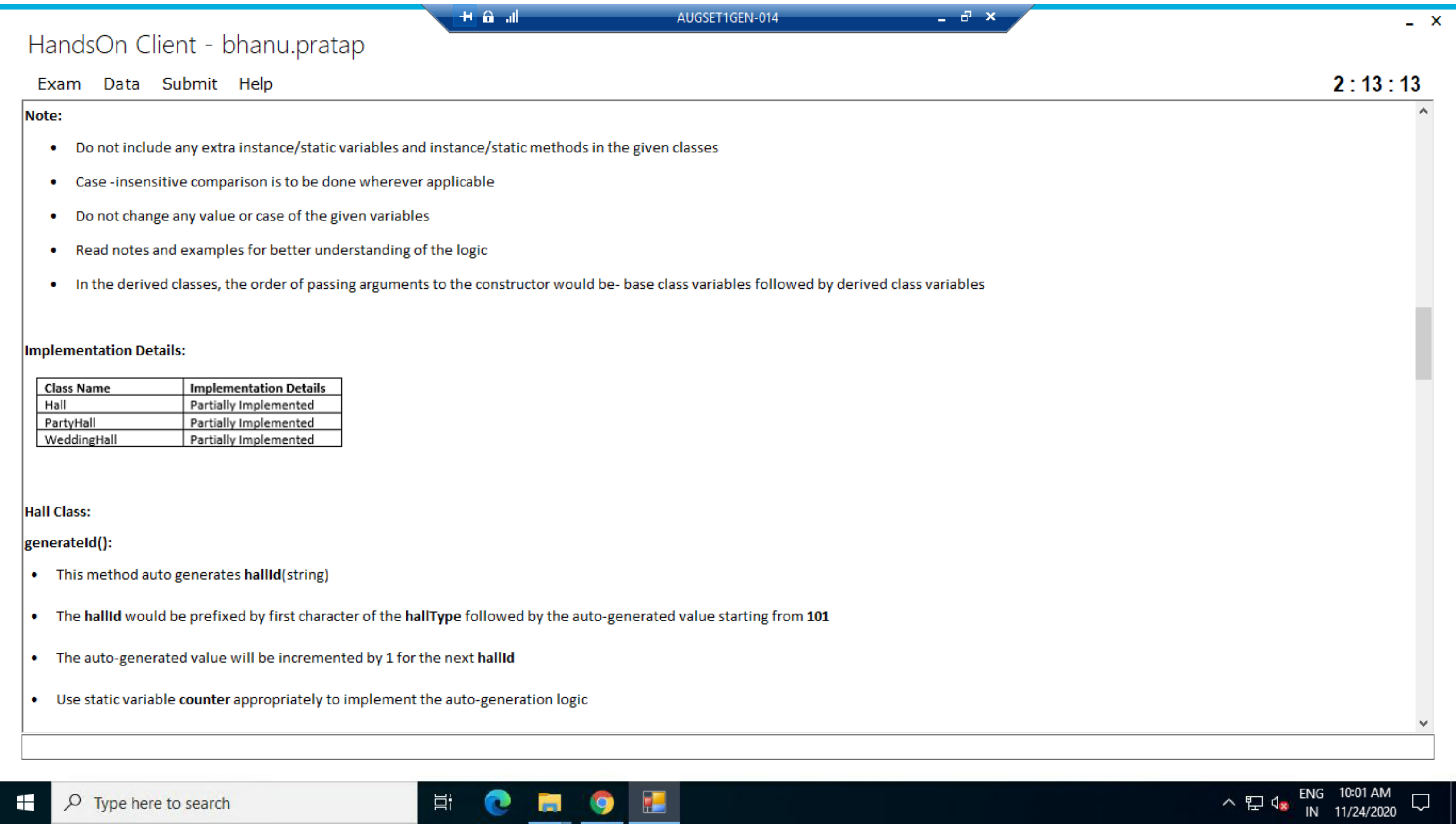
- numOfHours : int

+ PartyHall(price : float , numOfHours : int) : Hall(price,"Party")
+ getNumOfHours() : int
+ calculatePrice() : void

WeddingHall

- noOfDays : int
- decorationRequired : String []
- decorationMaterials : String [] -> static
- decorationMaterialsPrice : int [] -> static

+ WeddingHall(price :float , noOfDays : int ,decorationRequired : String[]): Hall(price, "Wedding")
+ getNoOfDays() : int
+ getDecorationRequired() : String[]
+ calcDecorationAmount() : Integer
+ calculatePrice() : void



Note:

- Do not include any extra instance/static variables and instance/static methods in the given classes
- Case -insensitive comparison is to be done wherever applicable
- Do not change any value or case of the given variables
- Read notes and examples for better understanding of the logic
- In the derived classes, the order of passing arguments to the constructor would be- base class variables followed by derived class variables

