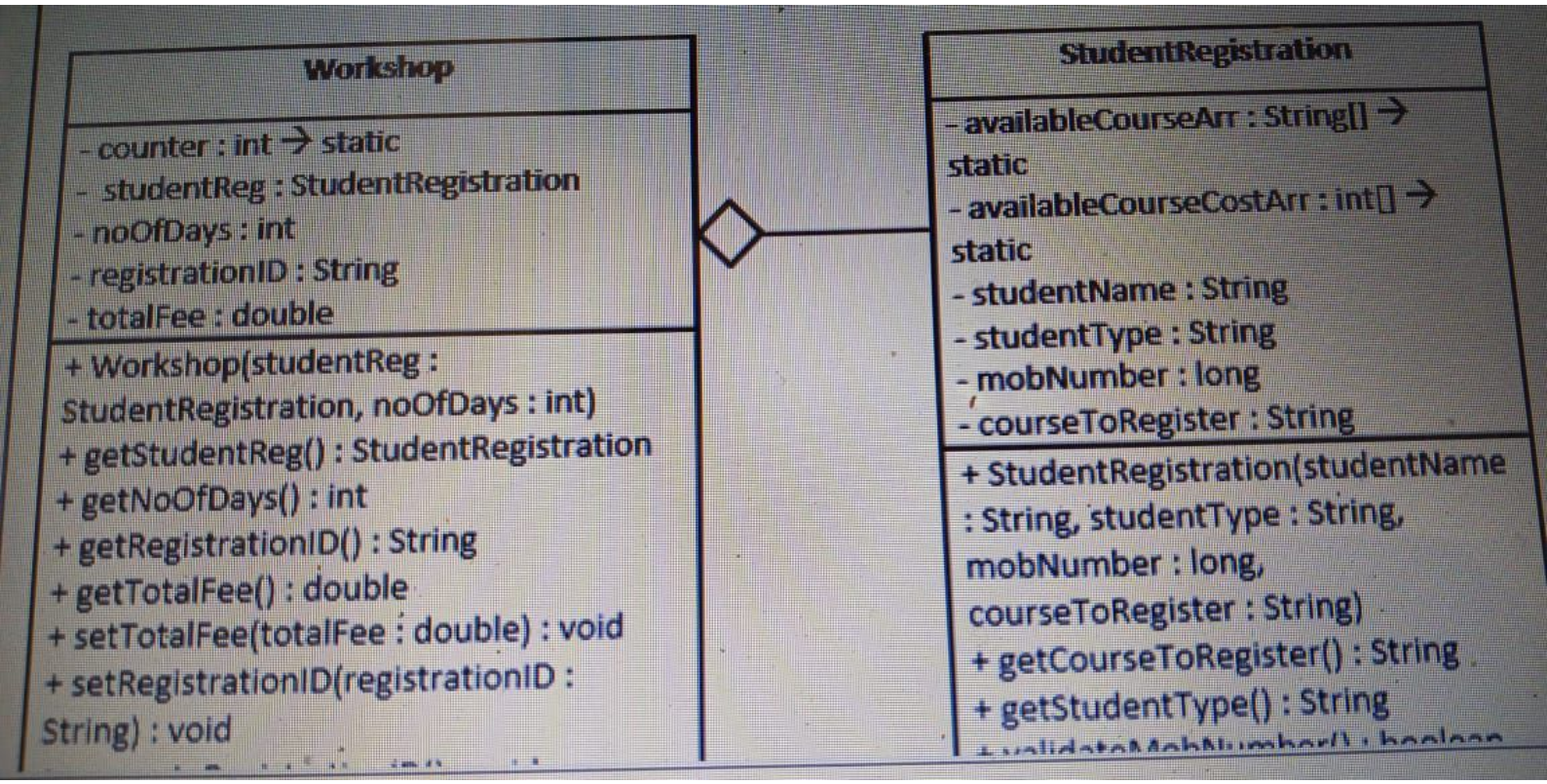
