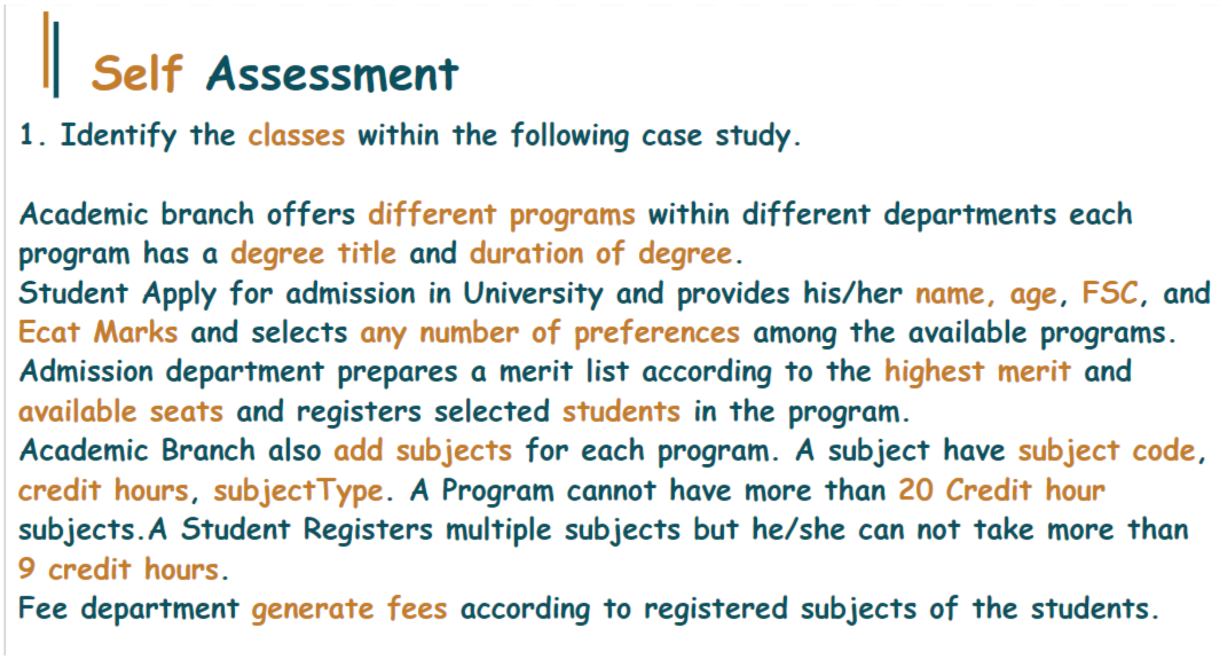
Department of Computer Science

University of Engineering and Technology, Lahore

Problem 01: University Admission Management System (Case Study with Separate BL, DL and UI along with Object Relational Mapping Technique)

Read the following question carefully.



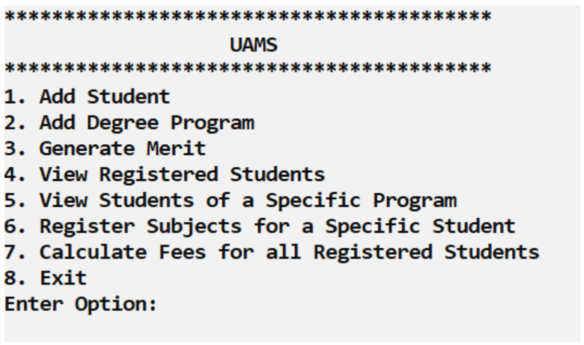
Self Assessment

1. Identify the classes within the following case study.

Academic branch offers different programs within different departments each program has a degree title and duration of degree.
Student Apply for admission in University and provides his/her name, age, FSC, and Ecat Marks and selects any number of preferences among the available programs.
Admission department prepares a merit list according to the highest merit and available seats and registers selected students in the program.
Academic Branch also add subjects for each program. A subject have subject code, credit hours, subjectType. A Program cannot have more than 20 Credit hour subjects.A Student Registers multiple subjects but he/she can not take more than 9 credit hours.
Fee department generate fees according to registered subjects of the students.

