## Introduction

After a week of rigorous coding, Welcome back!

**You have learned all about the Classes, Constructors, and member functions in the previous lab manuals. Let's move on to the next, new, and interesting concepts.**

Students, In Object-Oriented Programming, the Class is a combination of data members and member functions. In this Lab, we will learn about including **multiple classes** into our program to achieve the object's oriented philosophy.

**Let's do some coding.**

Students, Recall this question from the class that you attempted and developed CRC cards.

| **Problem Statement 01:** |
| --- |
| **Identify Classes, Data and Behaviour**<br><br>Now, we need to identify the classes, data and behaviour from real time scenarios.<br>Suppose you have to implement a software for a College or University Student Record Management System. University should maintain the information about student's full name, roll number, cgpa, matric marks, fsc marks, ecat marks, current semester, fees, home town, whether day scholar or hostelite and whether availing a scholarship or not.Merit of student can be calculated by adding 60% of Fsc Marks and 40% of ECat marks. Students can check their scholarship status: A student is eligible for scholarship if her merit is greater than 80% and she is hostelite. |
| **Solution** |

**– CRC Card**

| student |
| --- |
| Name: string<br>rollNumber: int<br>cGPA: float<br>matricMarks: int<br>fscMarks: int<br>homeTown: string<br>isHostelite: bool<br>isTakingScholarship: bool |
| student()<br>student(parametrized)<br>claculateMerit(): float<br>isEligibleforScholarship(float meritPercentage): bool |

**Self Assessment Task:** Implement the Class Student by using the CRC card that you developed in the class and use the driver program to test your member functions.

**Congratulations !!!!!!!!!!!!!!!! you have implemented your first class by using the CRC card.**

**Problem Statement 02:**

## Identify Classes, Data and Behaviour

Suppose you have to implement a software for a Library Management System. The library system must keep track of the books whether the book is available or borrowed. Books contains the title, list of chapters, number of pages, price, name of the author.
A person can see is the book available or borrowed. He can also see the bookmark is on which page number. Also he can see the name of a specific chapter.
What will be class for Book, its Data Members, and Behaviours.

**Solution**

**CRC Card**

| Book |
|---|
| author: string<br>pages: int<br>Chapters: List<string><br>bookMark: int<br>price: int |
| Book()<br>Book(parametrized)<br>getChapter(chapterNumber: int): string<br>getBookMark(): int<br>setBookMark(pageNumber: int): void<br>getBookPrice(): int<br>setBookPrice(): void |

**Self Assessment Task:** Implement the Class Student by using the CRC card that you developed in the class and use the driver program to test your member functions.

**Great Work Students !!! You guys are doing an excellent job. Well done. Take a two-minute break. Well deserved.**

**Problem Statement 03:**

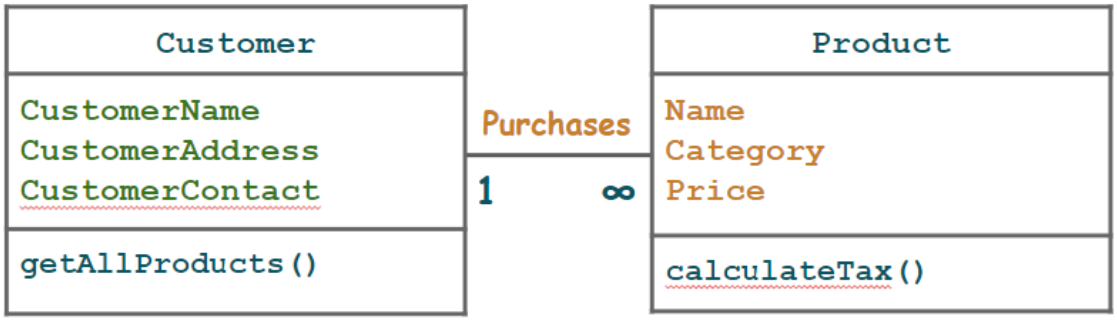
## Problem Scenario

Let's suppose a store needs to save the information of the product and the customer who has bought the product.

Store wants to calculate the total purchases of a customer. Store wants to calculate the tax on the purchased products as well.

**Solution**

**CRC Card**




We need to define **two separate classes** in order to achieve the object-oriented philosophy. So same kinds of data shall reside in the same place.

```
class Customer
{
    public string CustomerName;
    public string CustomerAddress;
    public string CustomerContact;
    public List<Product> products = new List<Product>();
    public List<Product> getAllProducts()
    {
        return products;
    }
    public addProduct(Product p)
    {
        products.Add(p);
    }
}
```

This code creates the **Customer** class **Data Members**, **Constructor**, and **Member Function** for adding products to the list.

```
class Product
{
    public string name;
    public string category;
    public int price;
    public float calculateTax()
    {
        // Implementation
    }
}
```

This code creates the **Product** class **Data Members** and **Member Function** for calculating tax.