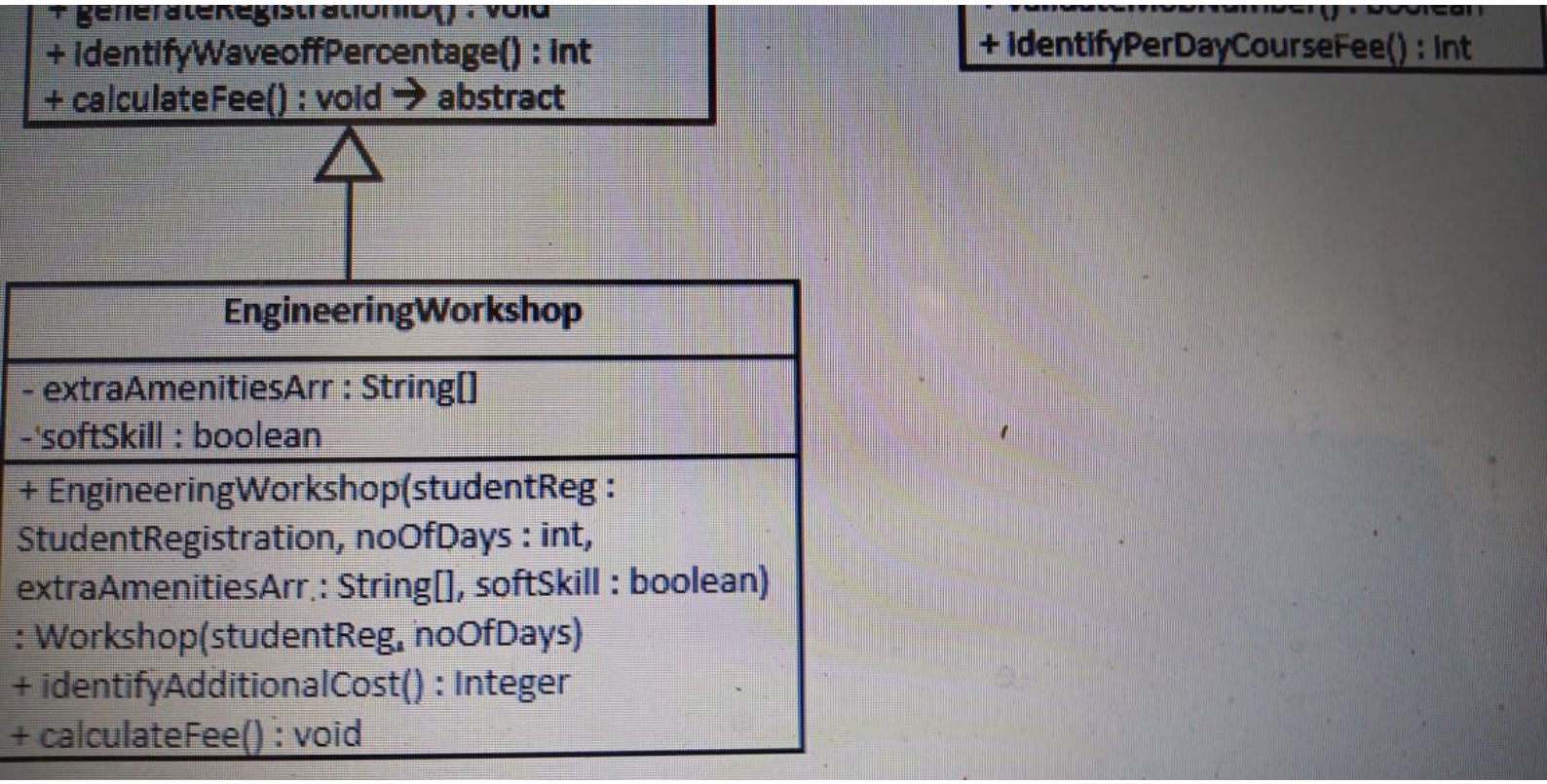