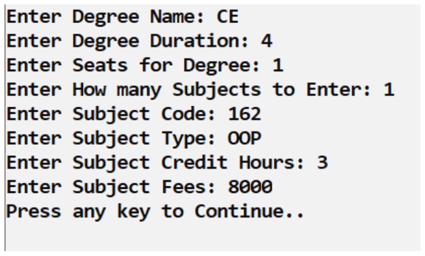
Wireframes of the UAMS:

1. Main Menu



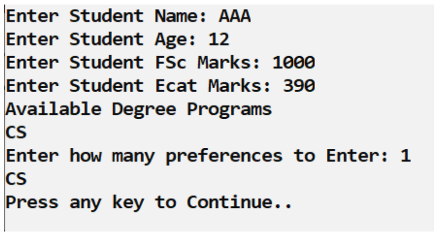
```
*****************************************
                  UAMS
*****************************************
1. Add Student
2. Add Degree Program
3. Generate Merit
4. View Registered Students
5. View Students of a Specific Program
6. Register Subjects for a Specific Student
7. Calculate Fees for all Registered Students
8. Exit
Enter Option:
```

2. Option 2: Degree Program



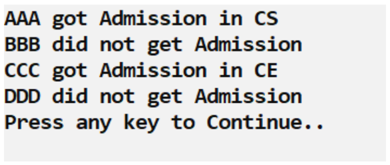
```
Enter Degree Name: CE
Enter Degree Duration: 4
Enter Seats for Degree: 1
Enter How many Subjects to Enter: 1
Enter Subject Code: 162
Enter Subject Type: OOP
Enter Subject Credit Hours: 3
Enter Subject Fees: 8000
Press any key to Continue..
```

3. Option 1: Add Student



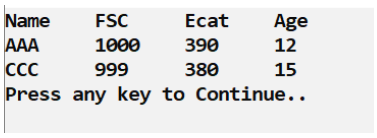
```
Enter Student Name: AAA
Enter Student Age: 12
Enter Student FSc Marks: 1000
Enter Student Ecat Marks: 390
Available Degree Programs
CS
Enter how many preferences to Enter: 1
CS
Press any key to Continue..
```

4. Option 3: Generate Merit



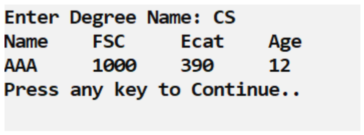
```
AAA got Admission in CS
BBB did not get Admission
CCC got Admission in CE
DDD did not get Admission
Press any key to Continue..
```

5. Option 4: Registered Students



```
Name    FSC     Ecat    Age
AAA     1000    390     12
CCC     999     380     15
Press any key to Continue..
```

6. Option 5: Specific Degree

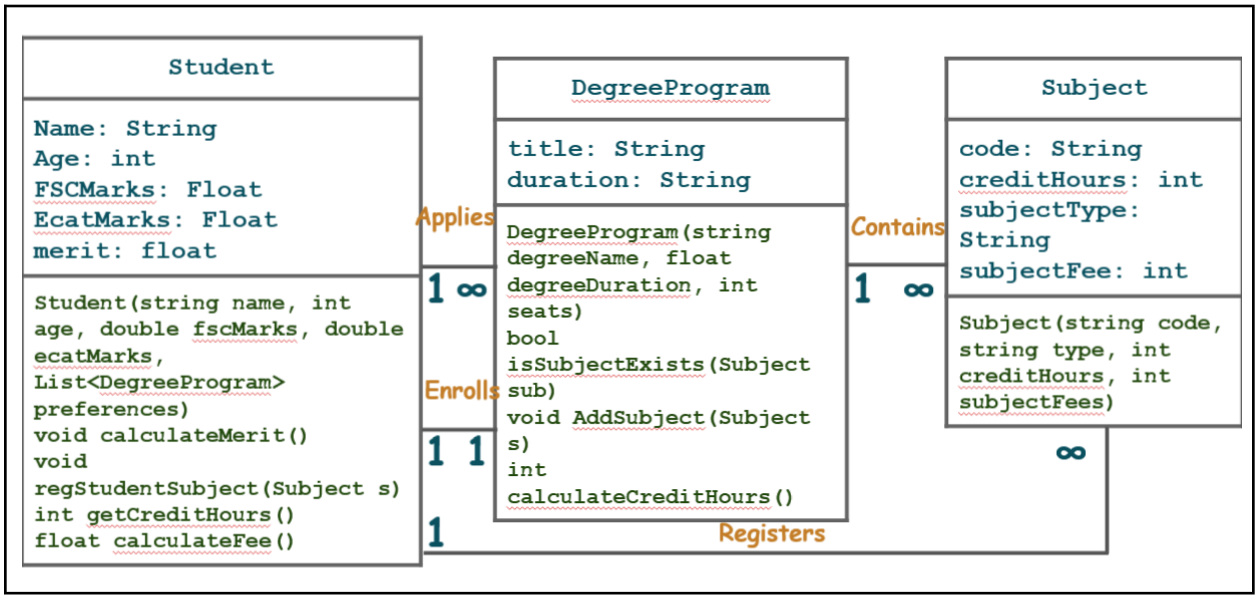


```
Enter Degree Name: CS
Name    FSC     Ecat    Age
AAA     1000    390     12
Press any key to Continue..
```

