Object Oriented Programming

Lab Manual 05

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 us learn how to sort a list by using the predefined function.

Item	Description
For linear data	**Syntax:** **ListName.sort();** **Working:** Sorts a string, int, or float type list

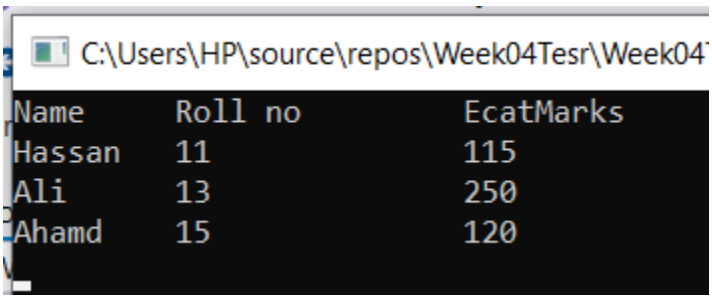
Code:	`List<int> integerList = new List<int>() { 1, 5, 4, 7, 2, 3, 6 };`
	`integerList.Sort();`
	`foreach (int i in integerList)`
	`  Console.Write(i + " ");`
	`Console.ReadKey();`

Solution:	C:\Users\HP\source\repos\Week04Tesr\Week04Tesr\bin\Debug\Week04Tesr.exe `1 2 3 4 5 6 7`
For Class data	**Syntax:** **newList = listName.OrderBy(o => o.classAttribtue).toList();** **Working:** Sorts a list in ascending order based on the given attribute value **Syntax:** **newList = listName.OrderByDescending(o => o.classAttribtue).toList();** **Working:**

	Sorts a list in descending order based on the given attribute value
Code:	

```
Student s1 = new Student("Ahamd", 15, 120);
Student s2 = new Student("Hassan", 11, 115);
Student s3 = new Student("Ali", 13, 250);
List<Student> studentList = new List<Student> (){ s1, s2, s3 };
List<Student> sortedList = studentList.OrderBy(o => o.rollno).ToList();
Console.WriteLine("Name \t Roll no \t EcatMarks");
foreach (Student s in sortedList)
{
    Console.WriteLine("{0} \t {1} \t \t {2}", s.name, s.rollno, s.ecatMarks);
}
Console.Read();
```

Solution:



Ascending Sorting of the list based on Roll Number

Self Assessment 01(a): Create a String and float type list and sort the list by using the sort() function.

Self Assessment 01(b): Create a class type list and sort it in descending order.

University Admission Management System

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.

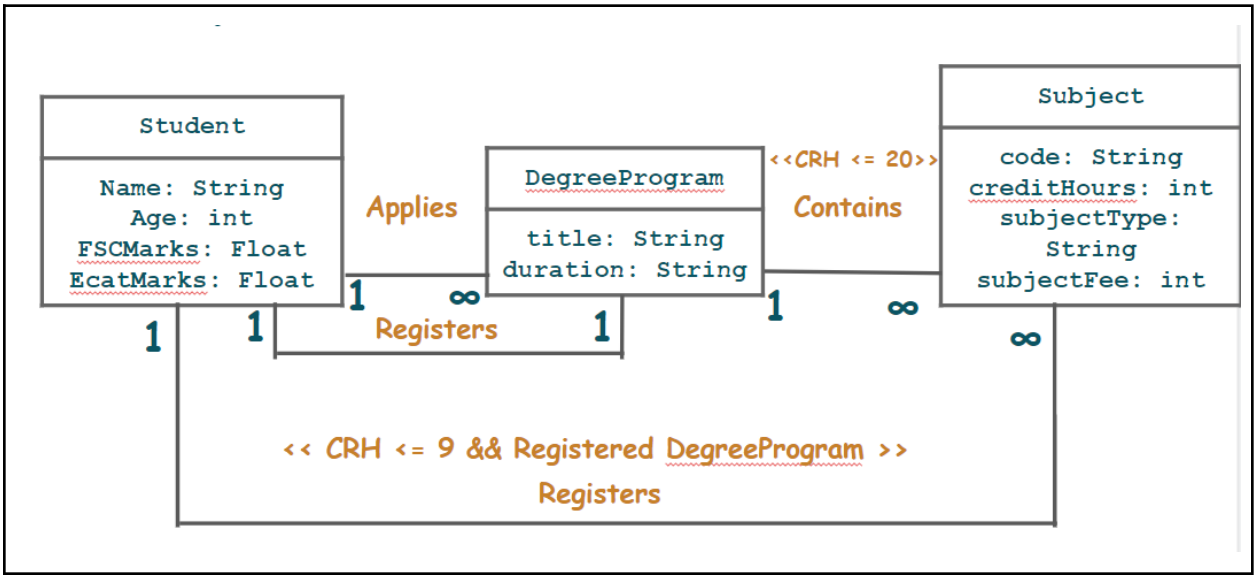
- Student Class
- Subject Class
- Program Class