Workshop
- counter : int → static
- studentReg : StudentRegistration
- noOfDays : int
- registrationID : String
- totalFee : double
+ Workshop(studentReg : StudentRegistration, noOfDays : int)
+ getStudentReg() : StudentRegistration
+ getNoOfDays() : int
+ getRegistrationID() : String
+ getTotalFee() : double
+ setTotalFee(totalFee : double) : void
+ setRegistrationID(registrationID : String) : void
StudentRegistration
- availableCourseArr : String[] → static
- availableCourseCostArr : int[] → static
- studentName : String
- studentType : String
- mobNumber : long
- courseToRegister : String
+ StudentRegistration(studentName : String, studentType : String, mobNumber : long, courseToRegister : String)
+ getCourseToRegister() : String
+ getStudentType() : String


+ identifyWaveoffPercentage() : int
+ calculateFee() : void → abstract
+ identifyPerDayCourseFee() : int
EngineeringWorkshop
- extraAmenitiesArr : String[]
- softSkill : boolean
+ EngineeringWorkshop(studentReg : StudentRegistration, noOfDays : int, extraAmenitiesArr : String[], softSkill : boolean) : Workshop(studentReg, noOfDays)
+ identifyAdditionalCost() : Integer
+ calculateFee() : void

Note:

- "AI" denotes "Artificial Intelligence", "ML" denotes "Machine Learning" and "CG" denotes "Compu
- 'This array is supplied and hence, no need to code
- Do not change the CASE of the elements in the array

availableCourseCostArr:

- It is a static array (int[]) which has *courseCost* (int) as its elements

StudentRegistration class:

availableCourseArr:

- It is a static array (String[]) which has *availableCourse* (String) as its elements
- The initial values of **availableCourseArr** are-

availableCourseArr	{"AI", "ML", "CG"}

Note:

availableCourseArr	{"AI", "ML", "CG"}

Note:

- "AI" denotes "Artificial Intelligence", "ML" denotes "Machine Learning" and "CG" denotes **"Computer Graphics"**
- This array is supplied and hence, no need to code
- Do not change the CASE of the elements in the array

availableCourseCostArr:

- It is a static array (int[]) which has *courseCost* (int) as its elements
- This array has one-to-one correspondence with the **availableCourseArr**

availableCourseCostArr:

- It is a static array (int[]) which has *courseCost* (int) as its elements
- This array has one-to-one correspondence with the **availableCourseArr**
- The initial values of **availableCourseCostArr** are-

availableCourseCostArr	{250, 350, 200}

Note:

- This array is supplied and hence, no need to code

identifyDerivedCourseCost():

- **This array is supplied and hence, no need to code**

identifyPerDayCourseFee():

- **This method identifies and returns the *feePerDay* (int) based on the courseToRegister (String)**
- **If the courseToRegister is present as one of the elements in availableCourseArr, set the *feePerDay* with the correspo**
- **Otherwise, set the *feePerDay* to -1**
- **Return *feePerDay***

Note: Perform **case-sensitive** string comparison

Example: If **courseToRegister** is "AI", then *feePerDay* would be **250** currency

Workshop class:

Example: If **courseToRegister** is "AI", then *feePerDay* would be **250** currency

Workshop class:

generateRegistrationID():

- This method auto-generates **registrationID** (String)
- The **registrationID** would be prefixed with **courseToRegister** followed by the auto-generated value starting from 100
- The auto-generated value would be incremented by one for next **registrationID**
- Use static variable **counter** appropriately to implement the auto-generation logic

Example: The first **registrationID** would be "**AI1001**" if the **courseToRegister** is "AI", the second would be "**ML1002**" if th

EngineeringWorkshop class:

EngineeringWorkshop class:

identifyAdditionalCost():

- This method calculates and returns the *additionalCost* (Integer) based on the **extraAmenitiesArr** (String[])
- If the **extraAmenitiesArr** is **null**, then set the *additionalCost* to zero (0)
- Otherwise, for each *element* of **extraAmenitiesArr**, identify the *cost*:
  - If the *element* is "video", then *cost* would be 1000 currency
  - If the *element* is "toolkit", then *cost* would be 1500 currency
  - Other than these values, the cost would be 0
  - Calculate the *additionalCost* by adding all the above identified *cost*

Note: Perform **case-insensitive** comparison

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

Problem Statement:

Description:

Write a Java program that accepts non-empty intStack (int Stack) and intQueue (int Queue) as input parameters and returns a

- **Consider the elements of intStack from top to bottom and the elements of intQueue from front to rear**
- **For each element of intStack and intQueue,**
  - **Find the *maximum* and *minimum* of the corresponding element of intStack and intQueue**
  - **Form a *number* which contains *maximum* value as the first part and *minimum* as the second part for each corresponding**