7. Option 6: Register Subjects
   Ask the Student name and then ask for the subject code. If the conditions are satisfied then the student's subject should be registered.

8. Option 7: Generate Fee
   Fees should be generated for all the registered students

Class Diagram with the member functions


Student
Name: String
Age: int
FSCMarks: Float
EcatMarks: Float
merit: float
Student(string name, int age, double fscMarks, double ecatMarks, List<DegreeProgram> preferences)
void calculateMerit()
void regStudentSubject(Subject s)
int getCreditHours()
float calculateFee()
Applies
1
∞
Enrolls
1
1
1
DegreeProgram
title: String
duration: String
DegreeProgram(string degreeName, float degreeDuration, int seats)
bool isSubjectExists(Subject sub)
void AddSubject(Subject s)
int calculateCreditHours()
Registers
Contains
1
∞
Subject
code: String
creditHours: int
subjectType: String
subjectFee: int
Subject(string code, string type, int creditHours, int subjectFees)
∞

Following are the Tasks related to this problem.

To Do Tasks:

1. Create a separate BL class for Subject, DL class for storing the list of subjects, and UI class for handling all the input output functions related to subjects.
2. Create a separate BL class for DegreeProgram, DL class for storing the list of DegreePrograms, and UI class for handling all the input output functions related to DegreePrograms.
3. Create a separate BL class for Student, DL class for storing the list of students, and UI class for handling all the input output functions related to Students.
4. If there are some UI functions that do not go into the UI of Subject, DegreeProgram or Student then add a separate UI class with name **ConsoleUtility.**
5. **Read the data from the files and also store the data in the files by following object relational mapping technique.**
6. Write the Driver Code (main function) for implementing the menu driven program.

Problem 02:

Miss Client wants to develop a software system for her departmental store. She wants this system to have the following functionalities.

As an Admin, she can

- Add Products.
- View All Products.
- Find Product with Highest Unit Price.
- View Sales Tax of All Products.
- Products to be Ordered. (less than threshold)

Following is the information that is required to save for the product.

Name of Product. Product Category. Product Price. Available Stock Quantity. Minimum Stock threshold Quantity after which the owner wants to order the product.

On All Grocery type of products, the sales tax is 10%, on all fruit types the tax is 5% and if there is any other type the tax is 15%

Customer information includes username, password, email, Address and Contact Number.

She also wants that

1. The Customers to view all the products
2. Customers can buy the products (When a customer buy a product then its quantity should decrease from the stock)
3. Generate invoice (While calculating the price of the products that the customer has bought, sales tax should be applied.)

Make 3 classes

1. Product
2. Customer
3. Admin (MUser) that we have previously developed with file handling

Menus on the Console.