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 Diagram without the member functions



Let's Start with fun coding.

University Admission Management System (Through OOP)

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

Solution:

Sr. #	Action	Description

1.	`class Student`
	`{`
	`public string name;`
	`public int age;`
	`public double fscMarks;`
	`public double ecatMarks;`
	`public double merit;`
	`public List<DegreeProgram> preferences;`
	`public List<Subject> regSubject;`
	`public DegreeProgram regDegree;`
	
	`public Student(string name, int age, double fscMarks, double ecatMarks, List<DegreeProgram> preferences)`
	`{`
	`this.name = name;`
	`this.age = age;`
	`this.fscMarks = fscMarks;`
	`this.ecatMarks = ecatMarks;`
	`this.preferences = preferences;`
	`regSubject = new List<Subject>();`
	`}`
		Creates a **Student** Class with one **Parameterized** Constructor.
		
		Important Note: Each student shall need a degree program preferences list and one registered subjects list and a selected Degree Program. These were determined through the relations between the Students Class and other Classes. Therefore, we need to include these attributes too.

1(a)	`class Subject`
	`{`
	`public string code;`
	`public string type;`
	`public int creditHours;`
	`public int subjectFees;`
	
	`public Subject(string code, string type, int creditHours, int subjectFees)`
	`{`
	`this.code = code;`
	`this.type = type;`
	`this.creditHours = creditHours;`
	`this.subjectFees = subjectFees;`
	`}`
	`}`
		In this code, we will create the **Subject** class. The attached code
		• Implements the **Subject class**
		• Provides Parameterized Constructor where the user must provide **subject code**, **subject type**, **subject fees,** and **credit hours** before creating a class object.

1(b)	
	`class DegreeProgram`
	`{`
	`public string degreeName;`
	`public float degreeDuration;`
	`public List<Subject> subjects;`
	`public int seats;`
	
	`public DegreeProgram(string degreeName, float degreeDuration, int seats)`
	`{`
	`this.degreeName = degreeName;`
	`this.degreeDuration = degreeDuration;`
	`this.seats = seats;`
	`subjects = new List<Subject>();`
	`}`
		In this code, we will create the **degree program** class. The attached code
		● Implements the **DegreeProgram**
		● Provides Parameterized Constructor where the user must provide **the degree name**, and **degree duration** before creating a class object.

1(c)	
	`public int calculateCreditHours()`
	`{`
	`int count = 0;`
	`for (int x = 0; x < subjects.Count; x++)`
	`{`
	`count = count + subjects[x].creditHours;`
	`}`
	`return count;`
	`}`
	
	`public bool AddSubject(Subject s)`
	`{`
	`int creditHours = calculateCreditHours();`
	`if(creditHours + s.creditHours <= 20)`
	`{`
	`subjects.Add(s);`
	`return true;`
	`}`
	`else`
	`{`
	`return false;`
	`}`
	`}`
	