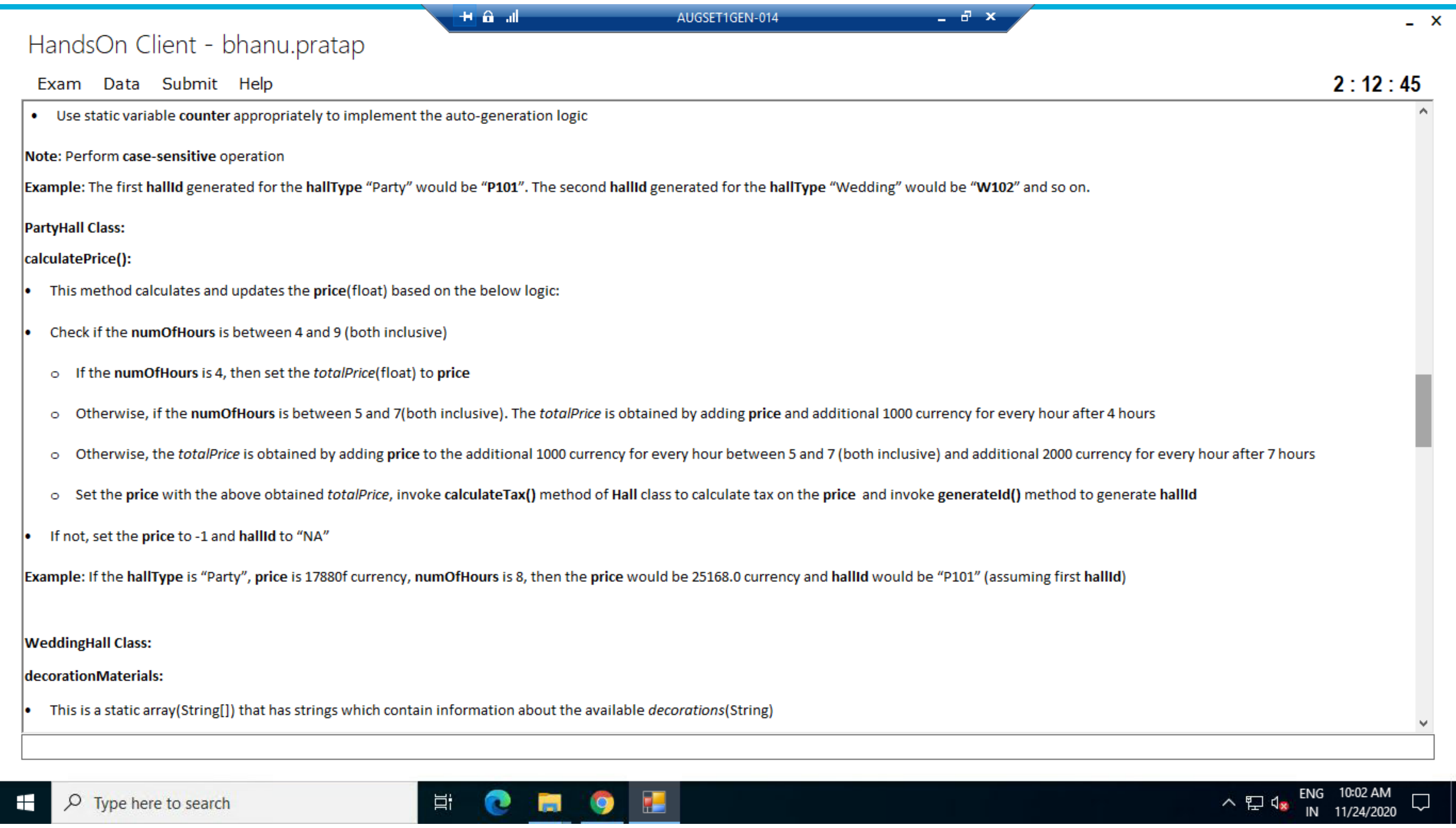
Implementation Details:

Class Name	Implementation Details
Hall	Partially Implemented
PartyHall	Partially Implemented
WeddingHall	Partially Implemented

Hall Class:

generateId():

- This method auto generates **hallId**(string)
- The **hallId** would be prefixed by first character of the **hallType** followed by the auto-generated value starting from **101**
- The auto-generated value will be incremented by 1 for the next **hallId**
- Use static variable **counter** appropriately to implement the auto-generation logic



- Use static variable **counter** appropriately to implement the auto-generation logic

Note: Perform **case-sensitive** operation

Example: The first **hallId** generated for the **hallType** “Party” would be **“P101”**. The second **hallId** generated for the **hallType** “Wedding” would be **“W102”** and so on.

PartyHall Class:

calculatePrice():

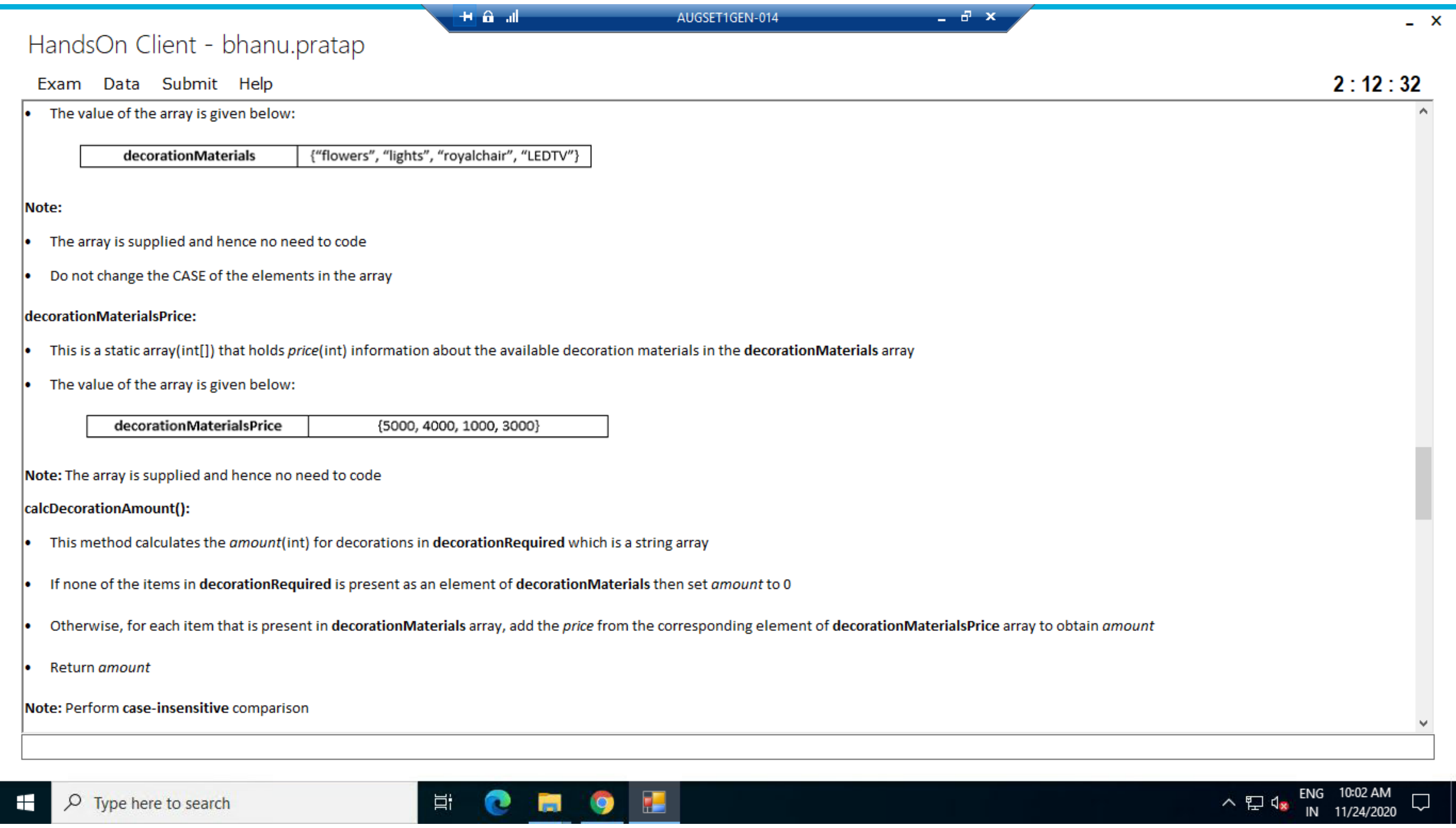
- This method calculates and updates the **price**(float) based on the below logic:
- Check if the **numOfHours** is between 4 and 9 (both inclusive)
  - If the **numOfHours** is 4, then set the *totalPrice*(float) to **price**
  - Otherwise, if the **numOfHours** is between 5 and 7(both inclusive). The *totalPrice* is obtained by adding **price** and additional 1000 currency for every hour after 4 hours
  - Otherwise, the *totalPrice* is obtained by adding **price** to the additional 1000 currency for every hour between 5 and 7 (both inclusive) and additional 2000 currency for every hour after 7 hours
  - Set the **price** with the above obtained *totalPrice*, invoke **calculateTax()** method of **Hall** class to calculate tax on the **price** and invoke **generateId()** method to generate **hallId**
- If not, set the **price** to -1 and **hallId** to “NA”