**Self Assessment Task 01:** Implement this program by using the CRC card that we have developed and then use the driver program to test your member functions.
**Self Assessment Task 02:** Implement the member function in the appropriate class for calculating the tax of all the products that a customer has purchased.

## Challenge # 01:

**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.

**Try out yourself.**

**Don't worry.**

**There is a solution on the next page.**

**Identification of Classes**

By looking at the above-mentioned self-assessment you can extract the following possible class-like structures from the given statement.

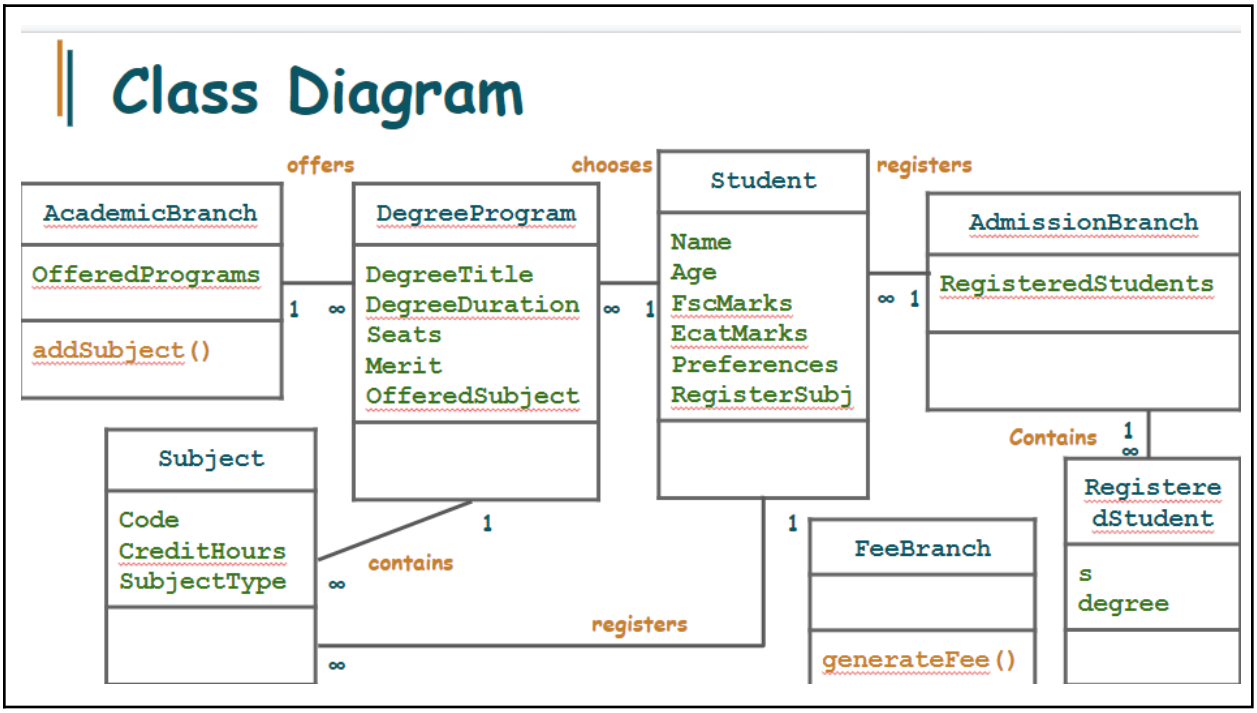
- Student Class
- Subject Class
- Academic Branch
- Admission Branch
- Registered Students
- Degree Program
- Fee Department

**Note**: Create a separate class in the same BL(Business Logic) folder of your program.

**Now Try to Build the Class Diagram/Domain Model of these classes.**

**Don't Worry. There is a solution ahead. First Try out yourself.**

## Class Realization




**Let's Start the fun coding.**

## University Admission Management System (Through OOP)

Now, that you have identified the classes in your program, it is time to start the coding.

**Solution:**

| Sr. # | Action | Description |
|---|---|---|
| 1. | `class Student`<br>`{`<br>`public string name;`<br>`public int age;`<br>`public double fscMarks;`<br>`public double ecatMarks;`<br>`public List<DegreeProgram> preferences;`<br>`public List<Subject> regSubject;`<br>`public bool gotAdmission;`<br><br>`public Student(string name, int age, double fscMarks, double ecatMarks)`<br>`{`<br>`this.name = name;`<br>`this.age = age;`<br>`this.fscMarks = fscMarks;`<br>`this.ecatMarks = ecatMarks;`<br>`preferences = new List<DegreeProgram>();`<br>`regSubject = new List<Subject>();`<br>`gotAdmission = false;`<br>`}` | Creates a **Student** Class with one **Parameterized** Constructor.<br><br>**Important Note**: Each student shall need a degree program preferences list and one registered subjects list. Therefore, we need to implement these classes too. |
| 1(a) | `class Subject`<br>`{`<br>`public string code;`<br>`public string type;`<br>`public int creditHours;`<br><br>`public Subject(string code, string type, int creditHours)`<br>`{`<br>`this.code = code;`<br>`this.type = type;`<br>`this.creditHours = creditHours;`<br>`}`<br>`}` | In this code, we will create the **Subject** class. The attached code<br>• Implements the **Subject class**<br>• Provides Parameterized Constructor where the user must provide **subject code**, **subject type**, and **credit hours** before creating a class object. |