	`public bool isSubjectExists(Subject sub)`
	`{`
	`foreach (Subject s in subjects)`
	`{`
	`if (s.code == sub.code)`
	`{`
	`return true;`
	`}`
	`}`
	`return false;`
	`}`
		This code
		● Includes member functions in the **degree program** class for adding **isSubjectExists** and **adding Subjects** and **caculateCreditHours().**

3.	`public void calculateMerit()`
	`{`
	`this.merit = (((fscMarks / 1100) * 0.45F) + ((ecatMarks / 400) * 0.55F)) * 100;`
	`}`
		Complete the **Student Class** by including the member function for performing the following tasks.
		● Merit Calculator
		● Registering Subjects for

	`public bool regStudentSubject(Subject s)`
	`{`
	`int stCH = getCreditHours();`
	`if (regDegree != null && regDegree.isSubjectExists(s) && stCH + s.creditHours <= 9)`
	`{`
	`regSubject.Add(s);`
	`return true;`
	`}`
	`else`
	`{`
	`return false;`
	`}`
	`}`
		students

3(a)	`public int getCreditHours()`
	`{`
	`int count = 0;`
	`foreach (Subject sub in regSubject)`
	`{`
	`count = count + sub.creditHours;`
	`}`
	`return count;`
	`}`
	
	`public float calculateFee()`
	`{`
	`float fee = 0;`
	`if (regDegree != null)`
	`{`
	`foreach (Subject sub in regSubject)`
	`{`
	`fee = fee + sub.subjectFees;`
	`}`
	`}`
	`return fee;`
	`}`
		Complete the **Student Class** by including the member function for performing the following tasks.
		● getCreditHours
		● calculateFee

Let us now implement the **Static Functions** (in the program.cs file) for this project.		

<table>
<tr>
<td>4.</td>
<td>

```
static Student StudentPresent(string name)
{
    foreach (Student s in studentList)
    {
        if (name == s.name && s.regDegree != null)
        {
            return s;
        }
    }
    return null;
}

1 reference
static void calculateFeeForAll()
{
    foreach (Student s in studentList)
    {
        if (s.regDegree != null)
        {
                Console.WriteLine(s.name + " has " + s.calculateFee() + " fees");
        }
    }
}
```

</td>
<td>

Implement functions for

- Checking if a student exists in the list of students
- A function to show the “calculated fee” of all the students.

Note: The function call inside the **calculateFeeForAll()** function, written as **s.calculateFee()** is actually calling the function inside the **Student Class**.

</td>
</tr>
<tr>
<td>5.</td>
<td>

```
static void registerSubjects(Student s)
{
    Console.WriteLine("Enter how many subjects you want to register");
    int count = int.Parse(Console.ReadLine());
    for (int x = 0; x < count; x++)
    {
        Console.WriteLine("Enter the subject Code");
        string code = Console.ReadLine();
        bool Flag = false;
        foreach (Subject sub in s.regDegree.subjects)
        {
            if (code == sub.code && !(s.regSubject.Contains(sub)))
            {
                s.regStudentSubject(sub);
                Flag = true;
                break;
            }
        }
        if (Flag == false)
        {
            Console.WriteLine("Enter Valid Course");
            x--;
        }
    }
}
```

</td>
<td>

This code implements a function to allow users to **register any number of subjects** as they want.

</td>
</tr>
</table>

<table>
<tr>
<td>6.</td>
<td>

```
static List<Student> sortStudentsByMerit()
{
    List<Student> sortedStudentList = new List<Student>();
    foreach (Student s in studentList)
    {
        s.calculateMerit();

    }
    sortedStudentList = studentList.OrderByDescending(o => o.merit).ToList();
    return sortedStudentList;
}

1 reference
static void giveAdmission(List<Student> sortedStudentList)
{
    foreach (Student s in sortedStudentList)
    {
        foreach (DegreeProgram d in s.preferences)
        {
            if (d.seats > 0 && s.regDegree == null)
            {
                s.regDegree = d;
                d.seats--;
                break;
            }
        }
    }
}
```