Example: If the **hallType** is “Party”, **price** is 17880f currency, **numOfHours** is 8, then the **price** would be 25168.0 currency and **hallId** would be “P101” (assuming first **hallId**)

WeddingHall Class:

decorationMaterials:

- This is a static array(String[]) that has strings which contain information about the available *decorations*(String)



- The value of the array is given below:

decorationMaterials	{"flowers", "lights", "royalchair", "LEDTV"}

Note:

- The array is supplied and hence no need to code
- Do not change the CASE of the elements in the array

decorationMaterialsPrice:

- This is a static array(int[]) that holds *price*(int) information about the available decoration materials in the **decorationMaterials** array
- The value of the array is given below:

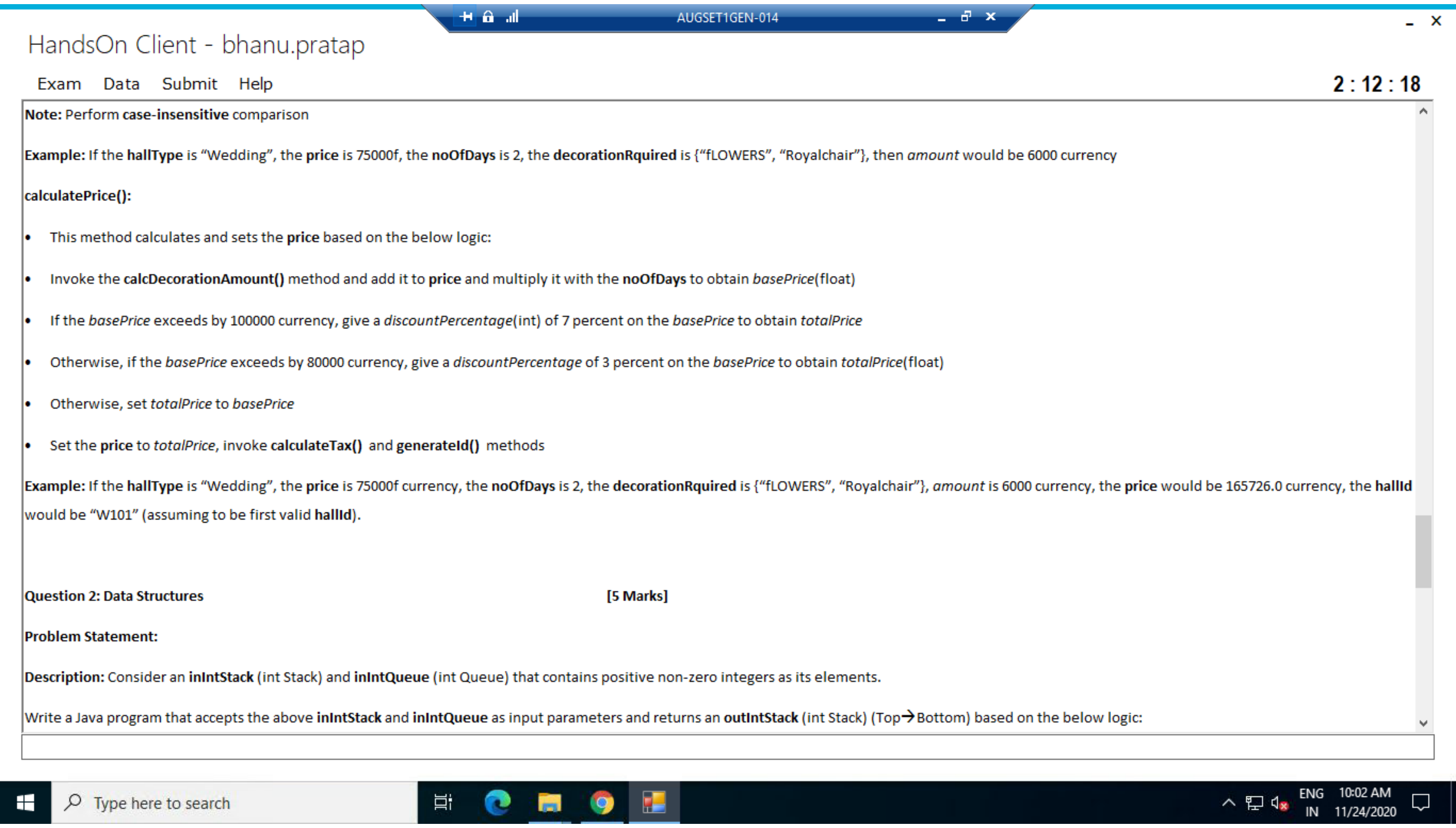
decorationMaterialsPrice	{5000, 4000, 1000, 3000}