    Note: If *maximum* value is 19 and *minimum* value is 3, then the *number* would be 193
  - **The *number* formed is to be pushed into temporary data structure so that these *numbers* can be finally pushed in the r**

Type here to search

Description:

Write a Java program that accepts non-empty inIntStack (int Stack) and inIntQueue (int Queue) as input parameters and returns outIn

- **Consider the elements of inIntStack from top to bottom and the elements of inIntQueue from front to rear**
- **For each element of inIntStack and inIntQueue,**
  - **Find the *maximum* and *minimum* of the corresponding element of inIntStack and inIntQueue**
  - **Form a *number* which contains *maximum* value as the first part and *minimum* as the second part for each corresponding ele**

    Note: If *maximum* value is 19 and *minimum* value is 3, then the *number* would be 193
  - The *number* formed is to be pushed into temporary data structure so that these *numbers* can be finally pushed in the reve
- Add the remaining elements from **inIntQueue** and **inIntStack,** if any, to the **outIntStack** from bottom to top in the same order
- Add *number(s)* from temporary data structure in reverse order to **outIntStack** from bottom to top
- Return **outIntStack**

- Return **outIntStack**

Assumptions:

- **inIntStack** and **inIntQueue** would contain positive integers
- Corresponding elements of **inIntStack** and **inIntQueue** would not be same

Note: No need to validate the assumptions

Example:

inIntStack (Top → Bottom): 3, 18, 9, 21, 6

inIntQueue (Front → Rear): 19, 10, 33, 12, 14, 27

outIntStack (Top → Bottom): 193,1810,339,2112,146,27

- In the above example, the front element of **inIntQueue** (Front → Rear) is 19 and the corresponding top element of **inIn**

out of the *maximum* and *minimum* value is 193 which will be added to the temporary data structure

outIntStack (Top → Bottom): 193,1810,339,2112,146,27

- In the above example, the front element of **inIntQueue** (Front → Rear) is 19 and the corresponding top element of **inIntStack** (Top → Bottom) out of the *maximum* and *minimum* value is 193 which will be added to the temporary data structure.
- The second element of **inIntQueue** (Front → Rear) is 10 and the corresponding second top element of **inIntStack** (Top → Bottom) *maximum* and *minimum* value is 1810 which will be added to the temporary data structure.
- The third element of **inIntQueue** (Front → Rear) is 33 and the corresponding third top element of **inIntStack** (Top → Bottom) *maximum* and *minimum* value is 339 which will be added to the temporary data structure.
- The next element of **inIntQueue** (Front → Rear) is 12 and the corresponding element of **inIntStack** (Top → Bottom) is 21. The *m* *minimum* value is 2112 which will be added to the temporary data structure.
- The last element of **inIntQueue** (Front → Rear) is 14 and the corresponding last element of **inIntStack** (Top → Bottom) is 6. The *minimum* value is 146 which will be added to the temporary data structure.

minimum value is 146 which will be added to the temporary data structure.

- Now the temporary data structure has the following elements:

  193, 1810 339, 2112, 146

- The remaining element from **inIntStack** is 27 which will be added to the **outIntStack** from bottom to top in the same order as t

  Now the **outIntStack** (Top → Bottom) would be **27**

- Finally, add *number(s)* from temporary data structure in reverse order to **outIntStack** from bottom to top.

  Hence the **outIntStack** (Top → Bottom) would be **193, 1810, 339, 2112, 146, 27**

Sample Input and Output:

(Top → Bottom) would be **193, 1810, 339, 2112, 146, 27**

inIntStack (Top → Bottom)	inIntQueue (Front → Rear)	outIntStack (Top → Bottom)
26, 21, 24, 17, 3, 5	6, 7, 6, 4, 14	266, 217, 246, 174, 143, 5
30, 11, 8, 12, 4	10, 3, 14, 16, 9, 123	3010, 113, 148, 1612, 94, 123
7, 19, 23, 5, 9, 16, 80, 52	12, 17, 2, 66, 43	127, 1917, 232, 665, 439, 52, 80, 1

WISH YOU ALL THE BEST