</td>
<td>

This code creates functions for the following operations

- Sorting the **student list** based on the **student merit**
- Giving admission to user and setting the value of Data Member **regDegree**

</td>
</tr>
<tr>
<td>7.</td>
<td>

```
static void printStudents()
{
    foreach (Student s in studentList)
    {
        if (s.regDegree != null)
        {
            Console.WriteLine(s.name + " got Admission in " + s.regDegree.degreeName);
        }
        else
        {
            Console.WriteLine(s.name + " did not get Admission");

        }
    }
}

static void clearScreen()
{
    Console.WriteLine("Press any key to Continue..");
    Console.ReadKey();
    Console.Clear();
}
```

</td>
<td>

This code implements the functionality for

- **Printing** all the students who got admission as well as those who failed.
- Function to **clear screen**.

</td>
</tr>
</table>

8.	```
	static void viewStudentInDegree(string degName)
	{
	Console.WriteLine("Name\tFSC\tEcat\tAge");
	foreach (Student s in studentList)
	{
	if (s.regDegree != null)
	{
	if (degName == s.regDegree.degreeName)
	{
	Console.WriteLine(s.name + "\t" + s.fscMarks + "\t" + s.ecatMarks + "\t" + s.age);
	}
	}
	}
	}
	static void viewRegisteredStudents()
	{
	Console.WriteLine("Name\tFSC\tEcat\tAge");
	foreach (Student s in studentList)
	{
	if (s.regDegree != null)
	{
	Console.WriteLine(s.name + "\t" + s.fscMarks + "\t" + s.ecatMarks + "\t" + s.age);
	}
	}
	}
	``
	_____
		Functions to
		• View the **registered students** in the system
		• **View registered students** in a **specific degree**

9.	```
	static void addIntoDegreeList(DegreeProgram d)
	{
	programList.Add(d);
	}
	static DegreeProgram takeInputForDegree()
	{
	string degreeName;
	float degreeDuration;
	int seats;
	Console.Write("Enter Degree Name: ");
	degreeName = Console.ReadLine();
	Console.Write("Enter Degree Duration: ");
	degreeDuration = float.Parse(Console.ReadLine());
	Console.Write("Enter Seats for Degree: ");
	seats = int.Parse(Console.ReadLine());
	
	DegreeProgram degProg = new DegreeProgram(degreeName, degreeDuration, seats);
	Console.Write("Enter How many Subjects to Enter: ");
	int count = int.Parse(Console.ReadLine());
	for (int x = 0; x < count; x++)
	{
	degProg.AddSubject(takeInputForSubject());
	}
	return degProg;
	}
	``
	_____
		Functions for
		• **Creating new degree**
		• **Adding a degree** into the **Program List**

10.	
	`static Subject takeInputForSubject()`
	`{`
	`  string code;`
	`  string type;`
	`  int creditHours;`
	`  int subjectFees;`
	`  Console.Write("Enter Subject Code: ");`
	`  code = Console.ReadLine();`
	`  Console.Write("Enter Subject Type: ");`
	`  type = Console.ReadLine();`
	`  Console.Write("Enter Subject Credit Hours: ");`
	`  creditHours = int.Parse(Console.ReadLine());`
	`  Console.Write("Enter Subject Fees: ");`
	`  subjectFees = int.Parse(Console.ReadLine());`
	`  Subject sub = new Subject(code, type, creditHours, subjectFees);`
	`  return sub;`
	`}`
	`static void addIntoStudentList(Student s)`
	`{`
	`  studentList.Add(s);`
	`}`
		Functions for
		• **Creating** new **Subject**
		• Adding student to **Students List**