Note: The array is supplied and hence no need to code

calcDecorationAmount():

- This method calculates the *amount*(int) for decorations in **decorationRequired** which is a string array
- If none of the items in **decorationRequired** is present as an element of **decorationMaterials** then set *amount* to 0
- Otherwise, for each item that is present in **decorationMaterials** array, add the *price* from the corresponding element of **decorationMaterialsPrice** array to obtain *amount*
- Return *amount*

Note: Perform **case-insensitive** comparison



Note: Perform **case-insensitive** comparison

Example: If the **hallType** is “Wedding”, the **price** is 75000f, the **noOfDays** is 2, the **decorationRquired** is {“fLOWERS”, “Royalchair”}, then *amount* would be 6000 currency

calculatePrice():

- This method calculates and sets the **price** based on the below logic:
- Invoke the **calcDecorationAmount()** method and add it to **price** and multiply it with the **noOfDays** to obtain *basePrice*(float)
- If the *basePrice* exceeds by 100000 currency, give a *discountPercentage*(int) of 7 percent on the *basePrice* to obtain *totalPrice*
- Otherwise, if the *basePrice* exceeds by 80000 currency, give a *discountPercentage* of 3 percent on the *basePrice* to obtain *totalPrice*(float)
- Otherwise, set *totalPrice* to *basePrice*
- Set the **price** to *totalPrice*, invoke **calculateTax()** and **generateId()** methods

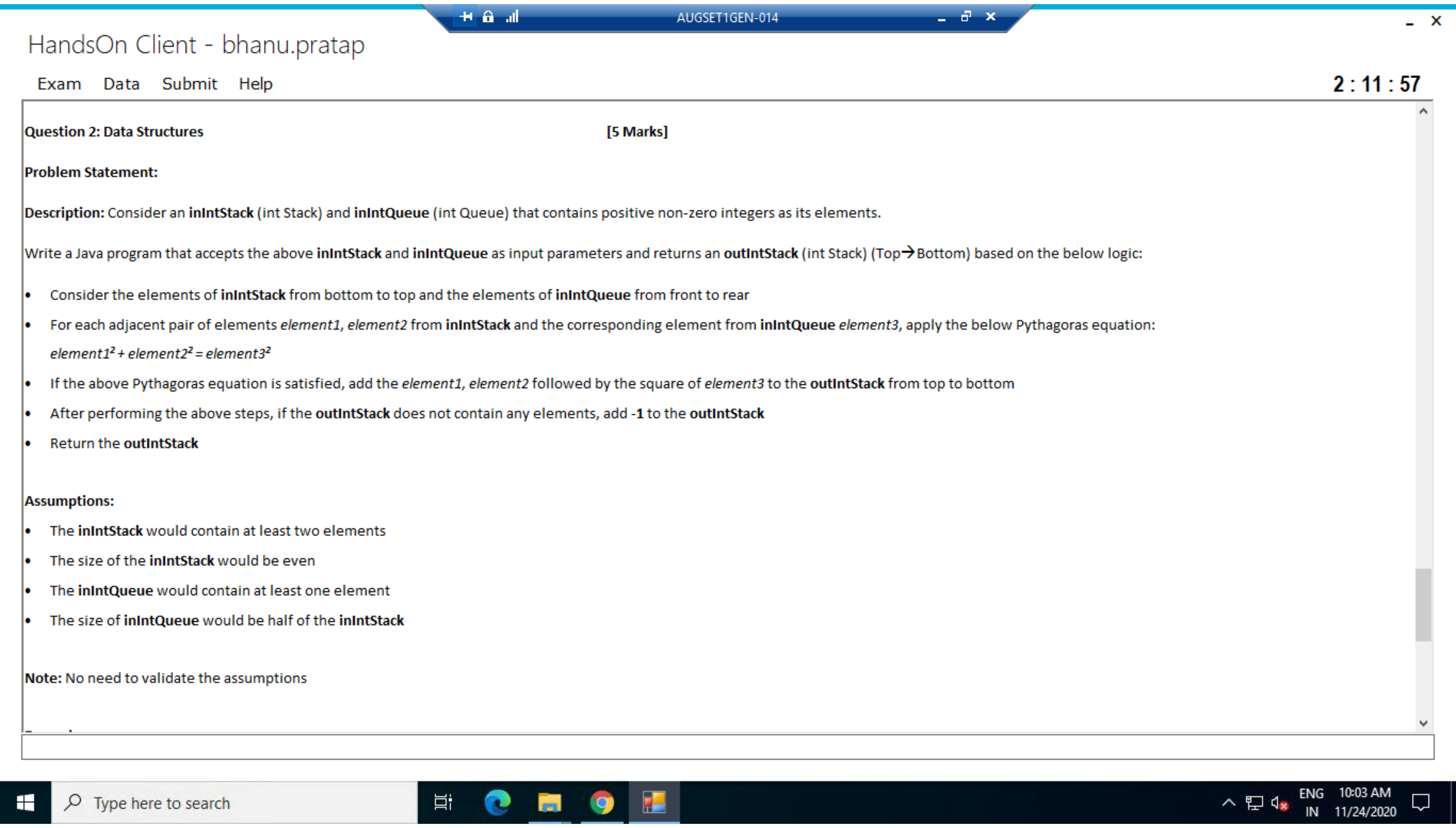
Example: If the **hallType** is “Wedding”, the **price** is 75000f currency, the **noOfDays** is 2, the **decorationRquired** is {“fLOWERS”, “Royalchair”}, *amount* is 6000 currency, the **price** would be 165726.0 currency, the **hallId** would be “W101” (assuming to be first valid **hallId**).