| | | |
|---|---|---|
| 1(b) | <br>`class DegreeProgram`<br>`{`<br>`public string degreeName;`<br>`public double degreeDuration;`<br>`public List<Subject> subjects;`<br>`public double merit;`<br>`public int seats;`<br><br>`public DegreeProgram(string degreeName, double degreeDuration)`<br>`{`<br>`this.degreeName = degreeName;`<br>`this.degreeDuration = degreeDuration;`<br>`subjects = new List<Subject>();`<br>`}` | In this code, we will create the degree program class. The attached code<br>● Implements the **DegreeProgram**<br>● Provides Parameterized Constructor where the user must provide **the degree name**, and **degree duration** before creating a class object. |
| 1(c) | <br>`public void AddSeatsAndMerit(int seats, double merit)`<br>`{`<br>`this.seats = seats;`<br>`this.merit = merit;`<br>`}`<br><br>`public int calculateCreditHours()`<br>`{`<br>`int count = 0;`<br>`for (int x = 0; x < subjects.Count; x++)`<br>`{`<br>`count = count + subjects[x].creditHours;`<br>`}`<br>`return count;`<br>`}`<br><br>`public void AddSubject(Subject s)`<br>`{`<br>`int creditHours = calculateCreditHours();`<br>`if(creditHours + s.creditHours <= 20)`<br>`{`<br>`subjects.Add(s);`<br>`}`<br>`else`<br>`{`<br>`Console.WriteLine("20 credit hour limit exceeded");`<br>`}`<br>`}` | This code<br>● Includes member functions in the degree program class for adding **seatandMerit** and **adding Subjects** and **caculateCreditHours().** |

| | | |
|---|---|---|
| 2. | `class FeeDepartment`<br>`{`<br>`public float calculateFee(Student s)`<br>`{`<br>`int stCH = s.getCreditHours();`<br>`return stCH * 2000;`<br>`}`<br>`}` | This code creates a class **fee department** that will include only a single function that will calculate the fee of the given **student object**. |
| 3. | `public double calculateMerit()`<br>`{`<br>`double merit = (((fscMarks / 1100) * 0.45F) + ((ecatMarks / 400) * 0.55F)) * 100;`<br>`return merit;`<br>`}`<br>`public int getCreditHours()`<br>`{`<br>`int count = 0;`<br>`for (int x = 0; x < regSubject.Count; x++)`<br>`{`<br>`count = count + regSubject[x].creditHours;`<br>`}`<br>`return count;`<br>`}`<br>`public void regStudentSubject(Subject s)`<br>`{`<br>`int stCH = getCreditHours();`<br>`if (gotAdmission && stCH + s.creditHours <= 9)`<br>`{`<br>`for (int x = 0; x < preferences[0].subjects.Count; x++)`<br>`{`<br>`if (s.code == preferences[0].subjects[x].code)`<br>`{`<br>`regSubject.Add(s);`<br>`}`<br>`}`<br>`}`<br>`else`<br>`{`<br>`Console.WriteLine("A student cannot have more than 9 CH or Wrong Subject");`<br>`}`<br>`}` | Complete the **Student Class** by including the member function for performing the following tasks.<br>• Merit Calculator<br>• CreditHour Calculator<br>• Registering Subjects for students |

Now, we need to implement the remaining classes including **Academics Branch, Registered Students,** and **Admission Branch**.