11.	
	`static Student takeInputForStudent()`
	`{`
	`  string name;`
	`  int age;`
	`  double fscMarks;`
	`  double ecatMarks;`
	`  List<DegreeProgram> preferences = new List<DegreeProgram>();`
	`  Console.Write("Enter Student Name: ");`
	`  name = Console.ReadLine();`
	`  Console.Write("Enter Student Age: ");`
	`  age = int.Parse(Console.ReadLine());`
	`  Console.Write("Enter Student FSc Marks: ");`
	`  fscMarks = double.Parse(Console.ReadLine());`
	`  Console.Write("Enter Student Ecat Marks: ");`
	`  ecatMarks = double.Parse(Console.ReadLine());`
	`  Console.WriteLine("Available Degree Programs");`
	`  viewDegreePrograms();`
		Function for
		• Creating a new student by taking information from the user

```
        Console.Write("Enter how many preferences to Enter: ");
        int Count = int.Parse(Console.ReadLine());
        for (int x = 0; x < Count; x++)
        {
            string degName = Console.ReadLine();
            bool flag = false;
            foreach (DegreeProgram dp in programList)
            {
                if (degName == dp.degreeName && !(preferences.Contains(dp)))
                {
                    preferences.Add(dp);
                    flag = true;
                }

            }
            if (flag == false)
            {
                Console.WriteLine("Enter Valid Degree Program Name");
                x--;
            }
        }
        Student s = new Student(name, age, fscMarks, ecatMarks, preferences);
        return s;
    }
```

12.

```
    static void viewDegreePrograms()
    {
        foreach (DegreeProgram dp in programList)
        {
            Console.WriteLine(dp.degreeName);
        }
    }
    static void header()
    {
        Console.WriteLine("****************************************");
        Console.WriteLine("                    UAMS                ");
        Console.WriteLine("****************************************");
    }

    static void viewSubjects(Student s)
    {
        if (s.regDegree != null)
        {
            Console.WriteLine("Sub Code\tSub Type");
            foreach (Subject sub in s.regDegree.subjects)
            {
                Console.WriteLine(sub.code + "\t\t" + sub.type);
            }
        }
    }
```

Functions for

- **View all degrees**
- **View Subjects**
- Print **Header**

13.	`static int Menu()`
	`{`
	`header();`
	`int option;`
	`Console.WriteLine("1. Add Student");`
	`Console.WriteLine("2. Add Degree Program");`
	`Console.WriteLine("3. Generate Merit");`
	`Console.WriteLine("4. View Registered Students");`
	`Console.WriteLine("5. View Students of a Specific Program");`
	`Console.WriteLine("6. Register Subjects for a Specific Student");`
	`Console.WriteLine("7. Calculate Fees for all Registered Students");`
	`Console.WriteLine("8. Exit");`
	`Console.Write("Enter Option: ");`
	`option = int.Parse(Console.ReadLine());`
	`return option;`
	`}`
		function to print the **main menu**

Let us now implement the **Main Driver Program** for this project.		

14.	`public class Program`
	`{`
	`static List<Student> studentList = new List<Student>();`
	`static List<DegreeProgram> programList = new List<DegreeProgram>();`
	`static void Main(string[] args)`
	`{`
		Create the following global lists.
		• List for all Students
		• List of all Programs

14(a)	`static void Main(string[] args)`
	`{`
	`int option;`
	`do`
	`{`
	`option = Menu();`
	`clearScreen();`
	`if (option == 1)`
	`{`
	`if (programList.Count > 0)`
	`{`
	`Student s = takeInputForStudent();`
	`addIntoStudentList(s);`
	`}`
	`}`
	`else if (option == 2)`
	`{`
	`DegreeProgram d = takeInputForDegree();`
	`addIntoDegreeList(d);`
	`}`
		Implement the Main Menu

14(b)

```
else if (option == 3)
{
    List<Student> sortedStudentList = new List<Student>();
    sortedStudentList = sortStudentsByMerit();
    giveAdmission(sortedStudentList);
    printStudents();
}
else if (option == 4)
{
    viewRegisteredStudents();
}
else if (option == 5)
{
    string degName;
    Console.Write("Enter Degree Name: ");
    degName = Console.ReadLine();
    viewStudentInDegree(degName);
}
```

14(c)