Question 2: Data Structures **[5 Marks]**

Problem Statement:

Description: Consider an **inIntStack** (int Stack) and **inIntQueue** (int Queue) that contains positive non-zero integers as its elements.

Write a Java program that accepts the above **inIntStack** and **inIntQueue** as input parameters and returns an **outIntStack** (int Stack) (Top→Bottom) based on the below logic:



Question 2: Data Structures **[5 Marks]**

Problem Statement:

Description: Consider an **inIntStack** (int Stack) and **inIntQueue** (int Queue) that contains positive non-zero integers as its elements.

Write a Java program that accepts the above **inIntStack** and **inIntQueue** as input parameters and returns an **outIntStack** (int Stack) (Top→Bottom) based on the below logic:

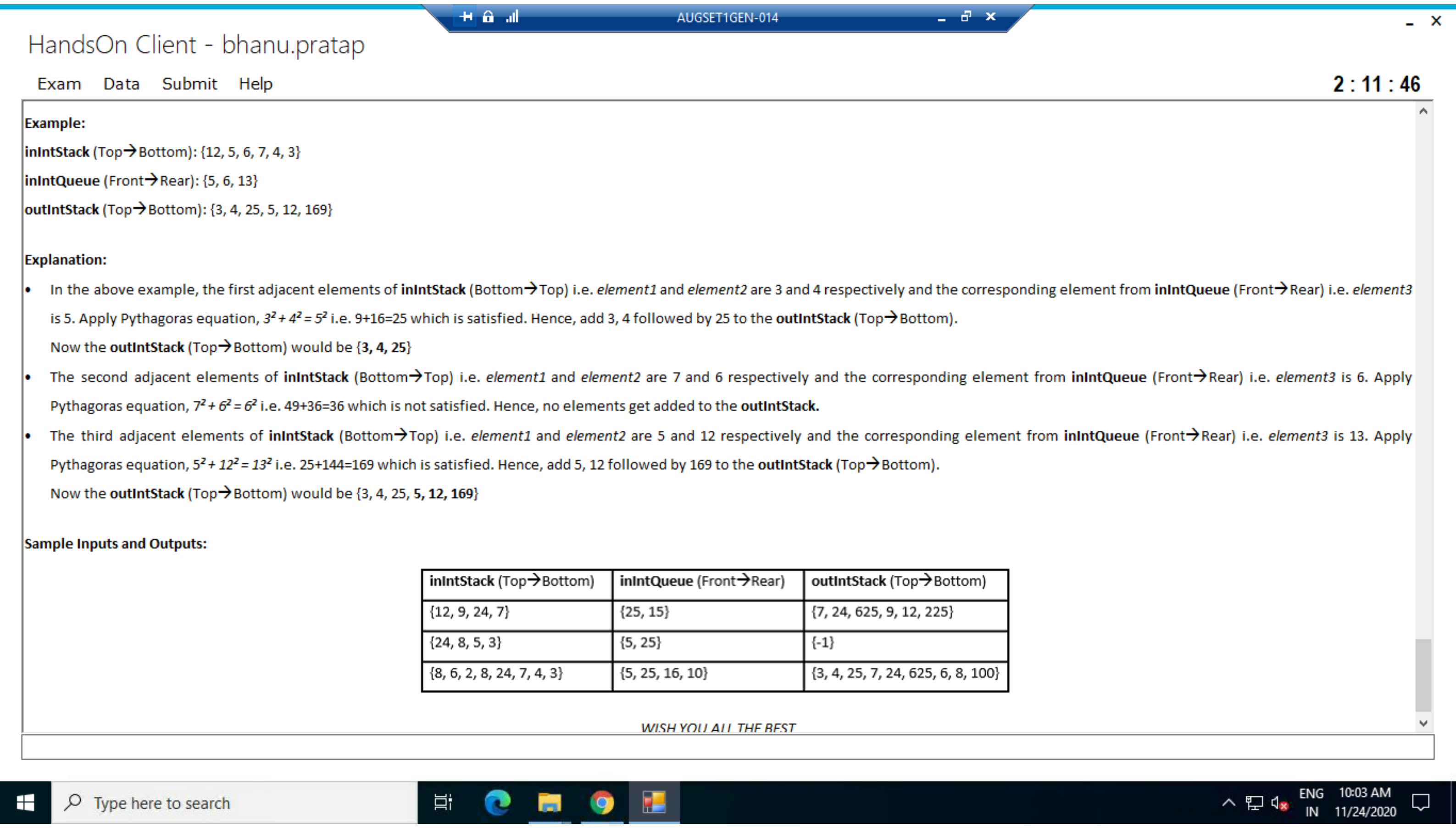
- Consider the elements of **inIntStack** from bottom to top and the elements of **inIntQueue** from front to rear
- For each adjacent pair of elements *element1*, *element2* from **inIntStack** and the corresponding element from **inIntQueue** *element3*, apply the below Pythagoras equation:

  $element1^2 + element2^2 = element3^2$
- If the above Pythagoras equation is satisfied, add the *element1*, *element2* followed by the square of *element3* to the **outIntStack** from top to bottom
- After performing the above steps, if the **outIntStack** does not contain any elements, add **-1** to the **outIntStack**
- Return the **outIntStack**

Assumptions:

- The **inIntStack** would contain at least two elements
- The size of the **inIntStack** would be even
- The **inIntQueue** would contain at least one element
- The size of **inIntQueue** would be half of the **inIntStack**

Note: No need to validate the assumptions



Example:

inIntStack (Top→Bottom): {12, 5, 6, 7, 4, 3}

inIntQueue (Front→Rear): {5, 6, 13}

outIntStack (Top→Bottom): {3, 4, 25, 5, 12, 169}

Explanation:

- In the above example, the first adjacent elements of **inIntStack** (Bottom→Top) i.e. *element1* and *element2* are 3 and 4 respectively and the corresponding element from **inIntQueue** (Front→Rear) i.e. *element3* is 5. Apply Pythagoras equation, $3^2 + 4^2 = 5^2$ i.e. 9+16=25 which is satisfied. Hence, add 3, 4 followed by 25 to the **outIntStack** (Top→Bottom).