| | | |
|---|---|---|
| 4. | <br>`class AcademicBranch`<br>`{`<br>`public List<DegreeProgram> ProgramList = new List<DegreeProgram>();`<br><br>`public void addDegree(string degreeName, int duration, int seats, int merit)`<br>`{`<br>`DegreeProgram deg = new DegreeProgram(degreeName, duration);`<br>`deg.AddSeatsAndMerit(seats, merit);`<br>`ProgramList.Add(deg);`<br>`}`<br>`public void AddSubject(string degName, Subject s)`<br>`{`<br>`for (int x = 0; x < ProgramList.Count; x++)`<br>`{`<br>`if (ProgramList[x].degreeName == degName)`<br>`{`<br>`ProgramList[x].AddSubject(s);`<br>`}`<br>`}`<br>`}`<br>`}` | This code provides the class implementation of class **AcademicsBranch**. It includes the following<br>● A-List of **DegreeProgram** type<br>● Member functions for **adding degrees** and **adding subjects** individually |
| 5. | <br>`class AdmissionBranch`<br>`{`<br>`public List<RegisteredStudent> regStudents = new List<RegisteredStudent>();`<br>`public void CalculateAdmission(DegreeProgram dp, List<Student> studentsList)`<br>`{`<br>`for (int x = 0; x < studentsList.Count; x++)`<br>`{`<br>`double stuMerit = studentsList[x].calculateMerit();`<br>`if (dp.seats > 0 && stuMerit >= dp.merit && studentsList[x].gotAdmission==false)`<br>`{`<br>`RegisteredStudent rStudent = new RegisteredStudent(studentsList[x], dp);`<br>`studentsList[x].gotAdmission = true;`<br>`regStudents.Add(rStudent);`<br>`dp.seats--;`<br>`Console.WriteLine("Student registered");`<br>`}`<br>`else`<br>`{`<br>`Console.WriteLine("student can't be registered in this program");`<br>`}`<br>`}`<br>`}` | This code provides the class implementation of class **AdmissionBranch**. It includes the following<br>● A list of **Student** type<br>● Member functions for **CalcluateAdmission** for registering students. |
| 5(a). | <br>`class RegisteredStudent`<br>`{`<br>`public Student s;`<br>`public DegreeProgram degree;`<br><br>`public RegisteredStudent(Student s, DegreeProgram degree)`<br>`{`<br>`this.s = s;`<br>`this.degree = degree;`<br>`}`<br>`}` | This Code<br>● will create the class **registeredStudents** and implements a parameterized constructor of this class. |
| Let us now implement the **Main Driver Program** for this project. | | |

| | | |
|---|---|---|
| 6. | `static void Main(string[] args)`<br>`{`<br>`  AcademicBranch academicBranch = new AcademicBranch();`<br><br>`  academicBranch.addDegree("CS", 4, 1, 78);`<br>`  academicBranch.addDegree("CE", 4, 1, 80);`<br>`  Subject PF = new Subject("CS-101", "Programming Fundamentals", 3);`<br>`  Subject OOP = new Subject("CS-100", "Object Oriented Programming", 4);`<br><br>`  academicBranch.AddSubject("CS", PF);`<br>`  academicBranch.AddSubject("CE", OOP);` | Create **Degrees** and **Subjects** |
| 7. | `Student s1 = new Student("ali", 24, 1000, 330);`<br>`Console.WriteLine(s1.calculateMerit());`<br>`s1.preferences.Add(academicBranch.ProgramList[0]);`<br>`s1.preferences.Add(academicBranch.ProgramList[1]);`<br><br>`Student s2 = new Student("fatima", 21, 921, 179);`<br>`s2.preferences.Add(academicBranch.ProgramList[0]);`<br>`s2.preferences.Add(academicBranch.ProgramList[1]);`<br>`Console.WriteLine(s2.calculateMerit());` | Create **students** and set **preferences** |
| 8. | `List<Student> students = new List<Student>();`<br>`students.Add(s1);`<br>`students.Add(s2);` | Create a list of **Students** and add the **student-type objects** to the list. |
| 9. | `AdmissionBranch AdmissionDept = new AdmissionBranch();`<br>`AdmissionDept.CalculateAdmission(academicBranch.ProgramList[0], students);`<br>`AdmissionDept.CalculateAdmission(academicBranch.ProgramList[1], students);`<br><br>`for (int x = 0; x < AdmissionDept.regStudents.Count; x++)`<br>`{`<br>`  AdmissionDept.regStudents[x].s.regStudentSubject(academicBranch.ProgramList[0].subjects[0]);`<br>`  Console.WriteLine(AdmissionDept.regStudents[x].s.name);`<br>`}`<br>`FeeDepartment FeeDept = new FeeDepartment();`<br>`for (int x = 0; x < AdmissionDept.regStudents.Count; x++)`<br>`{`<br>`  Console.WriteLine(FeeDept.calculateFee(AdmissionDept.regStudents[x].s));`<br>`}` | Implement the rest of the driver program by **incorporating** the **remaining elements**. |
| 10. | `86.2840908088467`<br>`62.2897722626274`<br>`Student registered`<br>`student can't be registered in this program`<br>`student can't be registered in this program`<br>`student can't be registered in this program`<br>`ali`<br>`6000` | The **output** of the program. |

**You have made it through all that. Excellent work students !!!**
**You guys are successfully en route to be Kamyab Programmers.**

"No more Work for Today."

**Good Luck and Best Wishes !!**
**Happy Coding ahead :)**