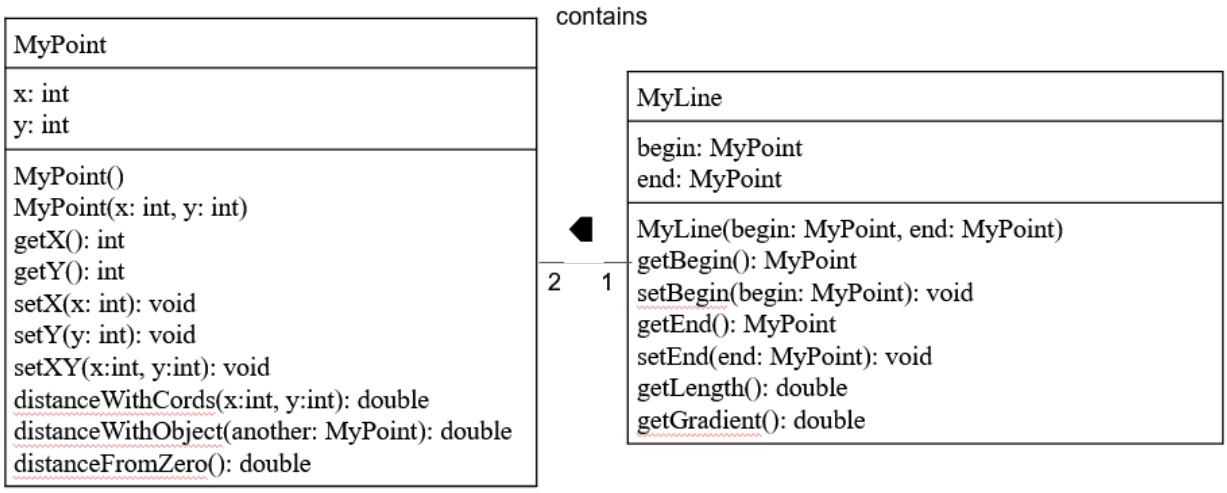
```
        else if (option == 6)
        {
            Console.Write("Enter the Student Name: ");
            string name = Console.ReadLine();
            Student s = StudentPresent(name);
            if (s != null)
            {
                viewSubjects(s);
                registerSubjects(s);
            }

        }
        else if (option == 7)
        {
            calculateFeeForAll();
        }
        clearScreen();
    }
    while (option != 8);

    Console.ReadKey();
}
```

Self Assessment Task 02: Identify the data that is common for all the objects of a class. Make that data and functions on that data **static** and include them in the respective classes.
Note: You will have to make a few changes to your code after performing the said task.

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

There are a few challenges given ahead. Have a try at those. Good Luck :)

Challenge 01:



A class called **MyPoint**, which models a 2D point with x and y coordinates, is designed as shown in the class diagram.

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.
- A method called distanceWithCords(int x, int y) that returns the distance from this point to another point at the given (x, y) coordinates.
- A method distanceWithObject(MyPoint another) that returns the distance from this point to the given MyPoint instance (called another).
- Another method distanceFromZero() method that returns the distance from this point to the origin (0,0)

A class called **MyLine**, which models a line with a begin point at (x1, y1) and an end point at (x2, y2), is designed as shown in the class diagram. The MyLine class uses two MyPoint instances (written in the earlier exercise) as its begin and end points. Write the MyLine class. Also write a test driver to test all the public methods in the MyLine class. Use distance formula to calculate the length of the line

Distance formula: $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

The gradient of a straight line is denoted by m where:

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

Menus on the Console. (This is only for 1 line, do not make a list)

1. Make a Line
2. Update the begin point
3. Update the end point
4. Show the update Point
5. Show the end point
6. Get the Length of the line
7. Get the Gradient of the Line
8. Find the distance of begin point from zero coordinates
9. Find the distance of end point from zero coordinates
10. Exit

Challenge 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%

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. MUser (or credentials) 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. 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. Exit

Good Luck and Best Wishes !!

Happy Coding ahead :)