1. SignIn
2. SignUp
3. Exit

If the user enters 1 then

Enter Username: AAA
Enter Password: 111

(if the user is valid and admin then show the admin menu)

1. Add Product.
2. View All Products.
3. Find Product with Highest Unit Price.
4. View Sales Tax of All Products.
5. Products to be Ordered. (less than threshold)
6. View Profile (Username, Password)
7. Exit

(if the user is valid and customer then show the customer menu)

1. View all the products
2. Buy the products
3. Generate invoice
4. View Profile (Username, Password, Email, Address and Contact Number)
5. Exit

Implementation Notes:

1. Make BL, DL and UI Class for Admin.
2. Make BL, DL and UI Class for Customer.
3. Make BL, DL and UI Class for Product.
4. If there are some UI functions that do not go into the UI of Admin, Customer or Product then add a separate UI class with name **ConsoleUtility.**
5. **Read the data from the files and also store the data in the files by following object relational mapping technique.**

Problem # 03:

Create a Class **MenuItem,** which has three instances

1. **name:** name of the item
2. **type:** whether *food* or a *drink*
3. **price:** price of the item

Following menu needs to be added. This data is present in the txt file in comma separated format. You must read the file and load the data from the file.

Name	**Type**	**Price**
"orange juice"	Drink	60
"lemonade"	Drink	50
"cranberry juice"	Drink	100
"pineapple juice"	Drink	100
"lemon iced tea"	Drink	120
"vanilla chai latte"	Drink	150
"hot chocolate"	Drink	140
"iced coffee"	Drink	140
"tuna sandwich"	Food	300
"ham and cheese sandwich"	Food	300
"egg sandwich"	Food	200
"steak"	Food	900
"hamburger"	Food	600
"cinnamon roll"	Food	150

Write a class called **CoffeeShop**, which has three instance variables:

1. **name** : a string (basically, of the shop)
2. **menu** : an list of items (of object type), with each item containing the item (name of the item), type (whether *food* or a *drink*) and price.
3. **orders** : an empty list of string type.

And a parameterized constructor which takes the name of the CoffeeShop as a parameter.

and eight methods:

1. **addMenuItem:** adds the menu item in the list of menu
2. **addOrder:** adds the name of the item to the end of the orders list if it exists on the menu. Otherwise, return "This item is currently unavailable!"
3. **fulfillOrder:** if the orders list is not empty, return "The {item} is ready!" and make the list empty. If the order list is empty, return "All orders have been fulfilled!"
4. **listOrders:** returns the list of orders taken, otherwise null.
5. **dueAmount:** returns the total amount due for the orders taken.
6. **cheapestItem:** returns the name of the cheapest item on the menu.
7. **drinksOnly:** returns only the *item* names of *type* drink from the menu.
8. **foodOnly:** returns only the *item* names of *type* food from the menu.

IMPORTANT: Orders are fulfilled in a FIFO (first-in, first-out) order.

Following are the Coffee Shops in the neighborhood with the menu items stored in the file in this format.

Name of the coffee shop, list of menu items separated by semicolon.

1. Tesha,orange juice;lemonade;cranberry juice;pineapple juice;ham and cheese sandwich;egg sandwich;steak;hamburger;cinnamon roll;lemon iced tea;vanilla chai latte;hot chocolate;iced coffee
2. One Cup,lemon iced tea;vanilla chai latte;hot chocolate;iced coffee;cranberry juice;pineapple juice;tuna sandwich;ham and cheese sandwich
3. MorningTime,tuna sandwich;ham and cheese sandwich;egg sandwich;steak;hamburger;cinnamon roll;orange juice;lemonade

Driver Program: (Following options should be shown to the user)

Welcome to the Coffee Shops Management System

1. Add a Menu item
2. Add a Coffee Shop
3. View the Cheapest Item in the menu from a specific coffee shop

4. View the Drink’s Menu from a specific coffee shop
5. View the Food’s Menu from a specific coffee shop
6. Add Order in a specific coffee shop
7. Fulfill the Order in a specific coffee shop
8. View the Orders’s List from a specific coffee shop
9. Total Payable Amount from a specific coffee shop
10. Exit

Test Cases:

// After creating an object of CoffeeShop and initializing its attributes

tcs.addOrder("hot cocoa") ➞ "This item is currently unavailable!"
// Tesha's coffee shop does not sell hot cocoa
tcs.addOrder("iced tea") ➞ "This item is currently unavailable!"
// specifying the variant of "iced tea" will help the process

tcs.addOrder("cinnamon roll") ➞ "Order added!"
tcs.addOrder("iced coffee") ➞ "Order added!"
tcs.listOrders ➞ ["cinnamon roll", "iced coffee"]
// the list of all the items in the current order

tcs.dueAmount() ➞ 290

tcs.fulfillOrder() ➞ "The cinnamon roll is ready!"
tcs.fulfillOrder() ➞ "The iced coffee is ready!"
tcs.fulfillOrder() ➞ "All orders have been fulfilled!"
// all orders have been presumably served

tcs.listOrders() ➞ []
// an empty list is returned if all orders have been exhausted

tcs.dueAmount() ➞ 0.0
// no new orders taken, expect a zero payable

tcs.cheapestItem() ➞ "lemonade"
tcs.drinksOnly() ➞ ["orange juice", "lemonade", "cranberry juice", "pineapple juice", "lemon iced tea", "vanilla chai latte", "hot chocolate", "iced coffee"]
tcs.foodOnly() ➞ ["tuna sandwich", "ham and cheese sandwich", "bacon and egg", "steak", "hamburger", "cinnamon roll"]

Problem # 04:

In this problem, you have to create a class called **Point,** which models a 2D point with x and y coordinates.
It contains:

- Two instance variables x (int) and y (int).
- A default (or "no-argument" or "no-arg") constructor that constructs a point at the default location of (0, 0).
- A parameterized constructor that constructs a point with the given x and y coordinates.
- Getter and setter for the instance variables x and y.
- A method setXY() to set both x and y.

Next, create a class named **Boundary**.

It contains:

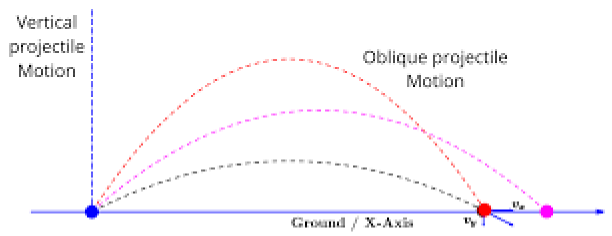
- Four attributes of Point type
  - TopLeft
  - TopRight
  - BottomLeft
  - BottomRight
- A default (or "no-argument" or "no-arg") constructor that constructs a boundary with default location of TopLeft(0, 0), TopRight(0,90), BottomLeft(90,0) and BottomRight(90,90).
- A parameterized constructor that constructs a boundary with the given TopLeft, TopRight, BottomLeft and BottomRight points.

Next, create a class named **GameObject.**

It contains 4 attributes:

- One attribute **Shape** (2D Array char type).
- A **StartingPoint** (Point type).
- A **Premises** (Boundary type).
- A **Direction** (String type).

LeftToRight	Starts from the starting point and keeps on moving towards the right and stops at the extreme right boundary point
RightToLeft	Starts from the starting point and keeps on moving towards the left and stops at the extreme left boundary point
Patrol	Starts from the starting point and keeps on moving towards the left and stops at the extreme left boundary point and reverses the direction.
Projectile	Starts from the starting point and move 5 points towards the top right and then 2 steps towards the right and then 4 steps towards bottom right.

	 Vertical projectile Motion Oblique projectile Motion Ground / X-Axis
Diagonal	Starts from the starting point and moves towards the bottom right and stops at the extreme right boundary point.

- A default constructor that initializes
  - Shape (1x3 line "---")
  - StartingPoint (constructs a point at the default location of (0, 0))
  - Premises (constructs a boundary with default location of TopLeft(0, 0), TopRight(0,90), BottomLeft(90,0) and BottomRight(90,90))
  - Direction ("LeftToRight")

- A parameterized constructor that takes
  - Shape, StartingPoint
  - Whereas **Premises** (constructs a boundary with default location of TopLeft(0, 0), TopRight(0,90), BottomLeft(90,0) and BottomRight(90,90)) and **Direction** with default direction ("LeftToRight")

- A parameterized constructor that takes
  - Shape
  - StartingPoint
  - Premises
  - Direction