  Now the **outIntStack** (Top→Bottom) would be {**3, 4, 25**}
- The second adjacent elements of **inIntStack** (Bottom→Top) i.e. *element1* and *element2* are 7 and 6 respectively and the corresponding element from **inIntQueue** (Front→Rear) i.e. *element3* is 6. Apply Pythagoras equation, $7^2 + 6^2 = 6^2$ i.e. 49+36=36 which is not satisfied. Hence, no elements get added to the **outIntStack.**
- The third adjacent elements of **inIntStack** (Bottom→Top) i.e. *element1* and *element2* are 5 and 12 respectively and the corresponding element from **inIntQueue** (Front→Rear) i.e. *element3* is 13. Apply Pythagoras equation, $5^2 + 12^2 = 13^2$ i.e. 25+144=169 which is satisfied. Hence, add 5, 12 followed by 169 to the **outIntStack** (Top→Bottom).

  Now the **outIntStack** (Top→Bottom) would be {3, 4, 25, **5, 12, 169**}

Sample Inputs and Outputs:

inIntStack (Top→Bottom)	**inIntQueue** (Front→Rear)	**outIntStack** (Top→Bottom)
{12, 9, 24, 7}	{25, 15}	{7, 24, 625, 9, 12, 225}
{24, 8, 5, 3}	{5, 25}	{-1}
{8, 6, 2, 8, 24, 7, 4, 3}	{5, 25, 16, 10}	{3, 4, 25, 7, 24, 625, 6, 8, 100}

WISH YOU ALL THE BEST

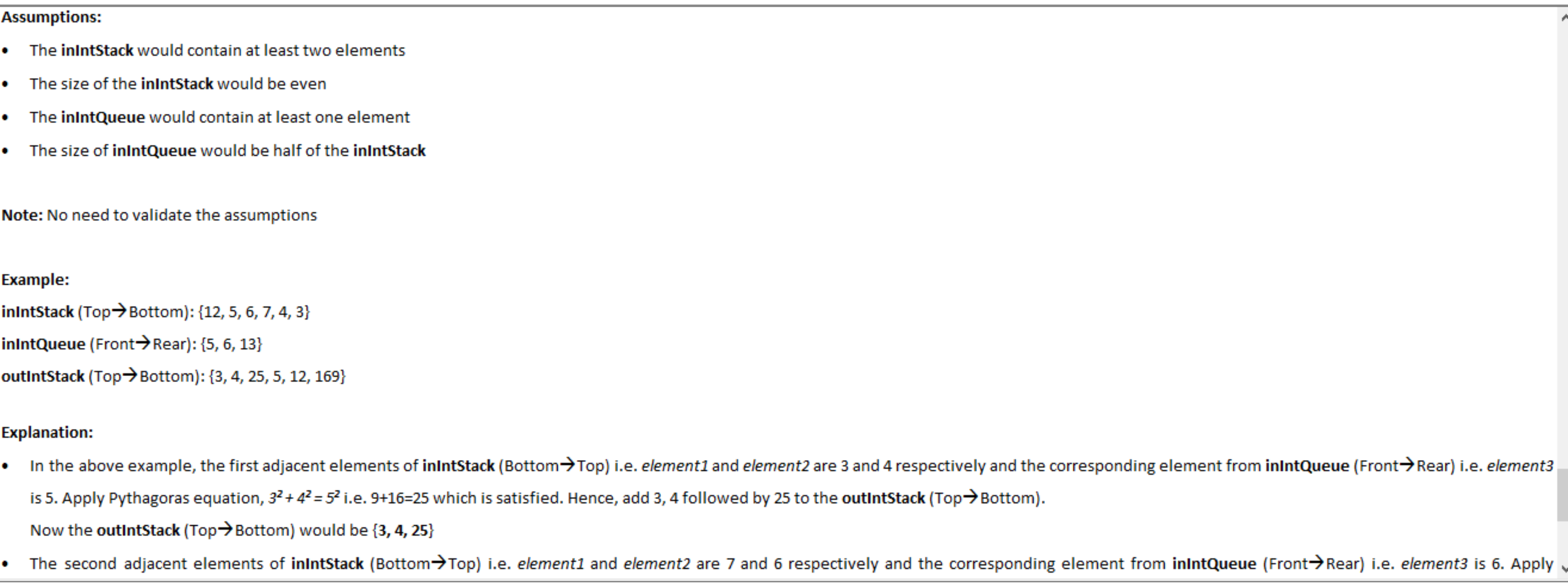
Question 2: Data Structures **[5 Marks]**

Problem Statement:

Description: Consider an **inIntStack** (int Stack) and **inIntQueue** (int Queue) that contains positive non-zero integers as its elements.

Write a Java program that accepts the above **inIntStack** and **inIntQueue** as input parameters and returns an **outIntStack** (int Stack) (Top→Bottom) based on the below logic:

- Consider the elements of **inIntStack** from bottom to top and the elements of **inIntQueue** from front to rear
- For each adjacent pair of elements *element1*, *element2* from **inIntStack** and the corresponding element from **inIntQueue** *element3*, apply the below Pythagoras equation: $element1^2 + element2^2 = element3^2$
- If the above Pythagoras equation is satisfied, add the *element1*, *element2* followed by the square of *element3* to the **outIntStack** from top to bottom
- After performing the above steps, if the **outIntStack** does not contain any elements, add **-1** to the **outIntStack**
- Return the **outIntStack**



Assumptions:

- The **inIntStack** would contain at least two elements
- The size of the **inIntStack** would be even
- The **inIntQueue** would contain at least one element
- The size of **inIntQueue** would be half of the **inIntStack**

