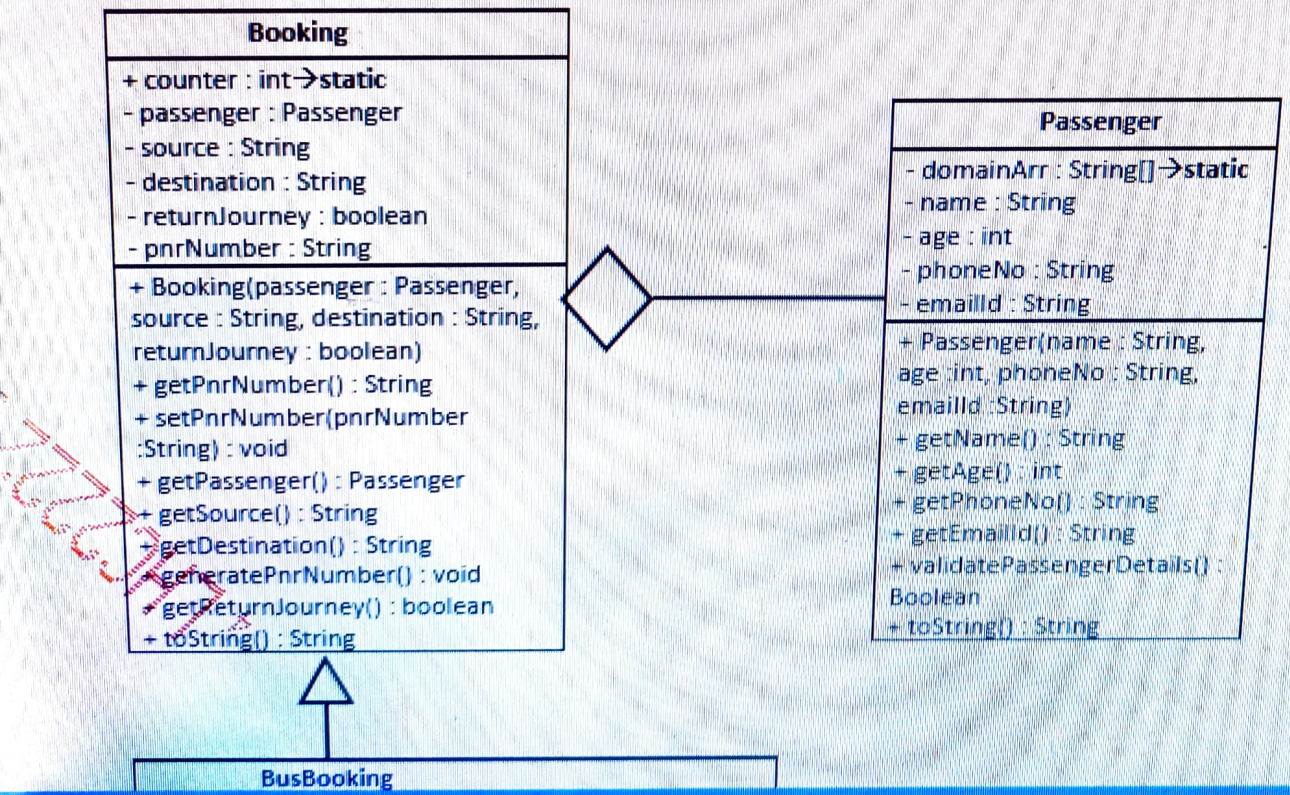
uestion1: Object Oriented Programming:

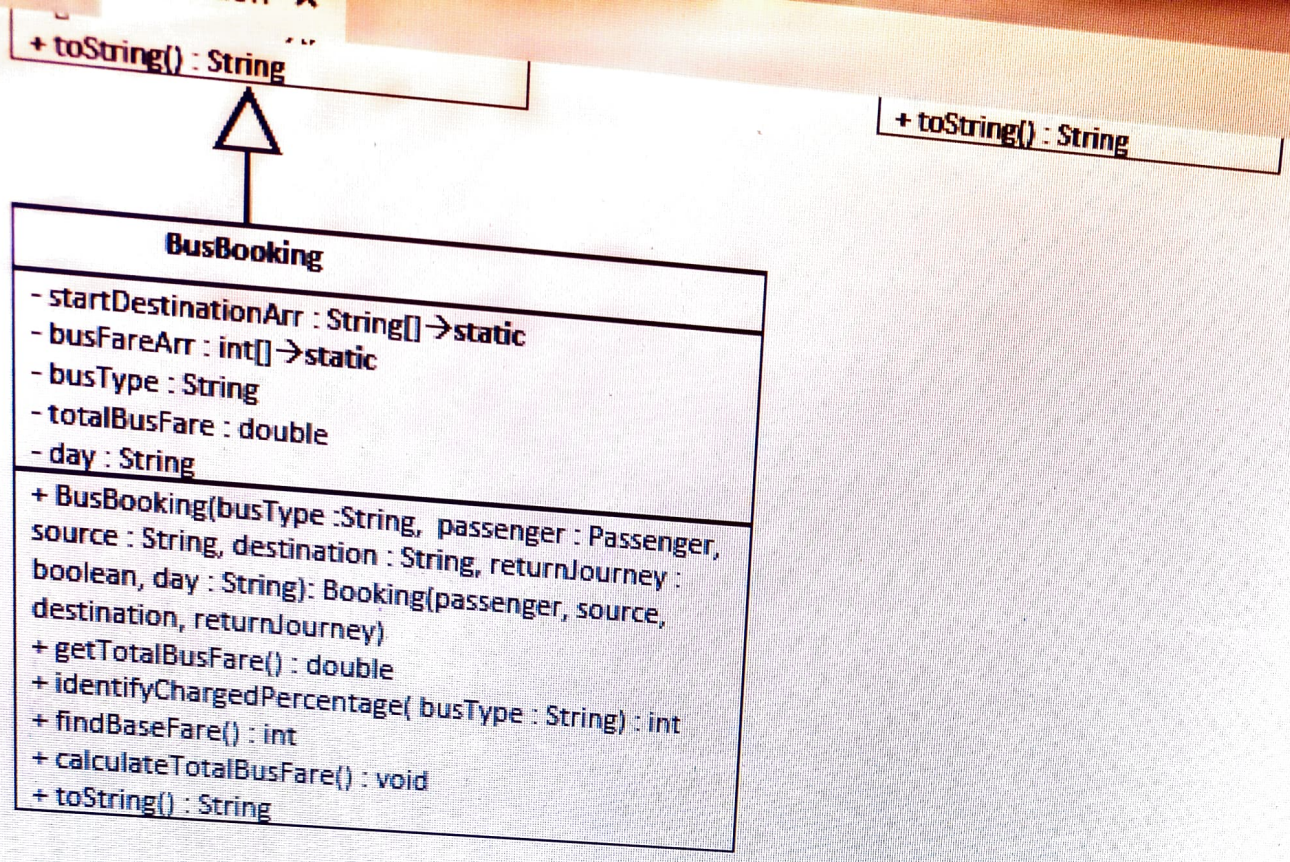
escription:

A popular transport service provider wants to automate its bus fare calculation system based on the bus type of passenger.

Write a Java program to implement the below class diagram.

Class Diagram:





ote:

- Do not include any extra instance/static variables and instance/static methods in the given classes
- Case sensitive 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

nplementation Details:

Class Name	Status of Implementation
Booking	Partially Implemented
Bus Booking	Partially Implemented
Passenger	Partially Implemented

ooking Class:

eneratePnrNumber():

- This method auto-generates and sets the **pnrNumber**(String)

- This method auto-generates and sets the **pnrNumber**(String)
- The **pnrNumber** would be prefixed with the first letter of passenger **name** in uppercase followed by auto-generated value starting from 1001
- The auto-generated value should be incremented by 1 for the next **pnrNumber**
- Use static variable **counter** appropriately to implement the auto-generation logic

Example: If the passenger **name** is **Peter**, then the **pnrNumber** would be **P1001**, second **pnrNumber** generated would be **R1002** if the passenger **name** is **Royal**

Passenger Class:

domainArr:

- This is a static array (String[]) which contains information about **emailId**'s *domain*
- The initial values of the **domainArr** are given below

domainArr	{"gmail", "yahoo", "hotmail"}

Note:

- The array is supplied. Hence no need to code
- Do not change the **CASE** of elements in the array

validatePassengerDetails():

- This method validates the passenger details and returns a Boolean
- Check if **age** is above 12 and length of **name** is greater than or equal to 3 and **phoneNo** is starting with character 9 and **emailId** contains any of the *domain* from **domainArr**
  Note: Perform **case-sensitive** comparison
- If so, then this method must return true
- Otherwise, return false

Assumption: phoneNo would contain 10 digits only

Note: No need to validate the assumption

Example: If the **age** is 26, **name** is Peter, **phoneNo** is 9865768743 and **emailId** abc123@gmail.com then this method must return true

BusBooking Class:

startDestinationArr:

- This is a static array (String[]) which contains information about *journey* in the format where first three characters of **source**(String) and **destination**(String) of the journey separated by **-**
- The initial values of the **startDestinationArr** are given below **:**

startDestinationArr	{"FLO-CAL", "CAL-GEO", "CAL-WAS"}

c2Fzd2F0LnBhbmlncmFoaTAx

Note:

- The array is supplied. Hence no need to code
- Do not change the **CASE** of elements in the array

busFareArr:

- This is a static array(int[]) which has *baseFare* as its elements
- This array has one to one correspondence with **startDestinationArr**
- The initial values of the **busFareArr** are given below **:**

busFareArr	{200, 400, 600}

Note:

- This array is supplied. Hence no need to code

findBaseFare():

- This method identifies and returns the *baseFare*(int) using **startDestinationArr** and **busFareArr**
- Identify *onwardJourney* (String) by concatenating first three characters of **source** followed by hyphen (**-**) and then first three characters of **destination**
- Identify *returnJourney* (String) by concatenating first three characters of **destination** followed by hyphen (**-**) and then first three characters of **source**
- Check if either *onwardJourney* or *returnJourney* is present as any one of the elements in **startDestinationArr**
  NOTE: Perform **case-insensitive** comparison
- If it is present then, return the corresponding *baseFare* from **busFareArr**
- Otherwise, return -1

NOTE: The *baseFare* of **source** to **destination** and **destination** to **source** would remain the same

Question ×

mple: If the **source** is ◆florida◆ and **destination** is ◆california◆ then this method would return 200 currency

culateTotalBusFare():

- This method calculates and sets the **totalBusFare** (double) and generates the **pnrNumber** based on the below logic:
- Invoke **validatePassengerDetails()** method of **Passenger** class
- If the above method returns false, then set **totalBusFare** and **pnrNumber** to **-1.0** and **◆NA◆** respectively
- Otherwise,
  - Identify *baseFare* (int) by invoking **findBaseFare()** method
  - Identify *taxPercentage* (int) by invoking **identifyChargedPercentage()** and by passing **busType** as parameter
  - If *baseFare* and *taxPercentage* is not -1 then,
    - Obtain *finalFare* (double) by applying additional *taxPercentage* on *baseFare*
    - If the **day** is either ◆FRIDAY◆ or ◆SATURDAY◆ or ◆SUNDAY◆ then add additional 30 currency to the *finalFare*
      Note: Perform **case-insensitive** comparison
    - If the **age** of the passenger is 60 or above, then provide 50% discount on *finalFare*
    - If **returnJourney** is **true,** then *finalFare* would be twice of its current value and further add additional 20 currency as reservation charge for onward and return journey
    - Otherwise, add additional 10 currency as reservation charge for onward journey to the *finalFare* and set the **totalBaseFare** with obtained *finalFare*
    - Invoke **generatePnrNumber()** method
  - Otherwise, set **totalBusFare** and **pnrNumber** to **-1.0** and **◆NA◆** respectively

c2Fzd2F0LnBhbmlncmFoaTAx

xample:

f the passenger **name** is ◆Peter◆, **age** is 26, **emailId** is ◆abc123@gmail.com◆, **phoneNo** is ◆9865768743◆ **busType** is ◆SEMISLEEPER◆, **source** is ◆florida◆, **destination** is ◆california◆, **returnJourne**
rue and the **day** is ◆Saturday◆ then the **totalBusFare** would be 520.0 currency and **pnrNumber** would be ◆P1001◆(Assuming that first passenger).

Question 2: Data Structures:

[5 Marks]

Problem Statement:

Description:

Write a Java program that accepts non-empty **inStrStack** (String Stack) (Top -> Bottom) and **inIntQueue** (int Queue) (Front -> Rear) as input parameters and returns **outStrStack**(String Stack) (Top -> Bottom)based on the below logic:

- Consider *data1* as front element of **inIntQueue** and *data2* as second element of **inIntQueue**
- Consider *stackData* as top element of **inStrStack**
- Check if sum of *data1* and *data2* is present in *stackData*. If yes, then add *stackData* to **outStrStack**

escription:

Write a Java program that accepts non-empty **inStrStack** (String Stack) (Top -> Bottom) and **inIntQueue** (int Queue) (Front -> Rear) as input parameters and returns **outStr**
Bottom)based on the below logic:

- Consider *data1* as front element of **inIntQueue** and *data2* as second element of **inIntQueue**
- Consider *stackData* as top element of **inStrStack**
- Check if sum of *data1* and *data2* is present in *stackData*. If yes, then add *stackData* to **outStrStack**
- Otherwise, check if last digit of sum is present in *stackData*. If yes, then add *stackData* concatenated with twice the sum to **outStrStack**
- Otherwise, add *stackData* to temporary data structure
- Repeat the above steps for all the elements of **inIntQueue** and **inStrStack**
- Empty the contents of temporary data structure to **outStrStack** (Top -> Bottom) such that they appear in the reverse order of their occurrence in temporary data structure

c2Fzd2F0LnBhbmlncmFoaT

Assumptions:

- **inIntQueue** would have at least 2 elements and **inStrStack** would have at least one element
- **inIntQueue** would always contain twice the number of elements as **inStrStack**

Note: No need to validate the assumptions

Example:

inStrStack (Top -> Bottom): **999**, **tr5**, **kil8**, **123**

inIntQueue (Front -> Rear): 3, 6, 17, 8, 31, 6, 8, 9

outStrStack (Top -> Bottom) : **kil8**, **123**, **tr550**, **999**

- In the above example, *stackData* is **999**, *data1* is 3 and *data2* is 6. The sum of *data1* and *data2* i.e. 3+6 is 9. Since it is present in **999** the *stackData* is added to **outStrStack**.
  Hence, **outStrStack** (Top -> Bottom) would be : **999**
- Now, the next *stackData* is **tr5**, *data1* is 17 and *data2* is 8. The sum of *data1* and *data2* i.e. 17+8 is 25. Since, the sum's last digit 5 is present in **tr5** the *stackData* is concatenated with twice i.e. **tr550** is added to **outStrStack.**
  Hence, **outStrStack** (Top -> Bottom) would be: **tr550**, **999**
- Now, the next *stackData* is **kil8**, *data1* is 31 and *data2* is 6. The sum of *data1* and *data2* i.e. 31+6 is 37. Since, its sum and last digit is not present in **kil8** the *stackData* is added to tempor structure.

inStrStack (Top -> Bottom): **◆999◆, ◆tr5◆, ◆kil8◆, ◆123◆**

inIntQueue (Front -> Rear): 3, 6, 17, 8, 31, 6, 8, 9

outStrStack (Top -> Bottom) :**◆kil8◆, ◆123◆, ◆tr550◆, ◆999◆**

- In the above example, *stackData* is ◆999◆, *data1* is 3 and *data2* is 6. The *sum* of *data1* and *data2* i.e. 3+6 is 9. Since it is present in ◆999◆, the *stackData* is added to **outStrStack**.
  Hence, **outStrStack** (Top -> Bottom) would be : **◆999◆**
- Now, the next *stackData* is ◆tr5◆, *data1* is 17 and *data2* is 8. The sum of *data1* and *data2* i.e. 17+8 is 25. Since, the sum◆s last digit 5 is present in ◆tr5◆, the *stackData* is concatenated with twice the sum i.e. ◆tr550◆ is added to **outStrStack.**
  Hence, **outStrStack** (Top -> Bottom) would be: **◆tr550◆**, ◆999◆
- Now, the next *stackData* is ◆kil8◆, *data1* is 31 and *data2* is 6. The sum of *data1* and *data2* i.e. 31+6 is 37. Since, its sum and last digit is not present in ◆kil8◆, the *stackData* is added to temporary data structure.
  Hence, temporary data structure would be: **◆kil8◆**
- Now, the next *stackData* is ◆123◆, *data1* is 8 and *data2* is 9. The sum of *data1* and *data2* i.e. 8+9 s 17. Since, its sum and last digit is not present in ◆123◆, the *stackData* is added to temporary data structure.
  Hence, temporary data structure would be: ◆kil8◆, **◆123◆**
- Empty the contents of temporary data structure to **outStrStack** (Top -> Bottom) such that they appear in the reverse order of their occurrence in temporary data structure.
  Hence, **outStrStack** (Top -> Bottom) would be: **◆kil8◆, ◆123◆, ◆tr550◆, ◆999◆**

Sample input and output:

inStrStack (Top → Bottom)	**inIntQueue** (Front → Rear)	**outStrStack** (Top → Bottom)
"hum8", "404"	4, 4, 11, 23	"40468", "hum8"
"67", "kiku", "800"	8, 9, 2, 6, 14, 23	"kiku", "800", "6734"
"39hj", "e55g", "v156", "duke", "100"	4, 5, 2, 3, 6, 6, 15, 6, 25, 5	"v156", "duke", "10060", "e55g", "39hj"