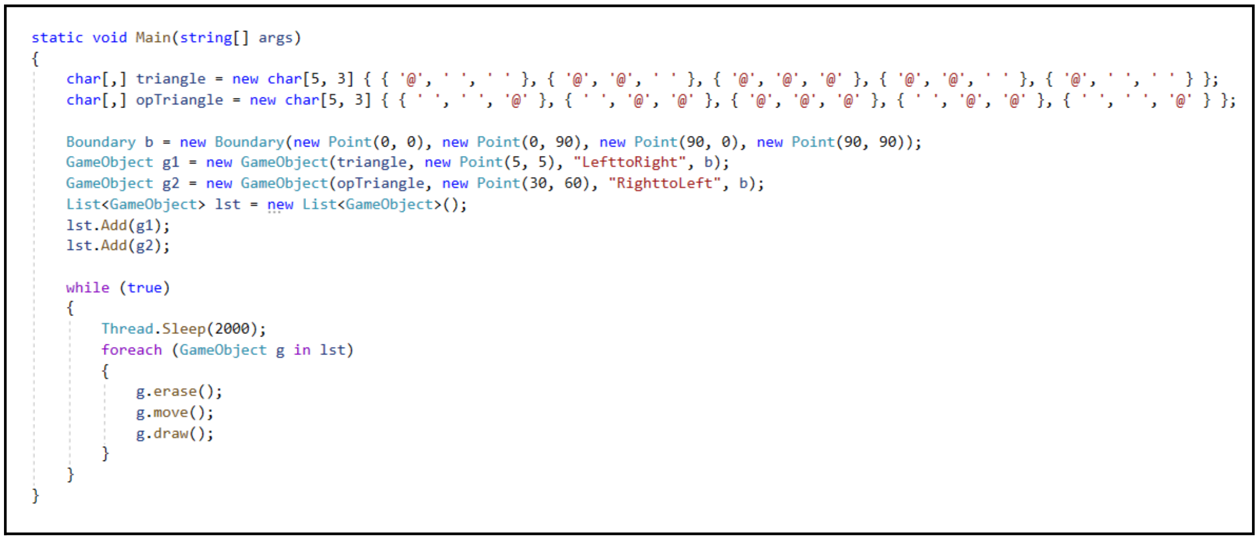
- It will also contain the following methods
  - **Move:** if the direction is "LeftToRight", the shape will move one step according to its direction. For example, if the direction is from left to right it will move the game object one step toward right.
  - **Erase:** When called, this method will erase the shape on the console.
  - **Draw:** When called, this method will draw the shape on the console.

- Following are some examples of the shapes you can draw (Use your creativity to come up with different shapes)

```
 ¤¤¤¤
¤¤¤¤¤¤  ¤
¤¤¤¤¤¤ ¤¤¤
¤    ¤
```

```
  ☼☼
 ☼☼☼☼
☼☼☼☼☼☼
 ☼☼☼☼
  ☼☼
 ☼☼☼☼
```

Demo Driver Program:



```
static void Main(string[] args)
{
    char[,] triangle = new char[5, 3] { { '@', ' ', ' ' }, { '@', '@', ' ' }, { '@', '@', '@' }, { '@', '@', ' ' }, { '@', ' ', ' ' } };
    char[,] opTriangle = new char[5, 3] { { ' ', ' ', '@' }, { ' ', '@', '@' }, { '@', '@', '@' }, { ' ', '@', '@' }, { ' ', ' ', '@' } };

    Boundary b = new Boundary(new Point(0, 0), new Point(0, 90), new Point(90, 0), new Point(90, 90));
    GameObject g1 = new GameObject(triangle, new Point(5, 5), "LefttoRight", b);
    GameObject g2 = new GameObject(opTriangle, new Point(30, 60), "RighttoLeft", b);
    List<GameObject> lst = new List<GameObject>();
    lst.Add(g1);
    lst.Add(g2);

    while (true)
    {
        Thread.Sleep(2000);
        foreach (GameObject g in lst)
        {
            g.erase();
            g.move();
            g.draw();
        }
    }
}
```

NOTE: This Demo Driver Program is written for your understanding. You must make the lists and console input and output functions in their respective DL and UI files respectively

Problem 05: Business Application

Identify the Classes from your Business Application and Draw the Class Diagram and Implement your Business Application with Separate Classes in BL,DL and UI Folder along with File Handling using ORM Technique.

Problem 06: Game Project

Identify the Classes from your Project and Draw the Class Diagram and Implement your Game with Separate Classes in BL, DL and UI Folder along with File Handling using ORM Technique.