Note: No need to validate the assumptions

Example:

inIntStack (Top→Bottom): {12, 5, 6, 7, 4, 3}

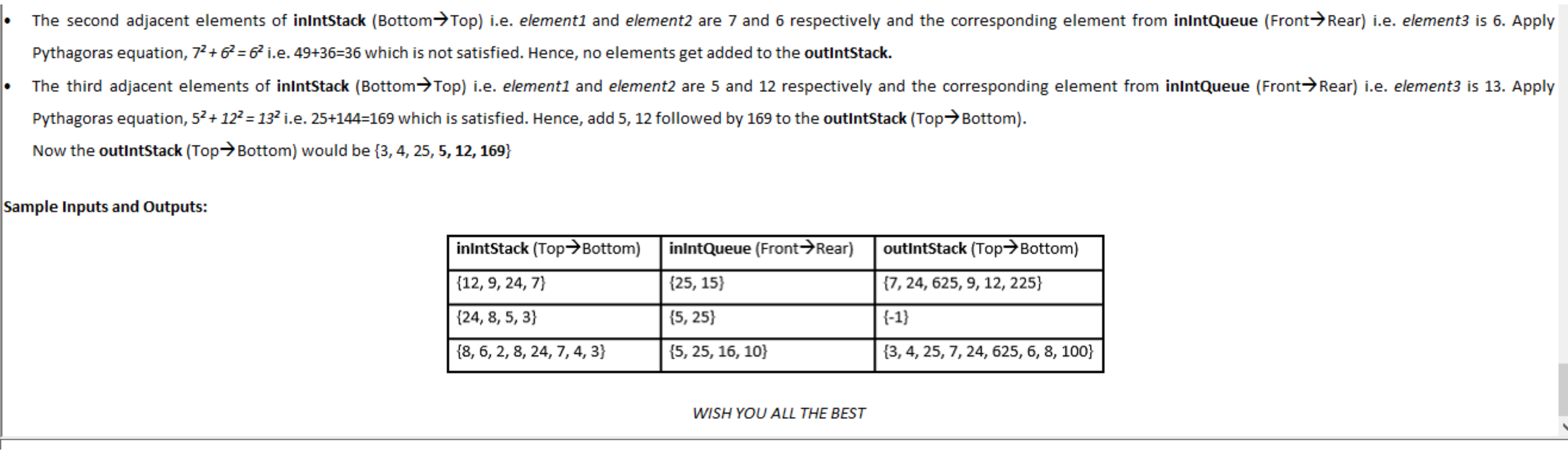
inIntQueue (Front→Rear): {5, 6, 13}

outIntStack (Top→Bottom): {3, 4, 25, 5, 12, 169}

Explanation:

- In the above example, the first adjacent elements of **inIntStack** (Bottom→Top) i.e. *element1* and *element2* are 3 and 4 respectively and the corresponding element from **inIntQueue** (Front→Rear) i.e. *element3* is 5. Apply Pythagoras equation, $3^2 + 4^2 = 5^2$ i.e. 9+16=25 which is satisfied. Hence, add 3, 4 followed by 25 to the **outIntStack** (Top→Bottom).

  Now the **outIntStack** (Top→Bottom) would be {**3, 4, 25**}
- The second adjacent elements of **inIntStack** (Bottom→Top) i.e. *element1* and *element2* are 7 and 6 respectively and the corresponding element from **inIntQueue** (Front→Rear) i.e. *element3* is 6. Apply



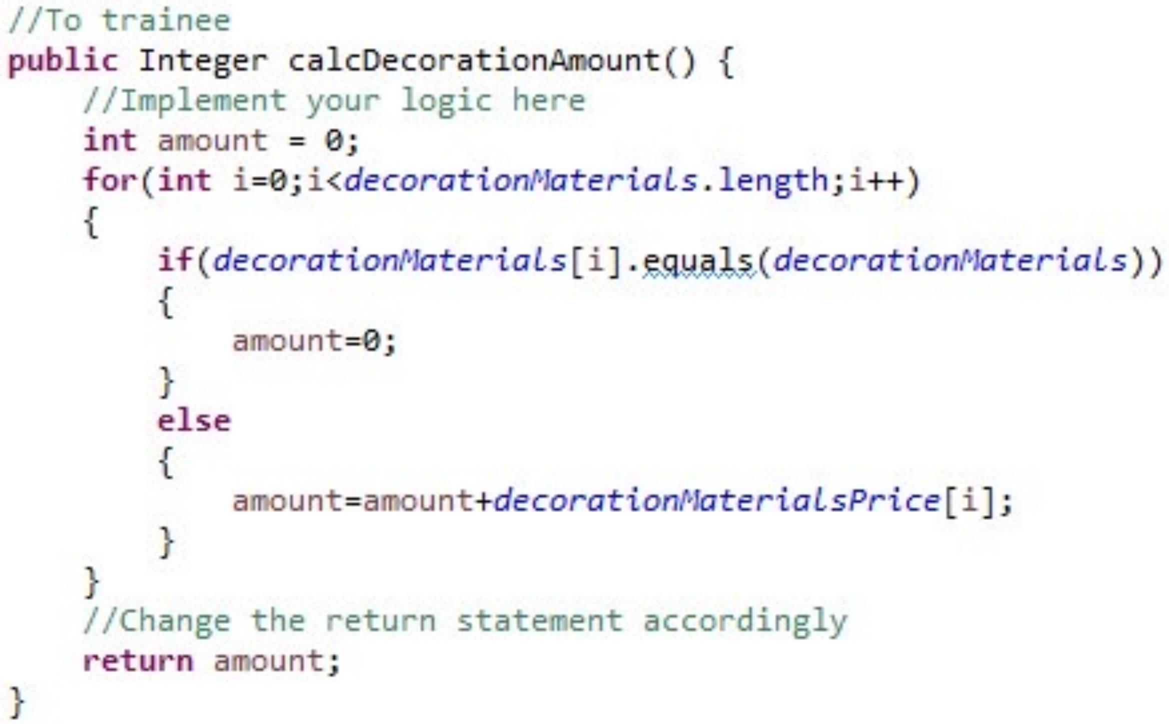
- The second adjacent elements of **inIntStack** (Bottom→Top) i.e. *element1* and *element2* are 7 and 6 respectively and the corresponding element from **inIntQueue** (Front→Rear) i.e. *element3* is 6. Apply Pythagoras equation, $7^2 + 6^2 = 6^2$ i.e. 49+36=36 which is not satisfied. Hence, no elements get added to the **outIntStack.**
- The third adjacent elements of **inIntStack** (Bottom→Top) i.e. *element1* and *element2* are 5 and 12 respectively and the corresponding element from **inIntQueue** (Front→Rear) i.e. *element3* is 13. Apply Pythagoras equation, $5^2 + 12^2 = 13^2$ i.e. 25+144=169 which is satisfied. Hence, add 5, 12 followed by 169 to the **outIntStack** (Top→Bottom).

  Now the **outIntStack** (Top→Bottom) would be {3, 4, 25, **5, 12, 169**}

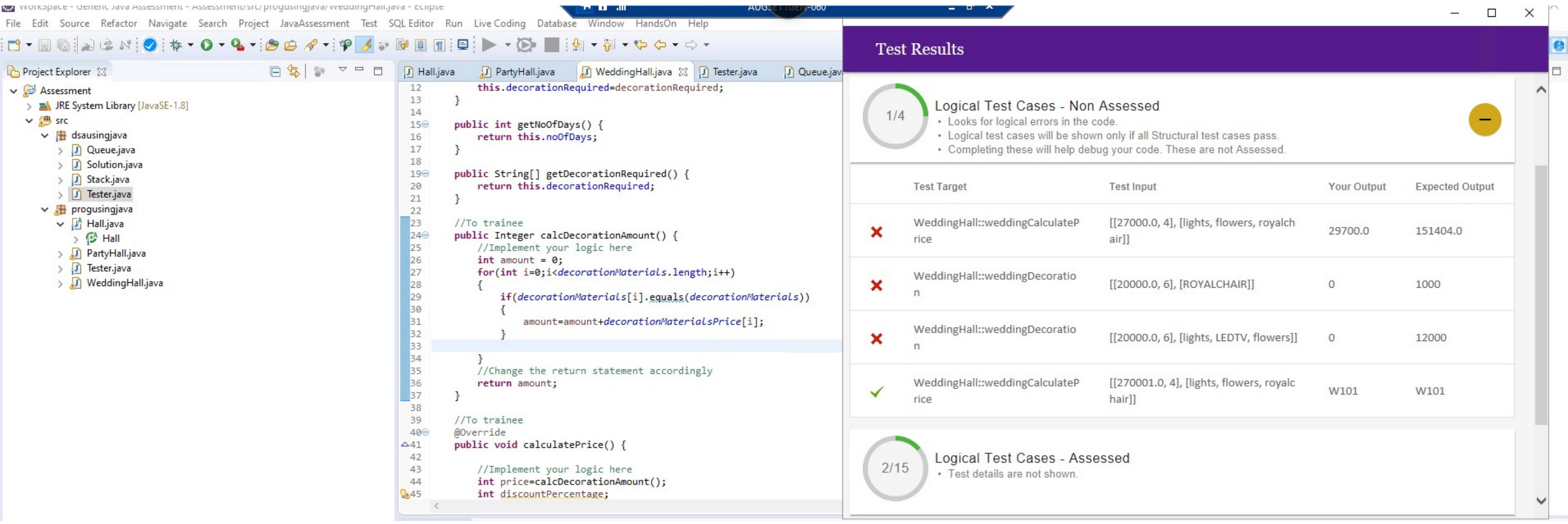
Sample Inputs and Outputs:

inIntStack (Top→Bottom)	**inIntQueue** (Front→Rear)	**outIntStack** (Top→Bottom)
{12, 9, 24, 7}	{25, 15}	{7, 24, 625, 9, 12, 225}
{24, 8, 5, 3}	{5, 25}	{-1}
{8, 6, 2, 8, 24, 7, 4, 3}	{5, 25, 16, 10}	{3, 4, 25, 7, 24, 625, 6, 8, 100}

WISH YOU ALL THE BEST



```
//To trainee
public Integer calcDecorationAmount() {
    //Implement your logic here
    int amount = 0;
    for(int i=0;i<decorationMaterials.length;i++)
    {
        if(decorationMaterials[i].equals(decorationMaterials))
        {
            amount=0;
        }
        else
        {
            amount=amount+decorationMaterialsPrice[i];
        }
    }
    //Change the return statement accordingly
    return amount;
}
```



```
        this.decorationRequired=decorationRequired;
    }

    public int getNoOfDays() {
        return this.noOfDays;
    }

    public String[] getDecorationRequired() {
        return this.decorationRequired;
    }

    //To trainee
    public Integer calcDecorationAmount() {
        //Implement your logic here
        int amount = 0;
        for(int i=0;i<decorationMaterials.length;i++)
        {
            if(decorationMaterials[i].equals(decorationMaterials))
            {
                amount=amount+decorationMaterialsPrice[i];
            }

        }
        //Change the return statement accordingly
        return amount;
    }

    //To trainee
    @Override
    public void calculatePrice() {

        //Implement your logic here
        int price=calcDecorationAmount();
        int discountPercentage;
```

Test Results

1/4 Logical Test Cases - Non Assessed

- Looks for logical errors in the code.
- Logical test cases will be shown only if all Structural test cases pass.
- Completing these will help debug your code. These are not Assessed.

	Test Target	Test Input	Your Output	Expected Output
✗	WeddingHall::weddingCalculatePrice	[[27000.0, 4], [lights, flowers, royalchair]]	29700.0	151404.0
✗	WeddingHall::weddingDecoration	[[20000.0, 6], [ROYALCHAIR]]	0	1000
✗	WeddingHall::weddingDecoration	[[20000.0, 6], [lights, LEDTV, flowers]]	0	12000
✓	WeddingHall::weddingCalculatePrice	[[270001.0, 4], [lights, flowers, royalchair]]	W101	W101

2/15 Logical Test Cases - Assessed

- Test details are not shown.